ISSN 1346-7301
国総研研究報告　第６２号
平 成　３１ 年　２ 月

国土技術政策総合研究所
研究報告

RESEARCH REPORT of National Institute for Land and Infrastructure Management
No. 6 2　　February 2019

地域居住空間の三次元アーカイブスの利活用

小林 英之

How to Make and Use 3D Archives of Settlements

Hideyuki KOBAYASHI

国土交通省　国土技術政策総合研究所

National Institute for Land and Infrastructure Management
Ministry of Land, Infrastructure, Transport and Tourism, Japan

国土技術政策総合研究所研究報告　　Research Report of NILIM
第６２号　2019年 2月　　No.62 February 2019

地域居住空間の三次元アーカイブスの利活用

小林　英之　*

How to Make ans Use 3D Archives of Settlements

Hideyuki KOBAYASHI　*

概要

集落・団地などの住居群とコミュニティインフラの形成過程、災害と復興等を立体的に記録し長期保存する三次元アーカイブスの構築方法と利活用方法を解説する。

ｷｰﾜｰﾄﾞ　：　三次元データファイル、メタファイル、コンパイラ

Synopsis

This document describes how to make and use 3D Archives of Settlements which permanently record 3D space of houses and common facilities that have been developed, damaged and re-constructed.

Key Words : 3D Data File, Meta File, Compiler

* 研究官　Researcher

はしがき

地域の居住空間の形成過程、とりわけ初期の開発過程や、災害による壊滅と復興に関しては、様々な手段で現代にまで伝えられている。残された記録は、後世の市民や研究者からは「歴史資料」として扱われ大切にされる。

伝統的な記録の形態としては、土の中に埋蔵された遺跡（地物の現物や、それを転写した土の形）、保存された歴史的建造物、石碑（建物に設置される定礎もこの一種）や木簡（木造建築の棟札もこの一種）、紙の上に記録された古文書（文字や絵図・建築図面）、模型、古写真や映画フィルムなどがある。

一方、1970 年代から、デジタル情報として居住空間を記述する方法が実用化され、デジタル映像に加えて、CAD データ、GIS データなどの地理空間情報が、設計計画に係るコミュニケーションの手段および生産の手段として広く活用されるようになった。具体例を挙げれば国土交通省版・景観シミュレーション・システムについても、設計された土木建築施設の完成後の姿を様々な角度から立体的に確認するための道具として使用されてきた。

このことはインターネット等の通信手段の発達に伴って、同時代的な情報交換のために大きく寄与してきた。しかしながら、これらの技術を、情報の長期保存のための手段として改めて見直した時に、二つの大きな課題が残されている。

一つは、前提となる技術的条件、とりわけ通信速度や扱えるデータ量が大きくなり、それに伴って使用される記録形式が発達してきた。このため、古いデータの保存形式が陳腐化し、これを読み込んで表示するようなシステムが利用できなくなる場合が生じている。

もう一つは、物理的にデータを保存するための商品としての媒体が変化してきた。媒体やそれを読み書きする装置（ドライブ）の寿命が必ずしも長くはないため、記録を読み出して利用することが困難となる。

従って、過去から未来に記録を残し伝えていく、という目的のためには、記録形式の陳腐化と、記録媒体の劣化により可読性が失われる、課題を解決しなければならない。

少なくとも建築物の当初建設時点で主として設計施工のための目的のために作成されたデータを、維持管理のために引き続き使用するためには、そのデータは建物の寿命である数十年から長いものでは数百年の期間にわたって保存でき、利活用できる状態が維持できなければ無意味なのであって、現在広く使用されている各種ディスク、IC チップ等のように数十年で劣化し、維持するためにはメディア更新を続けなければならないとすれば信頼性は得られない。さらに、防災対策等の目的と内容に関して、地域の災害記録が場合によっては 1000 年程度の将来期間に亘り教訓として維持できなければ、所期の機能を果たせないというような事態が生じうることは、容易に想像できる。

伝統的な紙などの媒体の上に記録された文書、絵、地図などは、500 年以上保存されてきた例も多く、またこれらを保存するための博物館等の社会的な制度が整備されている。

しかしながら、とりわけ土木建築施設等の立体的形状を記録したデータの場合、紙の上に全体を印刷することができない。無論、バイナリデータのダンプリストを紙に印刷する方法や、これらをコンパクトに圧縮した二次元バーコード等の形式で印刷することは可能であるが、可読性は著しく低い。この特徴は、動画、音源等のデジタルデータの保存の問題と共通である。

法律や制度によって裏付けられた公共データの場合には現在、データセンターにデジタルデータを置いて組織的に保存しているが、個々の住宅レベルの個人データに関しては、そのような体制は未整備であり、また実行するためには、稀にしかアクセスしないコールドデータを維持するために厖大な電力を消費することとなる。

かかる課題を解決するために国土技術政策総合研究所（以下、「国総研」という）において、保存すべきデータを解読するための完全な手順を記述したメタファイルを作成するためのスクリプト言語を開発した。これに従って作成したメタファイルを保存データに添付して、寿命の長い記憶媒体に保存することにより、将来の可読性を保つことができる。

更に、記録データを利活用しようとする将来のユーザのために、４種類の例示的な利活用処理系を試作した。更に、いくつかの保存する価値のある、既に失われた地域居住空間に関する記録を元に保存データを作成し、これらの例示的な処理系を用いて再生が可能であることを検証した。

現在のユーザは本書の解説に従って、記録データに手を加えることなく、その解読手順を指示したメタファイルを添付して保存措置を講じる。例示的に作成した現在の利活用処理系は、この保存工程を支えると共に、保存データ＋メタファイルの有効性を検証することが可能である。

更に、本書は現在のユーザのみならず、将来のユーザをも対象として執筆したものである。すなわち将来のユーザが、上記の方法で記録保存された記録データと添付されたメタファイルを手にした時に、これらを用いて利活用処理系を再構築し、コンパイラを用いてメタファイルから解読処理を行うプログラムを生成し、それを用いて記録データを再生することができるよう、システム開発者の手引きとなる資料も含んでいる。

第１章は、国総研の課題として行われた研究開発の背景と経緯を記録したものである。研究初期の段階で各種記録形式、記憶媒体、既存技術等の調査を行い、これに基づいて試作的なシステムを構築し、20 年以上遡る古写真や記録図面から復原した実際の三次元データを用いて、遠い将来の利用状況を想定した体験教室を行うに至った経緯を解説した。

第２章は、これらの技術を用いて、各地域の過去の保存データあるいはこれから保存しようとするデータを検証しようとする現在のユーザのための解説である。ユーザは、システム開発者から、一般市民や学童に至る最終的な利用者まで様々な階層が想定されるため、本システムのどの側面と関わるかによって、書き分けられている。

第３章は、本書の方法を用いて作成された記録（アーカイブス）を利活用するための、今はまだ無いシステムを開発しようとする将来のプログラマ等を対象とするものであり、４種類の異なる利活用形態を開発した具体的事例に即して解説している。本章の記述には、情報処理装置やプログラム開発環境に関して 2015 年時点では常識であった細かな技術的な情報

も含まれている。これらは、同時代的にはあまり価値がない記述も含んでいるが、数十年先には、遠い過去のものとなっている可能性が高いと考え、記録として残しておくこととした。

第４章は、本文を補足した、やや詳細な解説資料や記録を収録したものである。

デジタル・アーカイブスという言葉は既に広く用いられるようになったが、多くの場合には長期保存が担保された原本（古文書、古美術、古建築等）が既にあって、そこから発達著しい各種モデリング技術を駆使して作成された二次的な、多くは広報目的のシステムであって、データそのものが保存の対象ではない。しかしながら、今後を考えるとオリジナルの記録が当初からデジタルデータであって、それ自体の長期保存が求められる状況になることは疑いない。とりわけ、建築物などの大規模な構造物を記録した三次元データの場合には、紙にプリントした平面図・立図面などの二次元記憶媒体による記述には限界があり、本書で提示したようなアプローチが必要になると考える。

国土交通省国土技術政策総合研究所

目 次

図版一覧　　（掲載頁）

写真一覧

リスト一覧

表一覧

１．研究の経緯

はじめに

本書で解説する三次元アーカイブスの技術開発と適用事例は、平成 22-26(2010-14)年度の５カ年に亘り国土技術政策総合研究所（以下、「国総研」と略記）が実施した二つの研究課題の成果である。最初の３年間に実施した「三次元住宅情報の永久保存技術に関する研究」（１－１）では、土木建築施設を記録した三次元データの長期保存に関する過去の技術のレビューと、提案的なシステムの試作を行った。これに続く２年間に実施した「地域居住空間の三次元アーカイブスの利活用」（１－２）では、長期保存用サンプルデータの作成と、これを用いたいくつかの利活用形態のデモ、テストを行い、最終的な成果として三次元アーカイブスの作成と利活用の事例報告と、本書に掲載した様々の技術資料を作成した。更に、本研究の成果の内、技術的内容（アルゴリズム等）を「明細書」の形で文書化した上で、これを記録保存すると共に同一分野の既存技術に対する進歩性を明らかにすることを主目的として、本研究に関係する４件の特許出願を行い、全て登録されている。

１－１．三次元住宅情報の永久保存技術に関する基礎的研究

平成 22～24(2010-12)年度に、住宅等に関して作成された CAD データ、GIS データ等を、建物の寿命を超えて長期保存するために利用可能な既存技術、新たに解決すべき技術的課題を調査・研究し、提案的な技術開発を行った。

国総研の前身である建設省建築研究所第一研究部と土木研究所環境研究部において、1993 年度から景観シミュレーションの技術開発が行われ、土木建築施設や町並に関する三次元データが作成・蓄積されたが、Micro Station、Mini Cad、City Sight、Attract、Auto Cad、Studio 3D max 等の商用 CAD ソフトで入力された建物や点景のデータ、ステレオ空中写真から作成した地形データ（DEM）等のデータ形式（フォーマット）は大きく変遷し、その都度コンバータを作成して入力・編集を行った。この間、上記のような三次元データを扱う商用ソフトウェアの側でも、各種形式の入出力機能を備えるのが一般的となった。一方、1990 年代前半のデータ形式（主にプロッタ等を駆動して線画による図面を作成することを目的としていた）は古い過去のデータ（レガシーデータ）となっていった。

これと共に、記録媒体も、データカートリッジやフロッピーディスクから MO、CD-ROM、DVD 等に急速に変化し、古い媒体に記録されたデータは媒体の劣化やドライブの製造中止により利活用が困難な状況になった。

三次元データを地元説明会などに一時的に利用する場合には、最新のシステムと一時的な記録媒体を使用すれば目的を達することができる。しかし「ライフサイクルに亘って」と謳う CALS/EC(Continuous Acquisition and Life-cycle Support/Electronic Commerce)の本来の理念のように、建築物等を記録した各種データを、記録対象である建築物等の寿命を超えた長期にわたり利活用するためには、データ形式の変遷（陳腐化）と短命な記録

媒体は大きな障害となる。これに対処した最新のデータ形式への変換と新たな記録媒体への継続的な載せ替えは、データストックの増大に伴い大きなコストと手間を必要とする。

そこで本研究においては、従来の各種技術の変遷を調査するとともに、将来長期に亘る記録データの利活用を図るためのローコストで手間のかからない技術的手段を提案することを目的とした。

過去の記録形式や媒体に代わり、今後作成する物件に適用される新たなデータ形式や新しい記録媒体の採用を提案することと、その新たな形式や媒体の永続性を主張することは矛盾する。

（１）平成 22(2010）年度

①記録媒体に関する調査研究

旧建設省建築研究所において、1970 年代以来各種のデジタルデータを保存するために使用されてきた記録媒体について発展過程を調査し、媒体と読み書きのための装置（ドライブ）等のサンプルを収集した。これには、パンチカード、磁気テープ、データカートリッジ、フロッピーディスク（８インチ、５インチ、3.5 インチ)、光磁気ディスク MO（５インチ、3.5 インチ)、DAT、CD-R、DVD-R、コンパクトフラッシュ、SD カードなどが含まれる。

②データ形式に関する調査

本研究に先行して建設省建築研究所・土木研究所において開発した景観シミュレーション・システム(1993-)の運用に際して、様々の CAD データ、GIS データの景観検討への活用のために、ファイルコンバータが作成され、C 言語によるプログラムの蓄積があった。

その後産業分野において設計過程で作成される CAD データを製造工程 CAM で活用するためのデータ形式が急速に発達し、建築分野にも影響があった。土木・建築分野では、CALS プロジェクト以来のデータ標準化の努力の結果、IFC 形式が提案され、三次元データの上に物性や価格などの様々な属性データを載せてワークフローに乗せる BIM(Building Information Model)の技術が普及しつつあった。

米グーグル社が無償公開した Sketch Up(2006)等のモデラーと、それを用いて三次元入力した建物等を Google Earth 全球地図の上にプロットして町並を構築するような WEB システムが提供され、データ形式として XML をベースとした KML、それを圧縮した KMZ 等が提案されたが、それにも関わらずさらに新しいデータ形式の開発・提案も続いた。

地理分野においては、ステレオ空中写真から作成した DEM（mem 形式）に加えて、航空機搭載型のレーザー・スキャナを利用した高精度の地形データが数値地図（lem 形式）として供給され、さらにこれらを XML 形式に再編した基盤地図情報が国土地理院により開発・公開された。

各種の三次元データ形式を調査し、広く使われているデータ形式に関して、定義書（フォーマット解説資料）とサンプルデータの収集を行った結果を表１－１に示す。

このような歴史的過程の中で、コンバータなどを用いてデジタルデータを同時代的に共

有する技術は普及してきたが、様々な過去のレガシーデータ形式で記録されている建物の形状などの記録保存のための技術は不十分である。ある形式のデータの定義書・解説書が保存されていても多くの場合、そこに記載されている内容は多岐にわたる一方で、実際の

表 1-1-1 仕様書とサンプルデータを収集した三次元データ形式

拡張子	名称	開発者	開発年
IGS(IGE,IGES)	IGES (Initial 2D/3D Graphics Exchange Specifications)	National Bureau of Standards, USA	1980
OBJ	Wavefront Object	Wavefront Technologies, USA	1985
3D2 (3D, 3D4)	Stereo CAD-3D Object	Atari, USA	1985
OFF (COFF, NOFF, CNOFF, STCNOFF)	3D Object File Format	DEC (Digital Equipment Corporation), USA	1986
SAT (SAB)	ACIS Model	Spatial Corp, USA	1989
STL (STLB)	Stereo Lithography	3D Systems Inc., USA	1989
3DS (MLI)	3D Studio Scene	Autodesk, USA	1990
VTK(VTHB, VTR, VTI, VTS, VTU, PVTS, PVTU, PVTI)	Visualization Toolkit	KitWare Inc., USA	1993
LWO(LWS, LS, RIG, IFF, FPBM, LW2, LWOS)	Lightwave 3D Object	New Tek, USA	1994
FLT	Open Flight Scene	MultiGen Inc., USA	1994
PLY	Stanford polygon format	Stanford Graphics Lab, Stanford Univ., USA	1995
3DMF	3D Metafile	Apple, USA	1995
COB(SCN, SOB)	Caligari true Space	Caligari Corp., USA	1995
MGF	Materials and Geometry Format	Lawrence Berkeley Laboratory, USA	1995
C4D	Cinema 4D	MAXON Computer GmbH, Germany	1996
3DM	Rhinoceros 3D Model	Rhinoceros, USA	2000
DAE	COLLADA	Khronos Group Consortium	2004
SKP (KMZ)	Google Sketch Up	Google Inc., USA	2006

サンプルデータはその仕様の一部しか使用していない場合や、コメントの中に仕様外の意味のある情報を追記しているような場合も見られる。

近年の傾向として、XML をベースとしたデータ形式が増加した。XML 形式のデータの場合、既存の汎用ソフトウェアによってファイルを読み込みメモリ上のオブジェクトとして解析することによりプログラミングは容易となる。しかし、巨大なデータをすべて読み込むためには大容量のメモリを必要とすることや、一部破損したようなデータを解析することが困難になる点は注意を要する。

③長期保存を使命とする機関の調査

省エネ基準に適合した北方型住宅を登録する有償サービスを提供している北海道建築指導センター、統計データを保存している総務省統計局、CD-ROM 等を付録する書籍を保存している国会図書館、地質データを保管している産業技術総合研究所地質調査総合センターなどにおいて実際のデータの記録保存に使用されている記憶媒体に関する調査を行った。

④長期保存に資する先行技術（特許等）の調査

デジタルデータの長期保存のための記憶媒体・装置に関する先行技術、および記録形式に関して、特許庁の特許電子図書館および発明通信社の検索システム HYPAT-I において先行技術の調査を行った。

記録媒体・装置に関しては、検索条件を、

｛３次元 or 三次元 or ３D｝and
｛情報 or データ or 図形 or 画像｝and
｛記録媒体 or 記憶媒体 or 保存媒体 or 記録装置 or 記憶装置 or 保存装置｝and
｛長期｝and
｛建物 or 建築 or 住宅 or 家 or 景観｝

とした。また、記録形式に関しては、検索条件を、

｛３次元 or 三次元 or ３D｝and
｛情報 or データ or 図形 or 画像｝and
｛記録 or 記憶 or 保存｝ and
｛形式 or ビットストリーム｝and
｛長期｝and
｛建物 or 建築 or 住宅 or 家 or 景観｝

とした。最後の｛建物 or 建築 or 住宅 or 家 or 景観｝という条件を外すといくつかの先行特許文献が検索されたため、具体的に明細書を取得して検討を行った。

その結果、本研究が課題とする、建物等を記録した三次元データの長期保存に適用可能な技術は、少なくとも国内特許文献の中には存在しないことが判明した。

これらの調査結果を含め、本研究に関する情報を交換するための WEB サイトを、まちづくり・コミュニケーション・システムを稼働している国総研のサーバー上にコーナーとして開設した(2011.2.1)。

http://sim.nilim.go.jp/MCS/phi/phi.asp

⑤ソフトウェア資産の整理と出版

1993 年以来、建設省土木研究所・建築研究所および国総研において、景観シミュレーションのために開発してきた各種プログラムと、データ処理のアルゴリズムを解説した国土技術政策総合研究所研究報告第 42 号「国土交通省版・景観シミュレーション・システム ver.2.09 のアーキテクチャ」を出版した(2011.3.14)。

http://www.nilim.go.jp/lab/bcg/siryou/rpn/rpn0042.htm

このアーキテクチャにおいては、新たなデータ形式を用いたデータの入力は、独立した実行形式として追加することができる。この方法は、パラメトリックな部品を実装する手段として開発された「外部関数」を拡張し、引数としてデータファイルを指定し、そのファイルを変換するコンバータを実行形式とすることにより実現している。

さらに、画面に表示されている建物やその構成要素を選択した上で、あるデータ形式で出力する機能も、プラグイン dll として追加できるようにした。

⑥特許出願

特に、三次元図形演算処理のアルゴリズムと、処理を具体化したプラグイン dll に関して、特許出願した。(「三次元図形演算プログラム、ダイナミックリンクライブラリ及び景観検討装置」、出願番号 2011-052466,2011.3.10、特許第 5039978 号, 2012.7.20)

この出願は、当該システムの「製品」としての利用を拡大普及することよりもむしろ、汎用性の高い技術を文書化・公開し広く共有することを主な目的とした。

三次元図形演算を行う処理は、更に改良（成功率と処理速度の向上）を行う余地があることから、既に公開している国土交通省版・景観シミュレーション・システムに機能を追加するダイナミックリンクライブラリ(dll.dll)として、システム全体を再ビルドすることなしに単独でデバッグし改良することができる。さらに地形編集等を目的とした別のプラグインから利用したり、景観シミュレーション以外の目的のためのアプリケーションから利用したりできる独立性の高い形で提供した。

（２）平成 23(2011)年度

①コンパイラ処理系による解決

データ形式の標準化は、CALS プロジェクトとして１９９０年代から永く取り組まれているが、主な目的と成果は同時代的・共時的(synchronic)な情報交換とそれによる設計・生産の合理化にある。しかし、ある時代に広く普及したデータ形式も、技術革新に伴いやがて陳腐化し過去のものとなり、余り使われなくなる。このことは文書や表計算を始めあらゆるデータに関して生じる現象であるが、特に建物等を記録した三次元データに関しては、新たなデータ形式が絶え間なく提案され普及した結果、2010 年度の調査によれば、機械部品等も含めるとすでに 300 種類に及ぶデータ形式が存在することが判明した。即ち、過去に開発された様々なデータ形式に代わる新たな究極のデータ形式をグローバルな標準とし、以後その標準を永久不変のものとして使い続けるようにする目論見は常に失敗してきたと

いえる。

一方、建築物等を記録したデータを建物の維持管理、増改築や売買・賃貸借などの目的に活用しようとする場合、作成されてから数十年あるいは数百年の後であってもそのデータが可読でなければならない。一般には、データ作成時に使用されていたマシン（PC 等）のハードや OS と、その上のアプリケーションは、そのままの形で長期にわたり使用し続けることは期待できない。多くの CAD ソフト等は、同時代あるいは近過去に広く使用されているデータ形式による入出力の機能を有しているが、すでに過去のものとなった形式によるデータは利用が困難になる。

様々なデータ形式で作成された記録を利用するためには、コンバータを作成することが一般的である。しかし、コンバータ実行形式（例えば Windows 上の EXE ファイルや DLL ファイル）は、特定の OS の上で利用可能であり、前提となるハードと OS が変化・陳腐化していくために、コンバータの実行形式はいずれ使用できなくなる。但し、過去のマシンや OS をソフト的に再現してその上で過去のアプリを実行するための仮想化技術は開発されつつあり、例えば 1990 年代のワードプロセッサ専用機をソフト的に再現し、専用のフォーマットで保存されている文書ファイルを開いて編集することも、十分な技術史料が残されていれば可能である。

そこで本研究においては、すでに陳腐化したデータ形式（レガシー形式）により記録されている土木建築施設のデータの再生を可能とするために、記録に使用されたデータ形式の解読方法をプログラムとして記述したメタファイル（テキスト形式）を添付して保存しておき、利活用段階においてはまずこのメタファイルをコンパイルし、生成した実行形式を用いてデータを読み込み、表示等の処理を行う処理系を開発することとした。このような方法であれば、遠い過去に作成され長期保存されたデータであっても、添付されているメタファイルをコンパイルする部分が維持できれば、本体である記録データを開いて利用することができる。このような目的のためには、コンパイラ自体は、高速計算等のための高度な最適化を必要とせず、むしろシンプルで再現が容易なものが、将来の異なるマシンや OS への可搬性の観点から適している。

長期保存の観点からは、このようなコンパイラが、必ずしも社会的に統一化・標準化される必要も、また広く普及する必要もない（望ましいことではあるが）。重要な点は、保存時に解読方法を記述するために作成され添付されたメタファイルの文法（即ちコンパイラの動作定義）が、長期間経過した利活用時点においても適用可能であれば十分であるという点である。このためには、このコンパイラの完全な仕様書（ソースコード）を保存データに添付する方法のほかに、本書のような形で記録化し、国立図書館等の資料の永久保存を使命とするような機関に保存しておき、添付されたメタファイルがその仕様（バージョン）に対応していることが特定できるようにしておく、という方法も有効である。

また、このような処理系は複数存在していても構わない。例えば、二つの処理系に基づく２種類のメタファイルが記録データ原本に添付され保存されていれば、データ本体が将

来再利用可能であることはより確かとなる。

本書で解説するコンパイラでは解読処理を記述することが困難な新たなデータ形式が将来開発され、それに適合する新たなメタファイルの文法体系とコンパイラが開発されたとしても、本書の時点でのメタファイルと保存データのペアの記録としての有効性が損なわれることはない。

本研究においては、処理系の一つの実現例として、まずPC上のWindowsアプリケーションとして「仮想コンバータ」と称する、コンバータに目的を特化したコンパイラ処理系を試作し、携帯端末等の他のOSへの可搬性を検証する方法を試した。ここでいう「仮想」とは、いわゆる「仮想記憶」や「仮想マシン」などの概念と同様に、特定のハードやOSに束縛されない論理的なコンバータという意味であって、具現化・実装に当たっては必ずある特定の（様々な）ハードやOSを必要としている。

景観シミュレーション・システムが様々な形式のデータを利用するためのコンバータの内の１０種は、外部関数として旧建設省建築研究所、国総研がC言語で開発しWindowsのための実行形式(.exe)の形で提供してきた(表4-3-1)。仮にこれらコンバータのソースコードを直ちに実行するような処理系（コンパイラ）が実現できれば、原本データにコンバータのソースコードを添付したものを保存しておき、利活用時点で利用可能なＯＳの上で稼働するコンバータ実行形式をこのソースコードから生成して実行すれば事足りる。

このように目的を限定したＣ言語は、標準Ｃ言語よりもシンプルなものであっても十分である。メタファイル（コンバータのソースコード）を将来実行するためには、将来のマシン上でコンパイラが再現可能であれば十分なのであり、インテル系プロセッサとWindowsのために開発された従来のコンバータ実行形式の永続性を確保する必要はない。

データファイルに添付して保存する形式定義ファイル（メタファイル）が解読対象とする個別データは既に既知であるから、例えばメモリ管理においても不特定の長さのメモリブロックを取得・解放するような処理も不要であり、個別データの解読に十分な長さの配列を宣言できればよい。

さらに、これまでのコンバータ作成の経験に基づくならば、個々の三次元データは、記述に用いられた特定のデータ形式の仕様全体の内のごく一部の項目しか用いていない。従って、あるデータ形式を一つの言語に喩えるならば、個々のデータの解読方法の指示のためには、メタファイルにおいて、当該データの記述に使用されている文法や単語だけを含む辞書のサブセットを用意して添付すれば十分である。その言語によって書かれた全ての言説を理解するための厖大な辞書と文法解析を添付して保存する必要はない。

個別データの解読に十分な単語と文法だけを記述することができるような、あらゆる言語・言説に共通の方法（文法と辞書の記述方法）を不変のものとして用意しておけばよい。

言語仕様の開発方針：
・ある特定のデータ形式で記述されたあらゆるデータファイルを解読する手順のサブセットとして、保存対象である個別データの解読手順が定義できれば十分である。このため

には、**保存データの中で使用されている文法と辞書だけを記述できる**ことを目標とする。
・メタファイルを記述するためのルールが将来とも様々なマシン、ＯＳの上で実行できる可搬性を担保するため、コンパイラ VC の公開ソースコード（C）として固定化する。
・コンパイラ VC が解釈するメタファイルの記述ルールを、C 言語を簡略化した（仮称）C--言語と名付け、ユーザはメタファイルをこのルールに従って作成する。
・C-- 言語は、サブルーチン処理を行う**関数**を定義することができるようにする。
・システムがデータ解読と形状構築に際して汎用性の高い処理を、メタファイル中で定義する必要がない、予め用意された**組込関数**として仕様と動作を定義する。
・組込関数の一部として、三次元形状を構築するための**ライブラリ関数**を提供する。
・ライブラリ関数は仕様のみ**テンプレート**として定義し、コールされた時の実際のマシン動作は将来、個別の**利活用処理系**の中で様々に定義することができるようにする。
・保存工程で作成され添付されるメタファイルは、あらゆる利活用処理系に共通とする。

②コンソールアプリの試作

このような処理系を最短で試作し実際のデータを用いてその有効性を検証するために、コンパイラの基幹部分に関しては、林(2010)により提供されているシンプルな C コンパイラを、出版社の許可を得て拡張することで実現した。この C コンパイラは、シンプルな命令から成る仮想マシン上の実行形式を生成する（３－２参照)。ソースコードは、C 言語で記述されている。変数型としては整数のみを扱う。

本処理系を実現するための拡張として、三次元座標値を扱うことができるように、変数型として、浮動小数および四元数を追加し、これらの四則演算を行う処理を追加した。

C 言語においては、関数はプログラム（本処理系の場合には、メタファイル）の中で定義するが、この他にコンパイラ（ソースコード）の中で予め定義され、ユーザープログラムから定義なしに直接呼び出すことができる組み込み関数を用意することができる。そこで、三次元図形の定義に必要な処理を行うための組み込み関数を、「ライブラリ関数」として追加した。このライブラリ関数は、LSS-G 形式のコマンド群の殆どを含むものとし、引数としては定数だけではなく変数や式を用いることができるようにした。

一例として、最も基本的な座標値を定義するための COORD 関数について解説する。

LSS-G 形式のコマンド

```
P0=COORD(1.0,2.0,3.0);
```

の１行は、三次元座標（1.0, 2.0, 3.0）を表すオブジェクト P0 を定義する。引数として記述する座標値は固定値（浮動小数）である。以下、面の定義などには P0 というラベルを参照する。

一方、本処理系におけるコンパイラ（仮想コンバータＶＣ）においては、ライブラリ関数の、メタファイル中でのライブラリ関数呼び出しとしてこの行を解釈・実行するため、

```
P0=COORD(x, y+2.0, z*3);
```

という表現で、引数である座標値を変数や数式により定義することが可能である。

この１行が実行されると、引数の数式を計算した上で、３の数値がライブラリ関数 COORD に渡され実行される。COORD 関数は、何らかの処理を行った上で座標値に対応するユニークなＩＤ整数値を返し P0 変数に代入する。

COORD 関数が呼び出された時の処理内容は利活用目的に様々に定義できるが、例えば最も簡単な処理として、このコンソールアプリ VC-1C の処理系では x が 0.0、y が 1.0、z が 3.0 であれば、出力ファイルに

“P0=COORD(0.0, 3.0, 9.0);”

という文字列１行、すなわちメタファイルの行とほぼ同一の形式で数式が演算された結果を引数とするコマンド文字列を出力し、戻り値としてこの三次元座標値に固有の整数値を返す。この戻り値は整数型の変数 P0 に代入される。

同様に LSS-G 形式のコマンドに対応する３０のライブラリ関数を実装した。コンパイラ処理系に組み込み関数を追加した手順を３－２(2)に、また個々のライブラリ関数の利活用処理系に応じた処理内容を定義した方法を３－２(4)に、ライブラリ関数の一覧を表３－２に、関数毎の詳細を４－２で解説する。

この最もシンプルな処理系であるこのコンソールアプリ VC-1C は､具体的な動作として、LSS-G 形式の三次元データをメタファイルとしてコンパイルすることができ、実行段階では同じ図形を再定義する LSS-G 形式のファイルを出力するように動作する。この処理は、コンバータとしては実用的な意味を持たないが、変換元と同じ内容（形状・色彩等）のデータが生成していない場合、データ内部の詳細な比較を行うことなしに画面上で同じものが表示されることをビューワ（景観シミュレータ）で目視確認することができ、これにより、処理系の基本的な部分のテストとデバッグ（ライブラリ関数の動作確認と、メモリーリークの解消など）を、LSS-G 形式の様々なサンプルを用いて行うことができた（３－３で VC-1C を解説した)。

言い換えると、既存の LSS-G 形式のデータファイルは、これを特殊なメタファイルとしても読み込むことができ、指定されたデータファイルに関わりなく固定形状を記述したメタファイルとして利用することができる。

このコンソールアプリケーション vc-1c.exe は、３の引数をもち、コンソール(Windows 上の cmd.exe)から vc-1c とタイプすることにより起動することができる。この時３個の引数を指定する。第一引数はメタファイルの名称、第二引数は入力データファイルの名称、第三引数は出力データファイルの名称である。メタファイルが LSS-G 形式のファイルである場合、その中にはデータファイルを読み込むための scanf 等の関数は含まれておらず、第二引数で指定されたデータファイルは単なるダミーとしての意味しか持たない。

次に、データファイルの入力のための、scanf, getc 等の関数を実装し、DXF 等の各種形式のデータファイルを解読し、LSS-G 形式のファイルとして出力する処理を、メタファイ

ル（プログラム）として作成しテストした。様々なデータ形式を読み込むためのメタファイルを作成するために、従来景観シミュレーションのためのファイルコンバータとして蓄積し公開してきた C 言語のソースコードから出発することにより、開発の手間を省力化することができる。

従来のコンバータのソースコードの出力部分とメタファイルにおけるコーディングを比較すると、入力対象のデータファイルから三次元座標値を読み込み、x,y,z という浮動小数の３値を取得し、固有の id を生成する処理を行った後に、

```
fprintf(wp,"P%d=COORD(%f,%f,%f);¥n",id,x,y,z);
```

という１行で出力ファイルに１行を追記していたステップは、

```
P0=COORD(x,y,z);
```

というステップに置き換えて記述できることとなる。

また、同じこの実行形式を Windows 上で動作するコンソールアプリ vc-1c.exe として景観シミュレータの外部関数に追加登録すると、従来

```
G0=FILE(dxf2lss, sampledata.dxf);　（コマンドA）
```

というステップで、sampledata という dxf 形式のファイルを入力し G0 という名称のオブジェクトとして定義していた処理は、

```
G0=FILE(vc-1c, dxf.cmm, sampledata.dxf);　（コマンドB）
```

という行に置き換えることができる。

従来のコマンドＡの処理において、G0=FILE(dxf2lss, sampledata.dxf);というコマンドが実行されると、dxf2lss.exe が起動して、sampledata.dxf というデータファイルを解釈し、LSS-G 形式のファイルに変換して G0 という名称のオブジェクトを生成する。

これに対してコマンド B が実行されると、vc-1c.exe が起動して、まず第二引数の dxf2lss.cmm というメタファイルをコンパイラＶＣのためのソースコードとしてコンパイルし、DXF 形式専用のコンバータの実行形式を生成する。次のステップでこの実行形式を実行し、第三引数として指定された sampledata.dxf の解読処理を行い、その処理の結果生成されるファイルを読み込まれて G0 という名称のオブジェクトを生成する。

vc-1c.exe は Windows 上で実行するコンソールアプリであるが、第二引数のコンバータの入力処理を記述したメタファイルは dxf2lss.cmm は編集可能なテキストファイルであり、Windows 以外の処理系においても利用可能となる。異なる OS への移植は、vc-1c.exe だけを移植すればよく、様々なデータ形式の入力用コンバータは、第二引数となる入力処理を定義したメタファイルはマシンや OS に依存することのない可搬性が得られた。この処理系については、３－３で解説する。

③プラグイン DLL の試作

上記の COORD 関数等の各ライブラリ関数の関数型（引数と戻り値）を保ったまま処理内容を変えることにより、LSS-G 形式の文字列をファイルに出力する処理系 VC-1C を代替する、メモリ上にデータを構築し直接表示する処理系を実現することができる。そこで、

ライブラリ関数が画面表示可能なメモリ上の表示データを生成する処理系を景観シミュレータのプラグイン DLL として作成し動作を確認した（vc-2v.dll)。Windows を OS として動作するこの処理系においては、メタファイルとデータファイルを指定して実行すると、実行形式がデータファイルを解読した処理結果は、外部ファイルを介することなくメモリ上に高速で直接構築され、様々な視点から眺めたり、偏光メガネを用いて立体視したりできる。この処理系については、３－４で解説する。

④携帯端末で動作する AR アプリの試作

異なる（未来の）処理系への可搬性を検証するために、③と同じ処理を Android 系タブレットの上で実行し、GPS 座標と姿勢センサを用いて、建物などの実際の場所（緯度経度で指定）にリアルタイムで写真合成し表示するシステム VC-3M を試作し、釜石において動作テストした。残念ながら、この段階で、各種センサの反応速度と計測精度がまだ実用に供するには不十分であったが、原理的には動作することを確認した。この処理系については、３－５で解説する。現場投入は機器の性能向上を待って 2014 年となった。

⑤災害関連の GIS データの編集に関するプログラム作成

平成 23(2011)年３月 11 日に東日本大震災が発生し、復旧・復興を支援するために国土地理院により大量の高精度 GIS データが、レーザー・スキャナ技術等を用いて作成された。これにより精度の高い地形データが無償で利用できる環境が整った。そこで、データの長期保存を目的とした①～③の研究開発と平行して、これらのデータを有効活用するために、数値地図などの読み込み、変換、分割や接続、座標系に関する処理、及び高台整地等の立体的図形演算処理を行う、景観シミュレータのためのプラグイン（flow.dll）を作成し、WEB 公開した（http://sim.nilim.go.jp/MCS/flow/flow1.asp)。

地形データの上に整地エリア（二次元的領域）と標高を指定し、地形との間に切土盛土の法面を指定勾配で生成して、閉じた図形として整地後の地形を生成するための、高台整地の図形演算のアルゴリズムを明細書に記述し特許を出願し、翌年登録された（高台整地プログラム、ダイナミックリンクライブラリ及び景観検討装置、出願番号 2012-134951, 2012.6.14、特許第 5320576 号,2013.7.26)。

（３）平成 24(2012)年度

①WEB サーバー上の三次元データ保管庫の開発

過去に作成された各種形式による記録データを、添付されたメタファイルに基づいて解析しデータベースに蓄積する WEB アプリケーションを開発した(VC-4D)。この処理系では、前記と同じ COORD 関数が実行されると、ライブラリ関数は SQL コマンドを発行し、三次元座標を格納するテーブルに三次元座標値を追加する。同様に面や立体などを定義する全てのライブラリ関数を SQL テーブルに展開する処理が終了した段階で、データファイルが記述している建物等が、データベース上に幾何学的要素に展開される。

この要素に展開されたデータを入力された記録データとは異なる様々なファイル形式の外部データに再構築・変換して出力するために、データベースから図形や座標値を系統的

に取り出す出力系ライブラリ関数を増補した。これを用いて、データベースに記録された建物等の形状を元のファイルとは異なる任意形式のファイルとして出力する処理をプログラミングし、出力用メタファイル（プログラム）を試作した（三次元データ保管庫)。この利活用範囲を拡張する処理系については、３－６で解説する。

これまでに作成した４種類の処理系において、データファイルの解読方法を記述したメタファイルは同じものが使用できることを比較検証した。即ち、同一のデータファイルとメタファイルを使用しつつ、読み込んだ後の利活用方法（ファイル出力、画面表示、データベース処理・・・）は、ライブラリ関数の実装方法によって変更ができる。

原データにメタファイルを添付して寿命の長い媒体に記録する「保存工程」においては、将来（数百年後）の利活用方法については未知数である。しかしながら、４種類の実装例に共通する基幹部分（コンパイラの仕様）と、利活用目的に応じて異なる動作を実装するライブラリ関数ついて実例を通して解説することにより、将来の利活用工程において、新たに必要となる処理系を実現するためのライブラリ関数のプログラミングに資する十分な具体的資料を提供することになると考える。

②奥尻島の災害復興に関するアーカイブスの作成

北海道南西沖地震(1993 年の被災調査の際に建設省建築研究所等が撮影し国総研に保管されてきた計測写真等を再整理すると共に、北方建築総合研究所（北総研)、北海道大学、奥尻町と連携して被災前の古写真の収集に努め、古写真の撮影位置・時期を特定した。写真から個々の住宅のテクスチャ付き三次元モデルを復原し、これを配列することにより、三次元アーカイブスとし、復興 20 周年となるこの年に WEB 公開した。

http://sim.nilim.go.jp/Okushiri

この段階での三次元データ形式と配信方法は、2001 年度に国総研で開発したまちづくり・コミュニケーション・システムと同様であるが、三次元データが将来計画案ではなく過去の復原である点が異なっている。

2011～12 年度に開発した４種類の処理系の構成と開発方法に関しては、「３.システム開発者の手引き」で詳しく解説する。

１－２．地域居住空間の三次元アーカイブスの利活用

平成 25～26(2013～14)年度の２年間をかけて、上記３年間の研究開発における基礎的な技術の成果を活用して、具体的なサンプル地域に関する三次元アーカイブスを WEB サイト上に作成すると共に、将来における利活用のイメージを具体化するために、例示することのできるアーカイブスを増補すると共に、タブレット端末を用いた処理系を性能向上して、実用的なレベルに高めた上で、現場デモ等に投入することを目指した。

（１）平成 25(2013)年度

①筑波移転に関するアーカイブスの作成

筑波移転に関する閣議了解から 50 周年、また上記の奥尻島における南西沖地震と復興か

ら 20 周年などを契機として、これらの地域に関する過去の記録に関する社会的な関心が高まったことを受け、国総研に保管されている資料の重点的な整理を行った。具体的には、ステレオ計測写真（ネガ）として保管されている資料のスキャン作業、現況図や計画図として保管されている紙図面のスキャン作業と、これらからの三次元データの復原作業を行った。

これらの成果（三次元データによる過去の記録）を、まちづくり・コミュニケーション・システム（三次元データによる未来の予測を目的として、2001 年度に開発）と同じ方法で、過去を記録する WEB サイトにとりまとめ、一般からアクセスできる状態とした。

東日本大震災被災地に関しては、瓦礫の中から古写真等が精力的に回収・保存され、スキャンされていた。また、国立国会図書館においては、アーカイブスのための「ひなぎく」サイトが開設され、情報の整理保存活動が行われていた。三次元データとして集約する方法は、これらの次の段階として有効な処理方法であると考えられたため、国土地理院の所蔵する過去の測量成果等も加えつつ、奥尻島、筑波地区等に関して三次元的な復原を検索用メディアとして、ここから復原の根拠となった各種資料にアクセスできるような形で整理を進めた。平成 26(2014)年 3 月 12 日には、記念シンポジウムを建築研究所展示館において行った（４－６）。

②記録媒体の試作

レーザー加工機、３D プリンタなど、データを寿命の長い素材の上に直接加工する技術や装置が急速に普及し始めていた。そこで、伝統的な棟札や定礎などのように建築物に付帯して設置保存される記録媒体を目的として、より高密・長寿命の素材に記録する方法を試した。データファイルとメタファイルを、レーザー加工機を駆動することができるデータ形式に変換して出力するための Windows 上のアプリケーションを試作した。この分野においては、現在加工方法や記録密度に関して新たな技術が出現しつつあるため、本書では概要に留める。

③発表と特許出願

研究成果（技術）の記録と普及を図るために、各種学会での発表を積極的に行った（章末文献①～⑨）。更に、代表的な技術に関して特許明細書の形で資料化し、２件の PCT 国際特許出願を行った。

[1]「データファイルを長期保存するためのファイル処理方法、ファイル処理装置及び保存媒体（PCT/JP2013/006737, 2013.7.3、特許第 5768280 号, 2015.7.3）」

[2]「情報処理装置、情報処理方法及びプログラム（PCT/JP2013/006737, 2013.11.15, 特許第 5817012 号, 2015.10.9 ）」

PCT 出願とした理由の一つは、国内移行手続きのための参考資料として、特許庁から国際サーチレポートが返されることから、平成 22(2010)年度に行った国内先行技術調査で検索にかからなかった先行技術が見いだされる可能性がある点である。実際に、記録データの長期保存・利活用を目的とするいくつかの先行技術が報告されたが、住宅等の建物の三

次元データの記録保存という技術の使用分野や、コンパイラ処理系を使用する本研究成果の技術の内容において異なることが判明した。

（２）平成 26(2014)年度

①奥尻島における体験教室

携帯端末の各種センサの性能が向上したため、これまでに作成したデータと、携帯端末を用いた現場での閲覧システム（VC-3M）を用いて、かつて町並が存在していた実際の現場での体験教室を実施した（奥尻島岬公園および浜風公園、平成 26(2014)年 10 月 8 日)。このために、レンタルにより調達した最新の携帯端末３機種を用いて動作確認と性能比較を行い、必要な改良・修正を加えて、同じアプリ、メタファイル、データを用いた場合における異機種間の互換性・再現性を改善した上で、現地体験教室を開催した（「むかしめがねアプリ」、２－１、２－２参照)。更に、OS として Android を使用する携帯端末への閲覧システムと表示用データのセットアップを行うための圧縮ファイル（zip）形式を作成し、三次元アーカイブスの WEB サイトに追加する形で公開した。このロードモジュールの作成方法を２－５で解説する。

②図面資料からのデータ作成

筑波移転前の研究機関の跡地（東京およびその周辺）に関する資料を収集・分析すると共に、旧建設省建築研究所が存在していた新宿区百人町３丁目に関して詳細に変遷を追跡した。特に、従来あまり重視されてこなかった建物や施設の写真資料や図面を整理した。その中で新たに発見された、従来の機関史等で言及されていない 1980 年代に除却された建物の記録図面の中から、主要な１６棟の実測図を資料とした、三次元データ作成業務を、一般競争入札で実施した。この業務の特徴は、データ作成に使用するソフトウェア（CAD 等）と、納品するデータ形式を指定せず随意とする代わりに、メタファイルを作成するために十分な内容を有する、データ形式に関する解説資料を成果に添付する、という特記仕様（４－６参照）としたことである。

2014 年 9 月 3 日に、上記の移転前の空間構成の記録に基づく復原をテーマとして、第 2 回研究会を開催した（建築研究所展示館)。なお、この研究会はその後 2019 年 10 月 30 日の第 5 回まで継続し、その成果は４－６に反映されている。

奥尻島、筑波地域の例に加えて、移転跡地に関するアーカイブスを増補すると共に、データ作成から利活用に至る作業手順等に関する解説と資料を本書の中に収録した（４－６)。

この５年間の一連の研究は建設省建築研究所、土木研究所、国総研において平成５(1993)年度以来行われてきた景観シミュレーション・システムに関連する研究の技術的延長上に行われたものであり、データ作成・編集や、テストデータの検査、試作プログラムのテストにおける道具として過去に開発したプログラムを活用した。しかし、目指した方向は、上記システムの拡充・発展ではなく、主に Windows アプリケーションとして 20 年間使用された上記システムに固有の束縛条件からの解放であり、各種三次元データの地域の居住空間の記録としての長期保存に向けた利活用技術の普遍化である。

１－３．成果

（１）論文

①小林英之、稲垣森太「奥尻島集落における古写真の位置比定と編年について－1993 年以前の青苗集落を中心として」日本建築学会北海道支部研究報告集(pp.405-412)、2013.6

② Hideyuki KOBAYASHI: ”3D Archiving Houses and Settlements -Alternative Technologies to support Lasting Diachronic Memory-”, International Symposium on City Planning, 2013.8, Sendai

③小林英之「奥尻島における古写真を用いた 1993 年被災集落の立体的復原について」日本建築学会大会学術講演梗概集(pp.457-8)2013.8

④小林英之「高台整地の景観シミュレーションープラグインの開発－」土木学会全国大会Ⅳ-048(2013.9)

⑤岡田成幸、中島唯貴、大柳佳紀、小林英之ほか「奥尻島災害復興過程にける生活環境の変容に関する研究」北海道地域自然災害資料センター紀要(2014.3)

⑥南慎一、小林英之、稲垣森太、大柳佳紀「奥尻島の記憶の町並再生プロジェクト報告」北海道地域自然災害資料センター紀要(2015.3)

⑦小林英之「新宿百人町の除却された建物群のデータ復原－三次元アーカイブスの永久保存に向けて－」日本建築学会大会学術講演梗概集(歴史意匠 No.9435, 2015.9)

⑧小林英之「建築物などを記録し長期保存されたレガシーデータの利活用方法－メタファイル・コンパイラと利活用ライブラリによる処理系－」日本建築学会大会学術講演梗概集(情報システム技術 No.11032 ,pp.71-72, 2016. 8)

⑨ Hideyuki KOBAYASHI: “Use of 3D Archives of Houses and Settlements for Permanent Memory -2 cases of 3D data attached with metafile for preservation of records with 4 trial cases of application toward future usage- “, 11th International Symposium on Architectural Interchange in Asia (ISAIA, September 2016, Sendai, C-5-4, 5p)

（２）特許

本研究の成果として出願および取得した特許は、以下の通りである。

①特許第 5039978 号「三次元図形演算プログラム、ダイナミックリンクライブラリ及び景観検討装置」平成 23 年 3 月 10 日出願(特願 2011-052466)、平成 24 年 7 月 20 日登録

②特許第 5320576 号「高台整地プログラム、ダイナミックリンクライブラリ及び景観検討装置」平成 24 年 6 月 14 日出願(2012-134951)、平成 25 年 7 月 26 日登録

③出願 PCT/JP2013/004121「データファイルを長期保存するためのファイル処理方法、ファイル処理装置及び保存媒体」平成 25 年 7 月 3 日出願

③－１（日本国内）特許第 5768280 号「データファイルを長期保存するためのファイル処理方法及び保存媒体」平成 25 年 7 月 3 日出願(特願 2013-5029309)、平成 27 年 7 月 3 日登録

③－２（日本国内、分割出願）特許第 6064266 号「データファイルを長期保存するためのファイル処理方法､及びファイル処理装置」平成 27 年６月５日出願(特願 2015-011343)、平成 29 年１月 6 日登録

③－３（インドネシア国内出願）「Tata Cara, Sarana dan Pembawa untuk Melestarikan Arsip Data selama Jangka Panjang」2016 年１月 21 日出願､22 日受理(P00201600426)

④出願 PCT/JP2013/006737「情報処理装置、情報処理方法及びプログラム」平成 25 年 11 月 15 日出願

④－１（日本国内）特許第 5817012 号「情報処理装置、情報処理方法及びプログラム」平成 25 年 11 月 15 日出願(特願 2014-506384)、平成 27 年 10 月 9 日登録

④－２（日本国内、分割出願）特許第 6064269 号「情報処理装置、情報処理方法及びプログラム」平成 25 年 9 月 8 日出願(特願 2015-189158)、平成 29 年 1 月 6 日登録

①は、三次元図形演算を行う機能を提供するダイナミックリンクライブラリに関するものである。1996 年にダウンロードサービスを開始した景観シミュレータ 2.05 で、道路法面生成機能､1999 年の 2.07 において試作したトンネル機能等で使用していたブーリアンの機能を拡張し、表面が穴あきポリゴンから構成された多面体の図形演算を精密な交線の計算とトポロジーの分析により実行するものである。DLL とすることにより、独自にデバッグすることができ、また、他のアプリケーションにおいても、ライセンスを国総研から取得することにより、図形演算機能を使用することができるようになる。2011 年にリリースした景観シミュレータ 2.09 においては、プラグイン DLL で追加した「園路生成機能」の中でこの機能を使用している。

②は、三次元ソリッド図形として表現された地形の上に任意のエリアと標高を指定し、平坦地を生成するものである。指定したエリアの周辺に、指定勾配の法面のテンプレートを作成し、これと地形との間に、①の三次元図形演算を適用することにより、住宅団地や集落を形成するための平坦地を生成することができる。

東日本大震災に際して、国土地理院によりレーザー・スキャナによる精密な測量が行われ、地形データが利用可能となったことから、これを活用して高台移転等の計画に利用可能な機能を提供した。

③は、この報告に解説している内容に深く関係している。住宅や社会資本を記録する三次元データには様々なフォーマットがあり、それらを開いて活用するためのアプリケーションや、その実行環境（OS）の急速な陳腐化が進むことから、貴重な記録を含むレガシーデータが利用困難となりつつある。このような状況に対処して、データファイル本体を解読するために十分な情報を含むメタファイルを添付して、長期保存媒体に記録保存すると共に、そのメタファイルを記述するための文法と辞書（コンパイラの仕様）を別途、永久保存しておく、という方法を実現した。

④は③のような形で保存されたデータを利活用する一つの形態として、記録された建物等が存在していた場所の現況を背景画像として、GPS 機能を備えた携帯端末を用いて、記

録された建物のパースを作成し、リアルタイムで写真合成を行う技術に関するものである。

（３）WEB 公開

①三次元住宅情報の永久保存技術に関する基礎的研究

http://sim.nilim.go.jp/MCS/phi/phi.asp

②地域居住空間の三次元アーカイブス

http://sim.nilim.go.jp/地域名

の web アドレスに、地域別に構築している。また、これらのコンテンツのサーバー間移植、OS の更新などについて、４－９で解説した。

（４）

雑誌紹介記事など

①小林英之「地域の三次元アーカイブスの利活用－災害や大規模開発を超えて持続する地域の生活空間を可視化する－」国総研レポート 2014, p.116

②小林英之「デジタル考古学事始め　第１部：近過去を振り返る」建築の研究，No.209, 2012.2, pp.1-6m，建築研究振興協会

③小林英之「デジタル考古学事始め　第２部：未来の考古学者へ」建築の研究，No.210, 2012.4, pp.25-30，建築研究振興協会

④小林英之「デジタル考古学事始め　第３部：道具編」建築の研究, No.211, 2012.7, pp.30-33, 建築研究振興協会

２．三次元アーカイブスの構築と利活用の手引き

はじめに

本章では、様々な資料を整理・加工して過去に構築された三次元アーカイブスが、どのような形で利活用可能かをまず示す(２－１、２－２)。

次に、将来の様々な利活用を可能とする記録保存データの整理・加工方法について、この研究の中で実行した手順を解説する(２－３)。

更に、本処理系の最大の特徴である、任意形式の保存データに添付して、解読処理を処理系に指示するための形式定義ファイル（メタファイル）の作成方法について解説し（２－４)、これを記録データに添付して保存する方法について解説する(２－５)。

最後に、データベース上でデータを保管すると共に、任意形式のデータとして取り出す「三次元データ保管庫」の操作・運用方法について解説する（２－６、２－７)。

２－１．タブレットを用いた現場での閲覧機能(VC-3M)の利用

本節では、利活用形態の一つである、スマートフォンまたはタブレットを用いて現場での記録の閲覧を行うエンドユーザーのための操作方法の解説を行う。

このプログラムは、端末の背面カメラが取得した画像を表示画面にそのまま表示すると共に、それに重ねて記録保存された建造物の三次元データを CG で合成表示する。その際に、端末に装備された GPS センサ、加速度センサ、磁気センサから計算した端末の位置とカメラアングルを用いて、記録データの透視図を作成することにより、その建物が存在していた位置あるいは将来計画されている位置に存在するかのように合成表示することが特徴である。写真 2-1,2 は、平成 26(2014)年 10 月８日に、奥尻島の岬公園において、平成 5(1993)年以前に存在していた集落を表示したタブレットを見ながら、小学生たちが過去の集落の道を歩いて追体験している風景である。「五区」と呼ばれていたこの集落は、平成 5(1993)年７月 12 日に発生した北海道南西沖地震とそれに伴う津波で消滅し、生き残った住民は「防災集団移転」により高台に新たに開発された団地に全て移転した。その跡地は公園として整備され、現在ここには住宅はない。昭和６(1931)年に建立された「徳洋記念碑」が変わらず存在してランドマークとなっている。

写真 2-1 奥尻島における体験教室の風景（2014.10.8）

児童が閲覧している町並のアーカイブスは、被災前の古写真から復原した個々の住宅を

道に沿って配置して再現したものである。

タブレット端末の操作は、基本的には眼前にかざしながら持って歩くだけであり、体験教室で試した結果、小学校低学年でも十分に、起動から終了までの一連の操作を理解し実行できることを確認した。子供向けに、「むかしめがね」という名称で説明した。

写真 2-2 タブレット端末の表示内容

（1）起動と表示する場所の選択

図 2-1-1 マイアプリ画面で、「むかしめがね」のアイコンをタップして起動する

使用したタブレットは、OS として Android を搭載しており、起動した状態で、主画面にアプリのアイコンが表示されている。画面上で、アイコンを指で軽く叩いて（以後、「タップ」して)、「むかしめがね」アプリを起動すると、次のような初期画面が表示される。

図 2-1-2　「むかしめがね」初期画面

「見たい場所」のボタンをタップすると、準備されているいくつかの場所から選択を行う画面に進む。「おわり」を押すと、終了して、OS の画面に戻る。

図 2-1-3　見たい場所の選択画面

見たい場所の一覧から、場所を選び、文字をタップすると、合成表示の画面に進む。

（２）GPS 衛星の電波取得待ち

合成表示画面では、背面カメラから取得した画像が表示され、その上に記録保存された建物などが合成表示される。

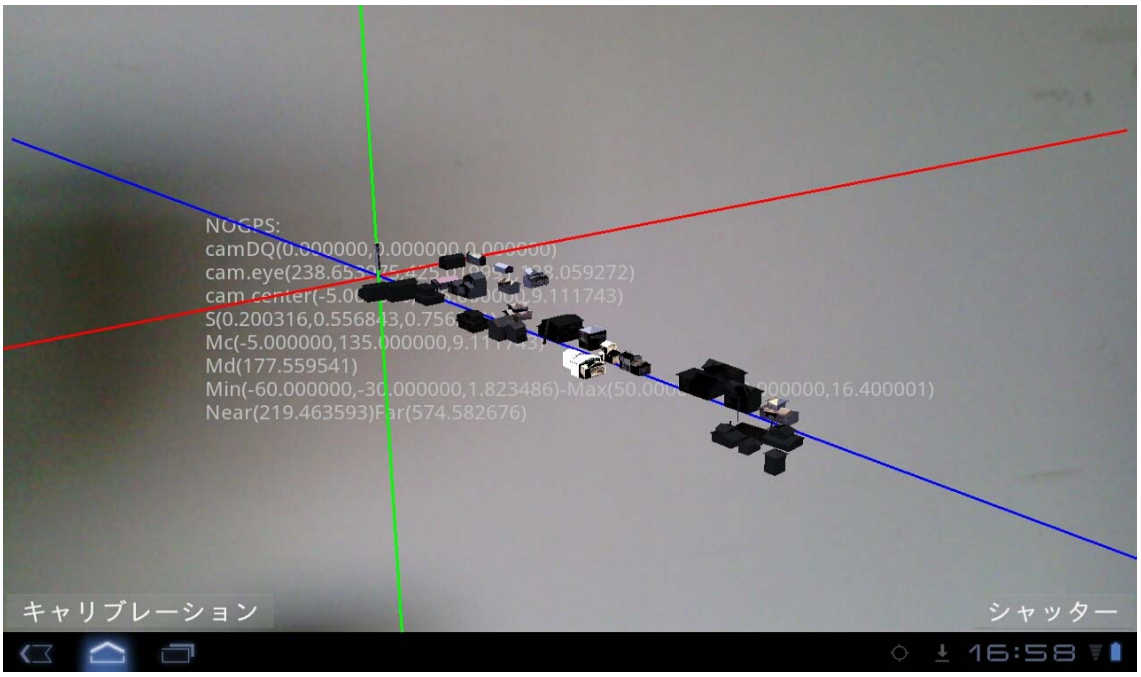


図 2-1-4 GPS 衛星を探索中の表示

ここで GPS 衛星からの電波から現在位置が取得できるようになるまでの数秒～数十秒の間は、端末の位置にかかわりなく、建物を表示する。その時、端末の向きだけは表示に反映する。例えば、端末を地面に向けると、建物の屋根を上から見下ろした表示となる。また端末を垂直に北に向けると、建物の南面を表示する。横に回転させると、建物が回転し、すべての壁面を見ることができる。

GPS 衛星からの電波が受信できない場所で動作させた場合には、この状態が続く。

（３）現場での閲覧

GPS 衛星から位置が取得できると、その位置からの表示に変わる。この時、町並の中に立っていれば、道の両側に建物の正面が並んで表示される。道に沿って進むと、手前の建物が後ろに隠れ、遠くの建物が近づいてくる。

建物の内部に入ってしまうと、建物の内部の壁だけが表示され、背景の風景は全て隠されてしまう。

町並から離れた位置から眺めた場合には、端末の向きが悪いと建物が全て範囲外となり、画面には背景だけが表示されることがある。そのような場合には、画面の中央に Left（ひだり）、 Back（うしろ）, Right（みぎ）といった文字が表示される。

Left（ひだり）が表示された場合には、端末を左に向けると、町並が表示の範囲に入ってくる。Back（うしろ）の場合には、回れ右をすると町並が見つかる。

Far（とおい）と表示されるのは、遠すぎて町並全体が画面中央の小さな点にしかならな

いような場合である。例えば東京からむかしめがねで、奥尻の町並を見ようとした場合には、北を向けても Far(とおい)と表示されるだけである。

（４）足による補正（高学年向き）

GPS センサの精度が低い場合には、町並が表示される位置がずれる。そのような場合には、左下の「キャリブレーションボタン」を押すと、町並の表示位置が画面に固定される。そこで、このまま端末を持って歩いて移動し、正しい位置を見つけ、そこでボタンを離す。すると、その間のずれの距離が記憶され、以後の表示はこれを用いて（補正して）表示される。GPS センサが衛星電波を受信し続けている間は、比較的滑らかに位置情報が得られる。しかし、一度 GPS センサを停止し、再起動すると、前回とは大きく異なる位置情報が得られる場合がある。従って、閲覧を開始してから補正を行うと、閲覧継続中は比較的正確な位置に表示される。

図 2-1-5　キャリブレーションボタン

この補正は、携帯端末の移動と回転の両方に関して同時に行うため、押し始めと離すタイミングで端末が同じ向きである必要はない。また、一度補正を行った状態で、さらにズレがある場合には、再度補正を行うと、補正は累積的に行うため、補正の操作を繰り返し正しい位置に近づけていくことができる。

（５）シャッターによる静止画の記録

うまく表示されている状態で、好きな場所が見つかった場合、そこで右下の「シャッター」のボタンを押すと、表示されている画像をデジタルカメラと同じように JPEG ファイルに保存すると共に、その時の端末の位置と姿勢を撮影記録として保存する。

図 2-1-6 シャッターボタン

画像の保存が終了すると、「保存しました」というメッセージが一瞬表示され自動的に消える。

（６）屋内での記録データの再生

現場での表示は、下の「戻る」ボタンにより終了し、最初の画面(1)に戻る。そこで、「おわり」のボタンをタップすると、アプリが終了する。この時に、シャッターで記録したデータがファイルに保存される。「おわり」を押さないまま、バッテリー切れ等により終了したような場合には、記録は残されない。

図 2-1-7　「歩いた場所」をタップして記録データを再生する

現場表示画面から戻った初期画面または、再びアプリを起動した時の初期画面で、「歩いた場所」のボタンをタップすると、シャッターで記録した場所の一覧が、小さな画像で表示される。

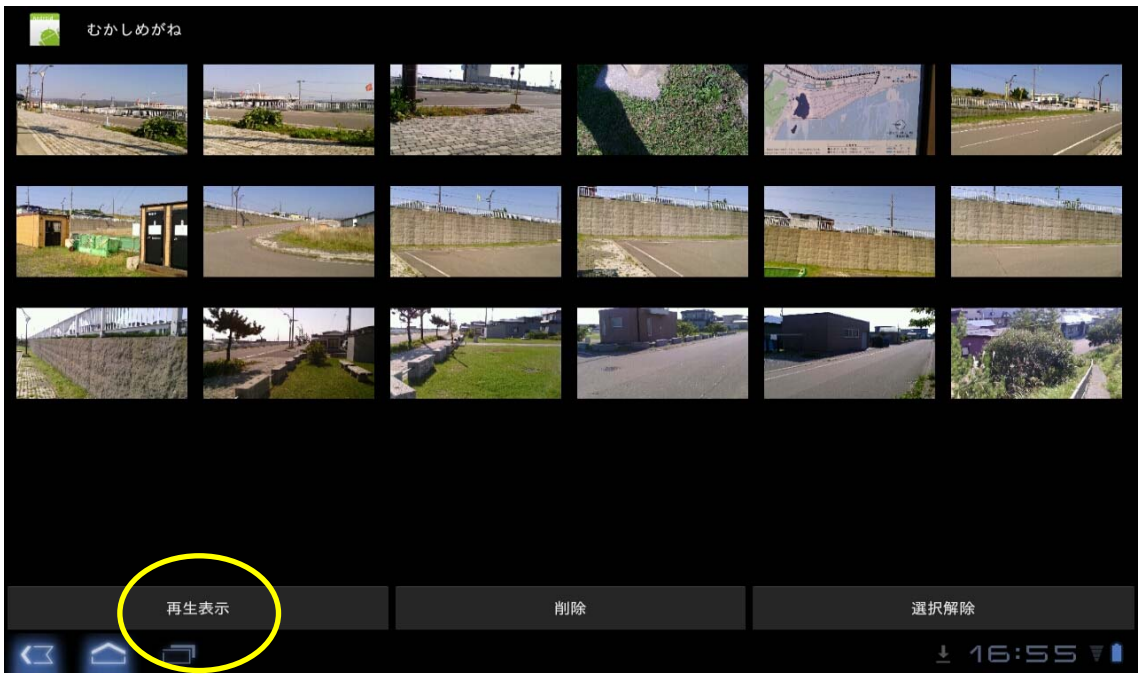


図 2-1-8　記録した場所の一覧（サムネイル画像）

この画面に並んで表示される小さなサムネイル（見出し）画像は、保存した背景画像だけが表示され、建物などのデータの合成表示はない。シャッターを押した時点で、建物等

が表示範囲の外であった場合には、サムネイル画像の両側に赤い線が表示される。

この一覧から、ある画像を選択すると、その画像の両側に緑色の表示が出て、選択された状態であることを示す。

この画像が一つ選択された状態で、「再生表示」ボタンをタップすると再生表示撮影した時点での合成画像が表示される。この再生表示画面は、視点位置が保存された画像に固定されているため、携帯端末を動かしても表示は変化しない。

一つのサムネイル画像を選択した状態で、「削除」ボタンを押すと、その画像と、カメラ位置・姿勢等の記録が削除される。（画像ファイルそのものは残している）

（７）その他

タブレットを使用する上で有用ないくつかの操作方法（ノウハウ）について解説する。

・画面キャプチャ（静止画）

広報用資料や解説資料を作成するために、画面のキャプチャないしスクリーンショットを作成したい場合がある。PC 上の Windows においては、PrtScr キーを押すことにより、クリップボードと呼ばれる一種のバッファに画面データがコピーされ、次に Ctrl－V のキー操作により、別のアプリケーション等（例えばワードプロセッサ）の上にこの画像を貼りこむことができる。類似の操作を Android タブレットで行う場合には、機種によって異なるが、例えば画面左下の「戻る」と「ホーム」のソフトボタン（画像で表示されているボタン）を同時に素早くタップすると、保存するファイル名を入力するダイアログが開き、画像ファイルとして画面を保存することができる。あるいは、「電源」と「音量(-)」のハードウェア・キーを同時に長押しすることにより、同様の操作ができる。作成した画像ファイルをワープロ原稿等に使用することができる。OS のバージョンにより操作方法が異なるため、機種毎に確認する必要がある。

・画面キャプチャ（動画）

HDMI 端子を有する携帯端末の場合には、接続ケーブルでモニタ等に接続することにより、表示されている映像を取り出して複製することができる。但し、コピー・プロテクトがかかっている機種の場合には、これを動画としてファイルに保存することはできない。

専用アプリをセットアップして、動画を保存する方法がある。操作マニュアルを動画として作成するような目的に使用可能である。

２－２．指導員のための手引き

指導員は、必要な携帯端末にプログラムとデータをセットアップし、起動できるような状態を準備する。次に、使用する携帯端末が、必要とする機能と十分な精度を有しているかをテストする。２－１で解説したエンドユーザーによる閲覧が終了した後、必要であれば撮影データ等の回収・保存を行う。最後にアンインストールを行い、携帯端末をセットアップ前の状態に戻す。

エンドユーザーが本節に解説する作業を自ら実行してもよい。

使用する携帯端末は、操作時点でインターネットに常時接続している必要はない。

（１）セットアップの準備

「むかしめがね」を携帯端末にセットアップするためには、必要なファイルを携帯端末の内蔵 SD カードにコピーした上で、VC-3M.apk というセットアップ・プログラムを実行する必要がある。

セットアップ・プログラムを実行するためには、内蔵 SD カードに存在するファイルを表示し実行するための、「ファイル・マネージャ」ないしこれに類するプログラム（Windows の「エクスプローラ」に相当するもの）が必要である。機種によってはプレ・インストールされているが、そうでなければ、フリーソフトをダウンロードしてもよい。

（２）必要なファイルのコピー

セットアップを行うためには、 WiFi(無線 LAN によるインターネット接続)経由でインターネットに接続してダウンロードする方法、USB 経由でパソコンに接続して必要なファイルをコピーする方法、あるいは USB メモリを接続して、そこからファイルをコピーする方法などがある。

コピーするファイルの内、大きな容量を占めているのが町並みを構成する建物等のデータである。これに、必要なプログラムと、「見たい場所」の一覧を表示するためのリストなどがある。これらは、「VirtualConverter」という、SD カードのルート直下のフォルダに格納されている。

WEB 経由でダウンロードしたセットアップ一式は、WiFi 経由でインターネット接続した携帯端末でダウンロードするか、またはこれをダウンロードした PC から USB 経由で携帯端末にコピーする。

セットアップ一式は、ZIP 形式で一つのファイルに束ねられているため、これを解凍する必要がある。この解凍は携帯端末上で行っても、また PC 上で予め解凍してからコピーしても良い。

（３）セットアップ

「むかしめがね」をセットアップするためには、「設定」で、商用ソフト以外のプログラムのセットアップができるようにしておく。セットアップを実行するためには、上記の「ファイル・エクスプローラ」で、VirtualConverter ディレクトリを開き、その中にある、VC-3M.apk というファイルをタップする。古いバージョンの VC-3M が既にセットアップされている場合には、上書きでセットアップすることができる。

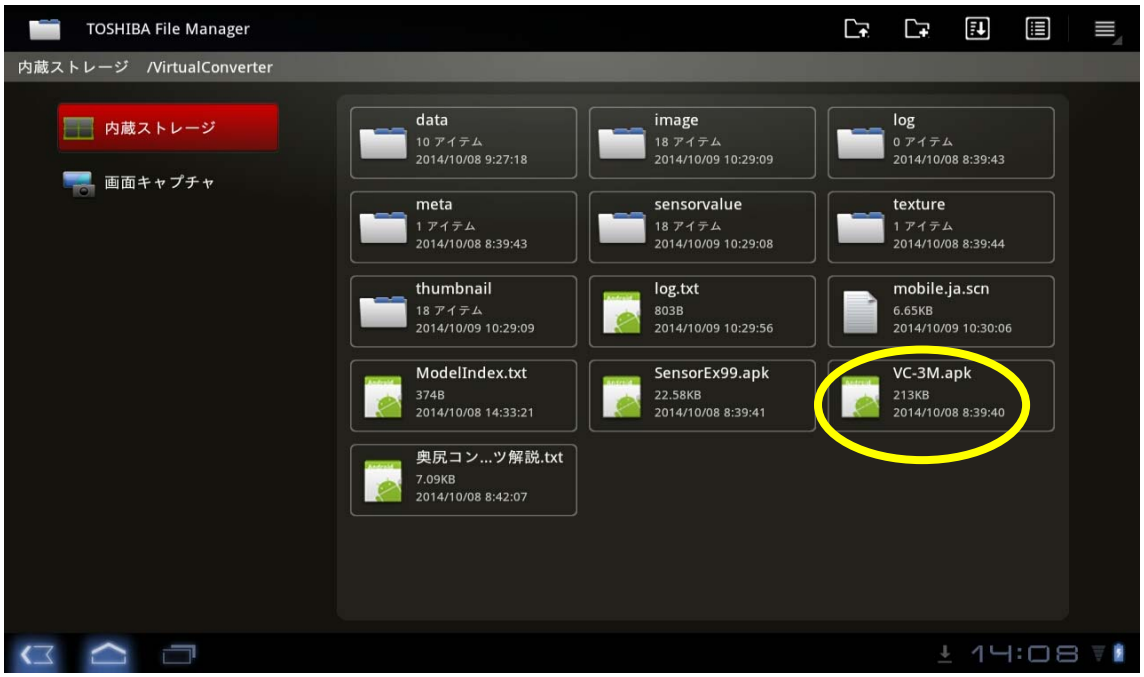


図 2-2-1 VirtualConverter ディレクトリ内部のファイル構成

（４）GPS 機能の設定

「むかしめがね」は、GPS を使用するので、「設定」画面で、GPS を使用できるように設定しておく。

この設定が行われていない場合には、アプリを起動した時点で、警告表示を行い、GPS が利用できない状態のままでアプリを実行する。その場合、GPS 電波を検索中、あるいは GPS を受信できないコンクリート造建物の中や鉄板屋根の下で使用した場合と同様の、端末の向きだけを用いて町並の全体を様々な角度から眺めるような表示動作を行う。

なお、GPS 機能は電力を消費するため、現地での閲覧等が終了した後には、再び「設定」画面を開いて、GPS のスイッチを切っておく方が、バッテリーが長持ちする。

（５）端末の精度の確認

GPS 衛星からの電波が利用できない状態において、加速度センサと、磁気センサの精度をテストすることができる。

加速度センサ（重力センサ）は、端末が静止した状態では、下を指す。しかし、手で持って歩き回る状態では、手振れにより加速度の向きは揺れ動いている。「むかしめがね」アプリでは、手振れの影響を緩和するために１秒程度以前までの過去の計測値の平均を求めて使用している。

磁気センサ（電子コンパス）の計測値は、携帯端末自身の帯磁によって影響を受ける。この影響は、携帯端末に固定した座標系では XYZ３軸の計測値に固定値を加えた値となる。従って、携帯端末を３軸に回転させることにより、その値を計測し、補正することができる。このような補正機能を備えた携帯端末の場合には、ユーザーの利用に先立って、端末自体を大きく振り回して回転させることにより、改善することができる。木製テーブルの

上などで、水平に 1 回転、横に 1 回転、縦に 1 回転する。

地磁気をベクトルとして見た時、水平ではなく地面と一定の角度を有しており、その水平成分が磁北を指している。このため、「むかしめがね」VC-3M では、加速度センサで検出した「下」の向きと直交する成分を抽出して、水平な北の向きとしている。

磁気センサが検出する磁場は、地磁気の他に建物や工作物の帯磁した鉄製品等の影響も受ける。また直流で送電している電車（地下鉄も含む）の架線などから生じる磁場の影響も受ける。例えば、無料でダウンロードできる「GPS Status」のようなアプリを起動し、現場付近で同じ場所で磁北の変化を観察したり建物内外で持ち歩いて磁北と建物の関係を見たりすることにより、影響を把握することができる。

携帯端末のセンサが計測する、携帯端末の座標系(UVW)に即した磁場の向き M と、重力の向き G を用いて、重力の向きに直角な磁北の向き（Y）と上（Z）を求める。

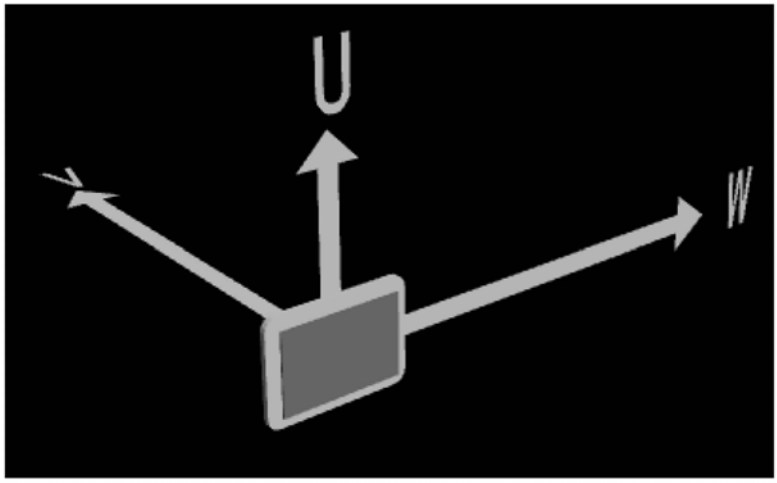


図 2-2-2 携帯端末の向きの取得

GPS 衛星からの電波からの位置（緯度、経度、標高）の計測ができるようになるまでの所要時間は、機種によって大きく異なる。一般に、過去の計測結果を記憶しておき、これを利用して GPS 衛星の探索を行うため、体験教室等の開始直前にテストしておくと、児童がアプリを起動してから、正常動作するまでの時間が短い。新しい携帯端末であってまだ GPS 機能を使用したことが無い場合や、長い間 GPS 機能を使用しなかった場合には、GPS 機能が立ち上がるのに長い時間がかかる。表 2-2-1 に、計測例を示す。

GPS 機能を使用し続けている間は、ほぼ同じ衛星（最低４）からの電波を使用して位置を算出するため、誤差は一定である。従って、上記の「足による補正」の機能は有効である。しかし、一度アプリを終了して GPS を停止し、再び起動すると、前回とは大きく異なる位置が計測される場合がある。

表 2-2-1 新宿百人町における GPS 電波取得に要した時間の計測結果（2015.9.19）

メーカー	製品名	OS	No	GPS 待(秒)	機材番号	緯度	（分）※	経度	（分）
SONY	XPERIA Z2	4.4	1	37	ST2-12	36	5.152	140	5.808
	(SGP512JP/B)		2	30	ST2-13	36	5.146	140	5.806
			3	89	ST2-14	36	5.147	140	5.802
			4	4	ST2-15	36	5.146	140	5.802
			5	29	ST2-16	36	5.144	140	5.809
SONY	XPERIA Z	4.1	1	62	ST3-19	35	42.354	139	41.755
	(SGP312JP/B)		2	31	ST3-20	35	42.353	139	41.754
			3	61	ST3-21	35	42.355	139	41.76
			4	38	ST3-22	35	42.357	139	41.756
			5	23	ST3-23	35	42.351	139	41.756
ASUS	NEXUS7	4.4	1	62	AN2-27	36	5.142	140	5.801
	(2013)		2	16	AN2-28	36	5.146	140	5.807
			3	64	AN2-29	36	5.149	140	5.809
			4	23	AN2-30	36	5.147	140	5.806
			5	100	AN2-31	36	5.145	140	5.804

※緯度の分以下を示し、１分以下は、秒を使用せず分の小数として示す。

（６）データの原点の確認

後述のように、データ作成段階で、建物等を記述する座標系の原点の緯度・経度・標高をいくつかの方法で定義することができる。この数値が正しい値でなければ、背景との合成が正しい位置に行われない。奥尻島の例では、2013 年 6 月に、２台の携帯端末を用いて GPS 計測した、ランドマークである「徳洋記念碑」の緯度経度を用いてデータを作成し、2014 年 3 月に建物単体のデータを用いて表示位置の確認を行った（写真 3-1)。この結果に基づいて、より位置精度の高い機種を精選した上で、同年 10 月に体験教室を実施した。この時、セッション直前のテストで東西に約５ｍ程度の位置ずれが生じたため、同じ位置で緯度経度を計測し直し、正しい位置に表示できるように補正した。GPS により計測される緯度経度は、大気の状態などにより影響され、また稀に地震・地滑り等による大地の移動の影響も受ける。このため、古い計測値を用いると誤差が生じる。しかし、体験教室等の実施中に大きく変化することはないため、直前に測り直してデータを補正することにより、前述の「足によるキャリブレーション」により個々の機材で補正する手間を省き、円滑にセッションを実施することができる。

この直前の補正は、ModelIndex.txt に登録してある、個々のタブレットにセットアップされていたテキスト・エディタを用いて、モデルの座標原点の経度の値を、約５ｍに相当する 0.0005 度だけ増やす方法で補正した（東経 139.4500→139.4505)。

（7）記録データの回収・保存

セッション中にシャッターボタンが操作された場合には、携帯端末中に画像ファイルと、その時点での携帯端末の位置・姿勢・時刻を記録したデータが作成されている。これらを回収するためには、PC と USB 接続して、あるいは着脱可能な SD カード等を介して、必要なファイルを取り出す。

簡単には、終了時点での VirtualConverter ディレクトリ以下を携帯端末のファイル・マネージャの機能を用いて ZIP 圧縮し、このファイルを回収する。

２－３．データ作成方法

データ作成は、建物や町並を記録する任意形式のデータを作成し、これを解読するためのメタファイルを添付し、これらに目録を付けて、パッケージ化するまでの工程である。この内、メタファイルを作成する工程は一種のプログラミングであるため、２－４として別項を設けて解説した。

なお、本節の解説は、現時点で最も利活用の有用性が高い、現場での表示（VC-3M）のためのデータ作成を中心に記述しているが、数百年後の利活用を目的とした長期保存を目的としたデータ作成とも、多くの部分は共通している。保存すべきデータは他にも様々な方法によって作成済のものを扱う場面も想定される。その場合、メタファイルを作成する工程が最も本質的な作業となる。

（1）建物の三次元データの作成

建物の三次元データを作成するためには、以下のような方法がある。

① CAD 入力

1990 年代から、CAD(Computer Aided Design)ソフトウェアを用いて三次元データを作成する方法が実用化し普及した。その後、機械分野では数値制御式(NC)加工機を用いた CAM(Computer Aided Manufacturing)と連携することにより急速に普及した。建築分野でも三次元 CAD の導入自体は早かったが、急速に普及したとは言えず、二次元の CAD データを紙に出力した図面が引き続き広く使われている。近年では、BIM として、三次元データに様々な属性データを持たせて建物を表現する手法が普及し、標準的な IFC 形式が保存データのフォーマットとして勧奨されている。

図 2-3-1 は、福岡県住宅供給公社が建替えを行った峰花台団地の設計図から、CAD ソフト Microstation により入力したデータである（平成 8(1996)年）。当時は、CAD ソフト毎の独自フォーマットで保存されたデータを、コンバータにより、建設省版・景観シミュレータの独自フォーマットである LSS-G 形式に変換して、現場での説明等に利用した。

図 2-3-2 は、1980 年頃に、解体処理に先だって実測された研究施設の記録図面から、Autocad により入力したデータである(平成 23(2015)年)。データは後述の IFC 形式で保存し、この形式を解読するためのメタファイル IFC.cmm を添付したものをタブレット端末に搭載し、VC-3M アプリケーションを用いて、リアルタイム写真合成を行い、現在は高層集

合住宅や防災公園となっている現場での閲覧に使用している。

この 20 年ほどの間、ソフトウェアの性能自体は本質的に変わりないが、使用している PC の性能（メモリ容量、計算速度、３D 表示速度）は飛躍的に向上しており、大規模なデータを表示できるようになった。

図 2-3-1 峰花台団地の景観シミュレーションと、竣工後の比較(1996)

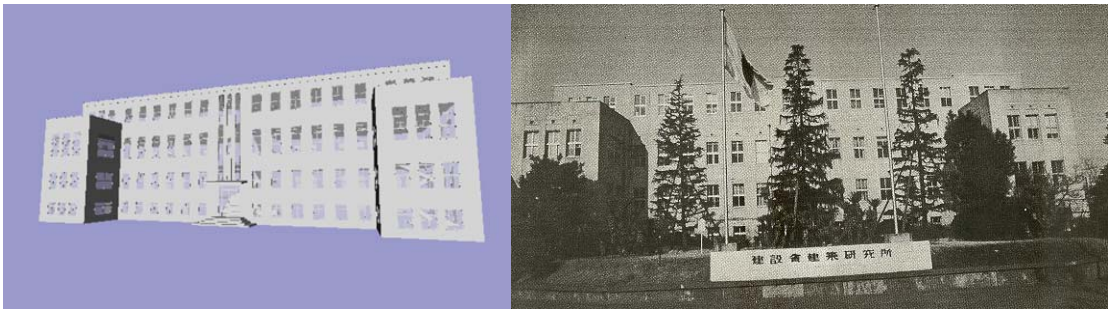


図 2-3-2 実測図からモデリングした旧建設省建築研究所の本館と、古写真(1976 年頃)

② 写真からのモデリング

立体的な形状を計測するために、ステレオ写真を使用する方法は、空中写真から等高線を抽出する方法を、地上写真にも適用することが一部に行われていた。1990 年代にその画像解析技術の自動化が進んだ。

建築物に関しては、直角、水平、垂直が仮定できる要素が多く含まれることから、１枚

の写真を基に立体的な形状を復原することができる。具体的には、写真上から二つの消点を抽出することができれば、建物の形状を求めることができる。このため、殆どの場合ステレオではない記録写真からでも、建物を復原することができる。このような作業を行うためのソフトウェアはいくつか存在している。

更に、一つの区域に関して多くの写真が残されている場合、写真そのものを変形して接合することは容易でないが、立体的に復原した断片を正しい位置に配置することにより、立体的な町並の復原が可能である。

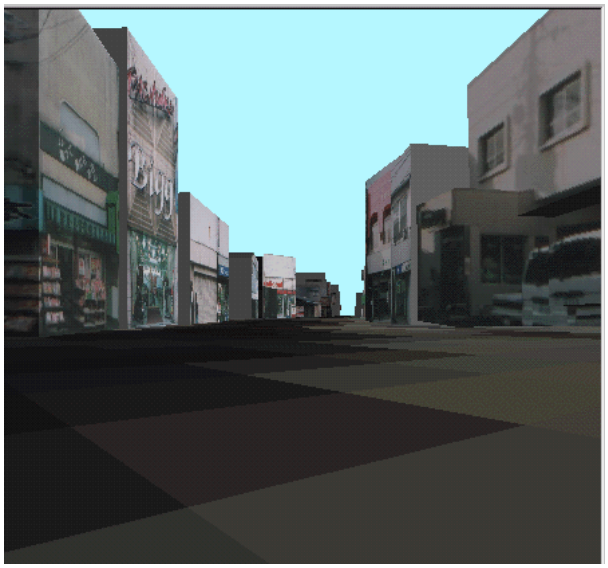


図 2-3-3 デジタルカメラ画像からモデリングした建物を配列した町並（幕張 1996 年）

幕張駅周辺においてデジタルカメラで撮影した沿道の現況建物から復原した建物を配列して作成した町並のデータ(平成 8(1996)年、旧建設省建築研究所)を図 2-3-3 に示す。図 2-3-4 は、奥尻島において昭和 58(1983)年の日本海中部地震による津波の直後に撮影された調査写真（サービス判印画紙）から国総研が平成 14(2012)年に復原した当時の住宅である。

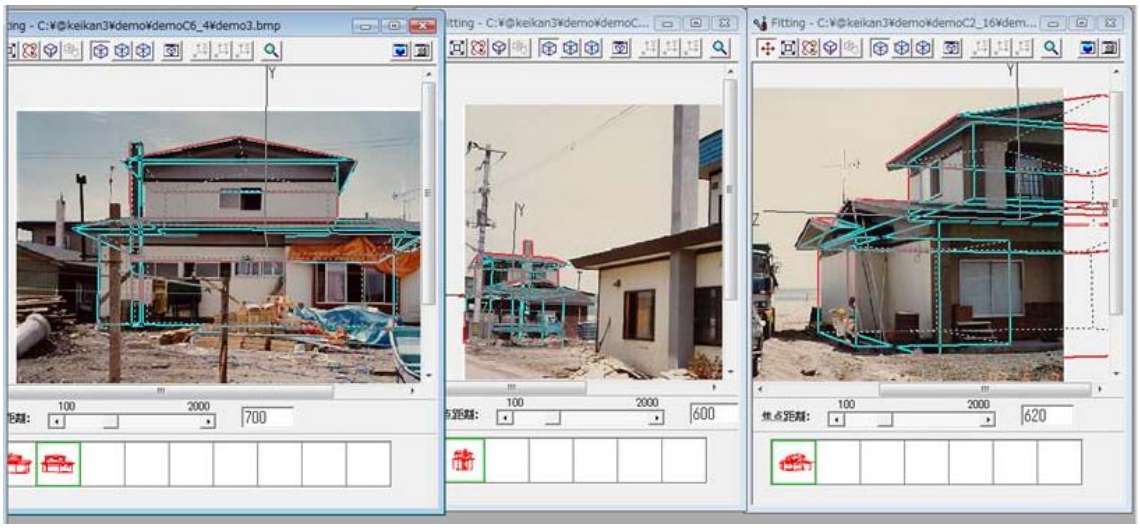


図 2-3-4 1980 年代の古写真から復原した奥尻島青苗の岬地区の住宅

③ レーザー計測

レーザー計測技術により、立体的な地形や建物を点群データとして計測することができる。2000 年頃から、航空機から地形を計測する技術が普及し、DEM データとして利用されるようになった。更に 2010 年代に入ってから、車載型のレーザー計測装置（MMS）が開発され、沿道の町並の形状を能率的に計測できるようになった。図 2-3-5 はつくば市内の建築研究所の実験施設を国総研が計測した点群データである(2013 年計測)。点群データは、XYZ 座標値を羅列しただけの単純なファイル形式による大規模なファイルである。これを建物単位、壁単位、等に分節化し、更には使いやすく小さいサーフェスモデルに変換する処理については、国総研の別課題において研究された(2016 年)。保存時点で適切な解析処理がすでに行われていれば、CAD 入力されたデータ等と同じ方法で扱うことができる。

本研究においては、点群データそのものを保存する場合に際して、位置座標のオフセット等を反映させた上で、レーザー計測装置の回転スキャン動作の中で取得された、連続する一群の計測点を折れ線として集約するところまでを行った。

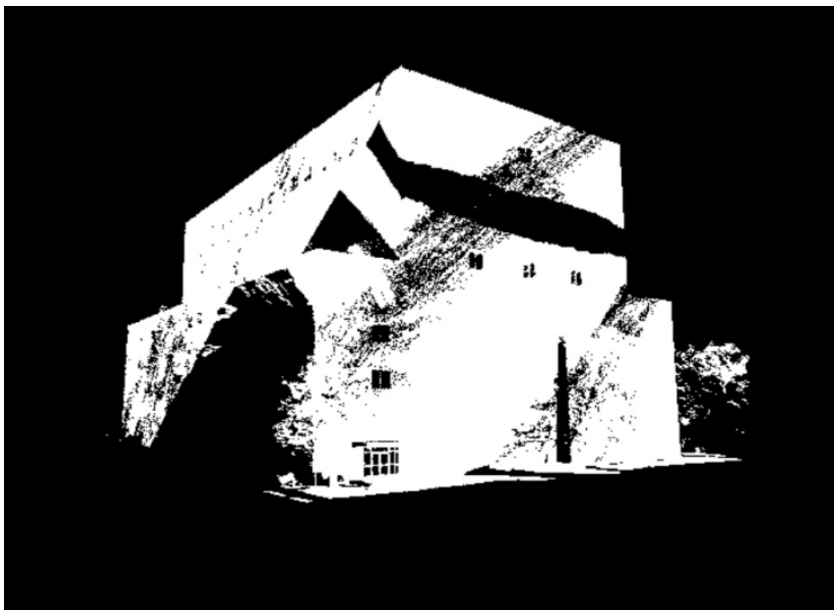


図 2-3-5 MMS で取得した建築研究所の実験棟建物の点群データ(実大構造物実験棟,2013)

（２）地形データの作成

① DEM データの利用

ある区域をメッシュに分割し、それぞれのメッシュの標高の平均値を記録したメッシュデータは、DEM(Digital Elevation Model)として、1990 年代後半から普及した。国土地理院ではこれを受け、数値地図標高として、(一財)日本地図センターからの有償販売を開始し、2010 年代には、「基盤地図情報」として無償で利用できるサービスを開始している。

様々な GIS ソフトで DEM データを採り込んで利用することができる。旧建設省建築研究所も、景観シミュレータ Ver.2.03(1997)*11 からコンバータを提供している。当時は統一された規格がまだなく、個別に作成された DEM データには様々な形式のヴァリエーション

が存在しており、試行錯誤しながら正しい変換結果を得るような作業を行う必要があった。

DEM に対応したオルソ画像が利用できる場合には、これをテクスチャとして地形に適用することにより、樹木・裸地などの識別が可能なデータを作成することができる。例えば景観シミュレータにおいては、地形メッシュが長方形領域である場合には、テクスチャ編集画面で横倍率、縦倍率をそれぞれ領域の東西長さ(m)、南北長さ(m)に指定すればよい。テクスチャ編集地形データが十分に精密である場合には、テクスチャではなく、頂点カラーとして定義する方法も可能である。国土交通省版・景観シミュレーション・システムに付属した「貿易コンバータ」で、DEM を LSS-G 形式に変換する設定画面において、頂点カラーを使用するチェックを入れた上で、使用する空中写真等の画像ファイル名を指定することで、頂点カラーのついた地形等のデータを作成することができる。多面体として表現された地形を構成する頂点位置を地表面に投影した XY 座標に対応する位置の色彩を、指定された画像ファイルのメッシュから補完計算を行った上で抽出する処理を行っている。

② 紙地図の等高線からの生成

古い都市計画図等には、造成後の現在とは異なる従前の地形が等高線と単点で表現されている。三次元モデルを作成するために、大判のスキャナを用いて紙地図から画像ファイル（本件では tiff 形式）を作成し、これを画面表示した上で、マウス操作で等高線を抽出するような作業が必要となる。奥尻島の平成 5 (1993)年の基本図（1:2500）から作成した地形データとその作業過程を、図 2-3-6 に示した。

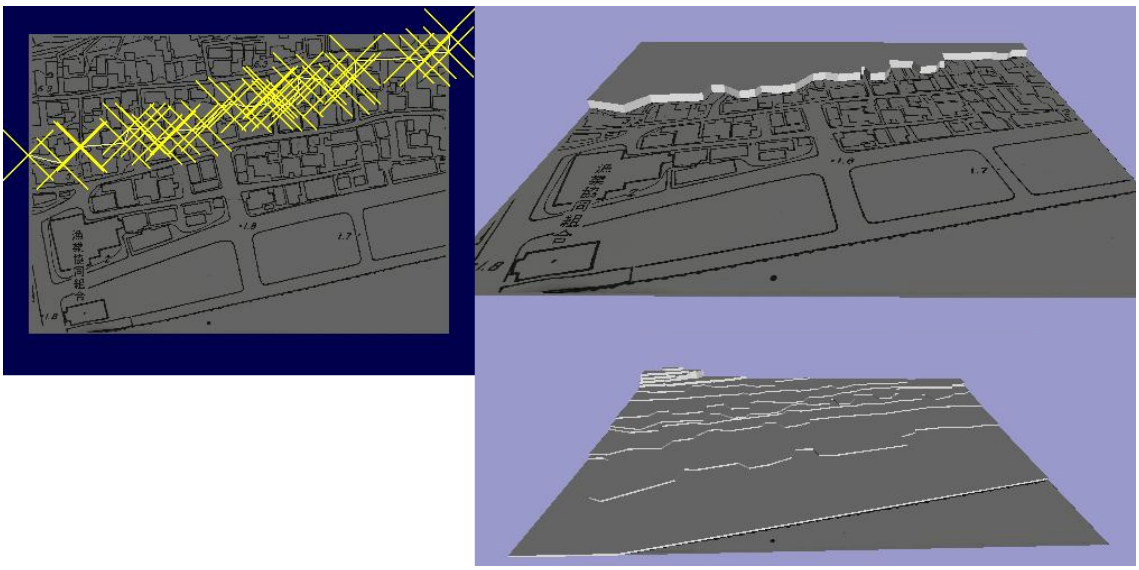


図 2-3-6 古い基本図(1:2500,1993)から等高線を抽出し、地形を再現する

なお、マウスとグラフィックディスプレイが普及する前の 1990 年頃までは、大判のデジタイザにテープなどで地図を固定し、磁気センサとボタンを有するポインティングデバイスを図上の点に合わせてボタンを押し、座標値を計測・入力する方法が広く行われていた。

③ ステレオ空中写真（ライブラリ）からの復原

ステレオ空中写真がアーカイブされていれば、これを解析して三次元データを作成する

ことができる。この場合、地形だけではなく、建物も併せて取得できるため便利である。

2001 年度に実施した「まちづくりのためのコミュニケーション・システムの開発」においては、参加した地方公共団体が 1:2500 基本図の作成のために過去に撮影した空中写真（多くは測量会社が保管）、あるいは国土地理院が 1:5000 国土基本図の作成のために過去に撮影した空中写真から、地形と建物をモデリングし、これを下図として計画案のデータを作成した。

図 2-3-7 ステレオ空中写真から復原した地形（広島,2001）

④ 衛星画像からの復原

国総研が 2004~6 年にインドネシア人間居住研究所の協力を得て、同国内の計画的な住宅地の排出量実態調査と、今後排出量を増加させない計画の検討を行った際には、日本の衛星「だいち」が撮影したステレオ衛星画像を使用した。同国においては、軍事的理由から空中写真の国外への持ち出しは禁止されている。従って、90 年代までは地形データを作成するためには、現地で空中写真の解析を行うか、既存地形図に描かれた等高線から旧来の手法で立体的な地形を作成する必要があった。衛星画像にはこのような制約がないため、地球環境研究に広く利用されている。「だいち」のリニアセンサーは、衛星から地表を見下ろす際に、真下だけではなく、前方と後方の斜め画像も取得する。これにより、あるエリアに関して、衛星が通過する前に撮影した画像と通過した後に撮影した画像をステレオペアとして解析して、既存の市街地と周辺地形の三次元データを作成することができる。

この方法をバンドン市とチレボン市に適用した[29]。このうちバンドン市周辺のデータの例を図 2-3-8～9 に示す。図 2-3-8 は、同じエリアに関して、ステレオ衛星画像から解析した地形と、地形図から抽出した等高線に基づき従来方法で作成した地形の細密度を比較したデータである。図 2-3-9 は、地形図から作成した概略の広域地形を遠景とし、計画団地の

周辺の地形についてはステレオ衛星画像から作成した詳細地形に衛星画像をテクスチャとして貼り付け、更にその上に集合住宅団地と戸建住宅団地計画案の CAD データを合成して作成したデータである。

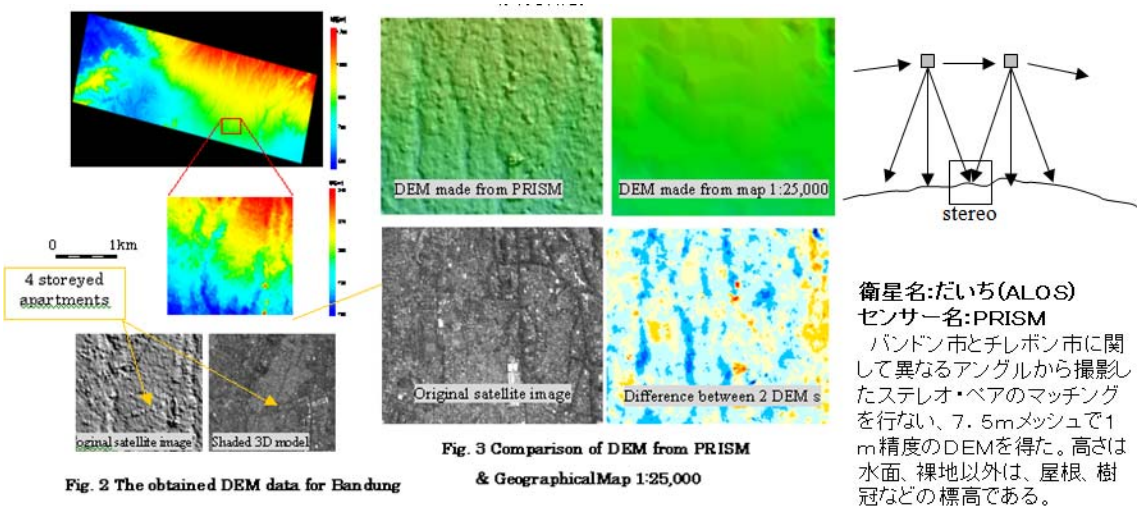


Fig. 2 The obtained DEM data for Bandung

Fig. 3 Comparison of DEM from PRISM & GeographicalMap 1:25,000

図 2-3-8 ステレオ衛星から復原したバンドン市内の地形と、従来データの比較

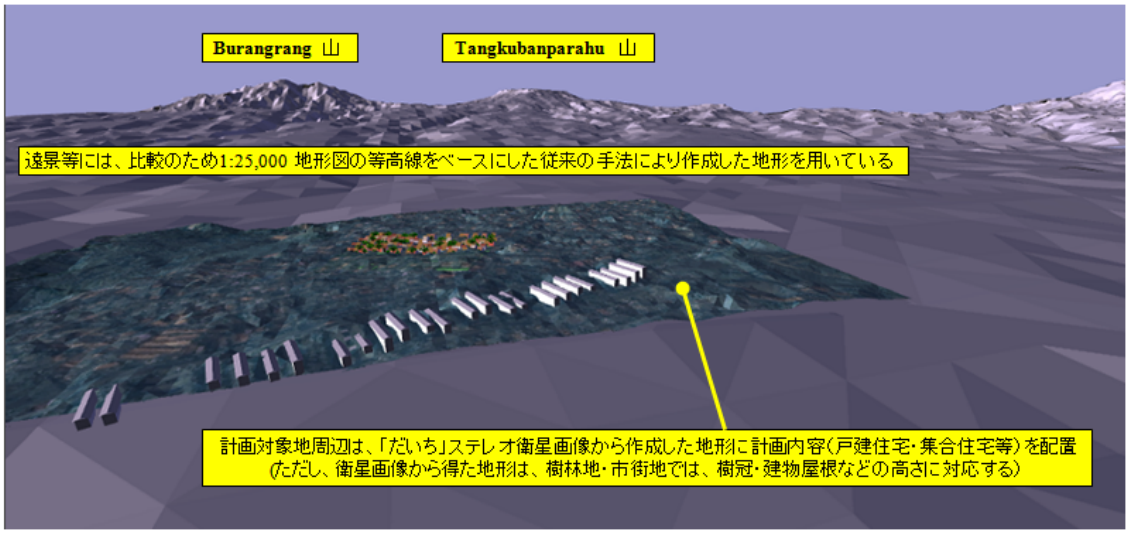


図 2-3-9 地形図等高線、ステレオ衛星画像解析結果、CAD 入力による計画案の合成

⑤ 基盤地図情報の利用

基盤地図情報（標高）は、一般地域においては、1:25,000 地図の等高線から補完して作成した１０ｍのメッシュデータであり、擁壁等の周辺で実際とはかなり異なる場合がある。一方、レーザー計測した結果は５ｍメッシュで作成されており、より精度が高い。1:2,500 地図に対応する精度である。

（３）団地の地盤面の作成

団地の地盤面が整地される前の地形データしか得られない場合には、地形の編集を行う必要がある。

① 景観シミュレータの高台整地機能の使用

前項のような各種の方法で作成した地形データに対して、ある一定の平面的なエリアを平坦に整地すると共に、周辺の地形との間に指定した勾配の法面（切土・盛土）を追加して隙間ない閉じた空間図形を生成する機能を、景観シミュレータのためのプラグインとして作成した。国総研のホームページからの公開プログラムに含まれている。この処理は、多角形として入力された地盤面から、指定した勾配で十分の法面テンプレートをまず作成した上で、これと地形との間の図形演算を行う処理を内部で行っている。

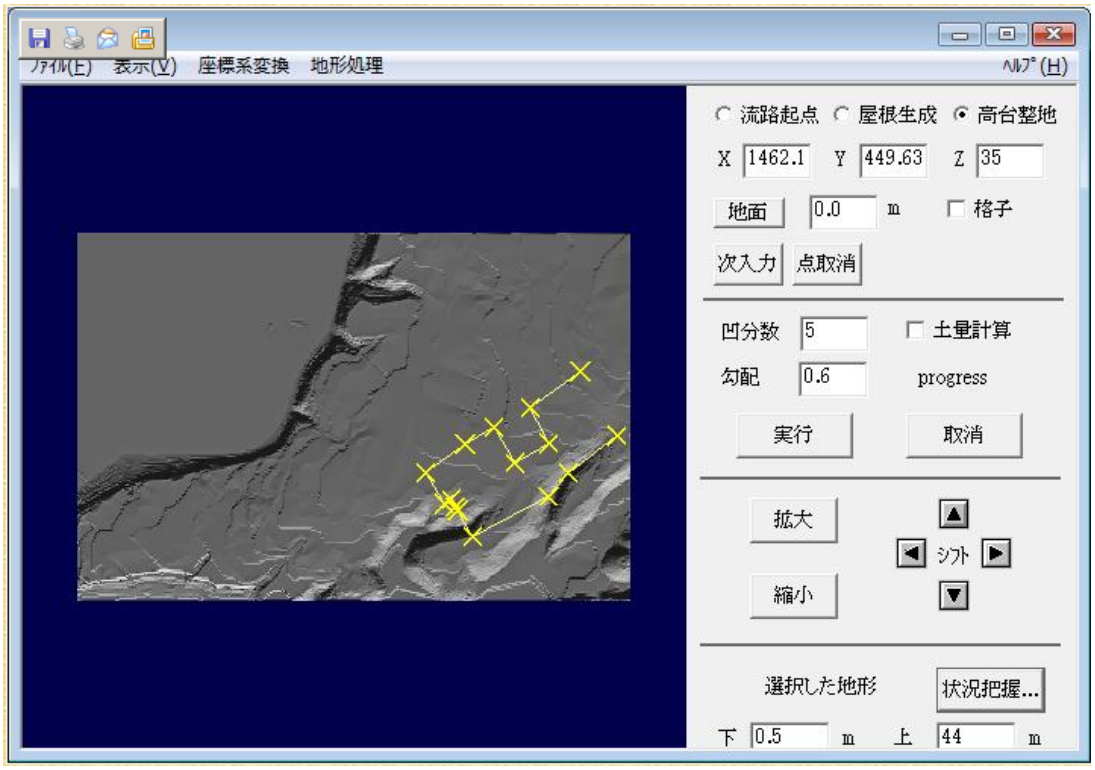


図 2-3-10 復原した地形の上に、造成エリアと標高を設定する

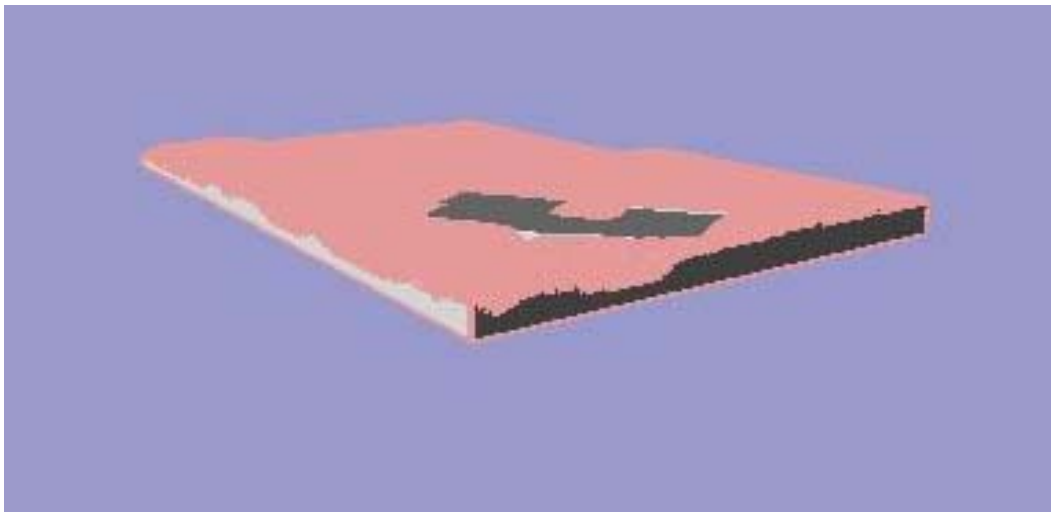


図 2-3-11 高台整地の図形演算結果

② 細部の編集

整地面、法面等の仕上げを、カラー、テクスチャ等を用いて編集する。

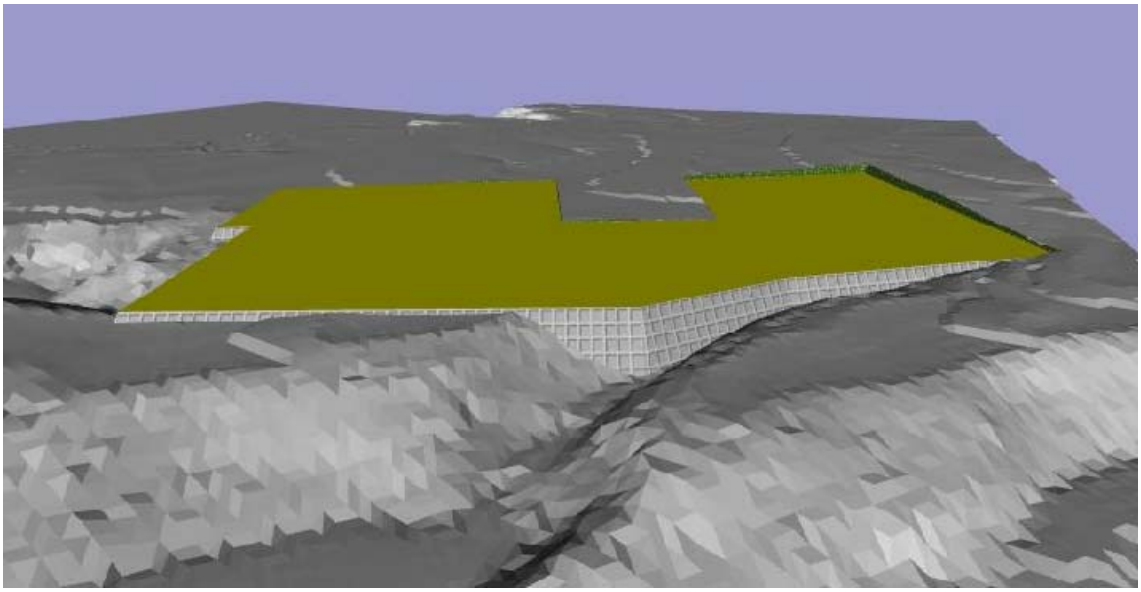


図 2-3-12 法面にコンクリート、植栽等を適用した。

（4）町並データの作成

① 建物の配置

原地形データの上に、被災前の空中写真をテクスチャとして貼り込んだデータを下地として、この上に個別にモデリングした道路と、古写真から個別に復元した住宅をそれぞれの位置に配置した結果を図 2-3-13 に示す。

図 2-3-13 古写真からモデリングした住宅を配置して復原した町並

② 属性データの付与

個々の住宅に、写真から判読できる階数、構造、屋根材料と共に、復原の根拠となった古写真の ID 等も属性として設定することにより、復原町並の三次元データを、資料整理・検索の手掛かりとしても利用できるようになる。

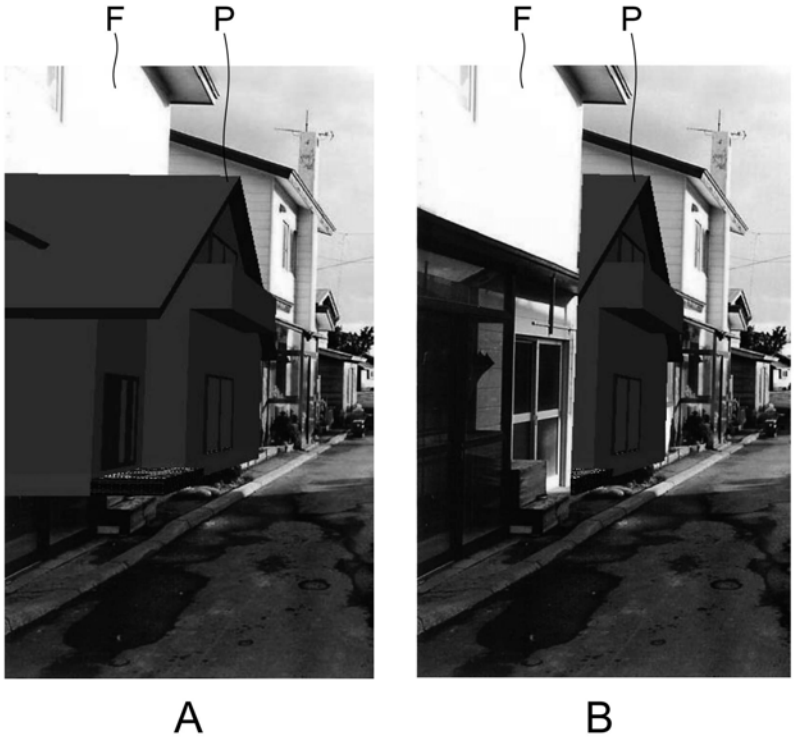
③ 地盤面の指定

GPS センサで計測した現在位置の直下に地盤の属性(&GROUND)が設定されている面が存在する場合には、表示に使用する高さを地盤面＋視点高さ 1.5m とする。これにより、造成等が行われた場所において、過去の地盤面に存在していた町並を地下に表示したり、現在の地盤と同じ高さに表示したり選択できるようにしている。但し、津波高さの水面を追加して建物の一部が水没しているように表現するなど、細部の工夫を行わなければ、単に縮小模型のような町並が足元に置かれているようにしか見えない。

④ 隠蔽面の作成

町並の中の空地に建物を表示する場合で、両隣の建物が現存するような場合には、両隣の建物の裏側に隠れる部分を表示しないようにする処理を行う必要がある。このためには、現存する両隣の建物の外壁を透明な板として作成し、復原建物に加えておく必要がある。

A では既存建物 F に復元建物 P が上書きされるため、手前にあるように見える。



B では復元建物 P に F の側面を透明な板として加えてあり、その背後となる P の部分が消える。

図 2-3-14 復原建物が隣の現存建物に隠れるような処理

（５）テクスチャファイルの作成

写真から立体的な復原を行った壁面等に、対応する元の写真の画像をテクスチャとして貼り込むことによって、再現性を高めることができる。奥尻島の住宅の復原作業に用いた

リアル・モデラーは、個々の建物の構成面を含む小さな画像を原写真から別ファイルとして切り出し、テクスチャとしてそれぞれの面に適用するような処理を行っている。

（６）ローカル座標系の計測と登録

① 座標原点の緯度・経度・標高の計測

町並のデータの座標原点の緯度経度標高を正しく指定することにより、携帯端末を用いて、正しい位置に表示することができる。図上計測は、例えば国土地理院のホームページから地図を検索すると、参考情報として図上の点の緯度経度を知ることができる。基盤地図情報は、緯度経度をベースとして作成されているため、補間により座標原点の緯度経度を把握することができる。また可能な場合には、閲覧に使用する携帯端末を用いて現場でGPS を用いて直接計測することも可能である。VC-3M を用いてシャッターを操作すると、たとえ正しい位置に表示が行われていない場合であっても、mobile.ja.scn に、シャッター時点で計測された GPS 座標値が記録されているので、これを用いてデータを準備し、精度を高めることができる。

計測された座標原点の緯度・経度・標高を表示コンテンツの中に入力するためには、３つの方法を用意した。第一は、「見たい場所」の一覧を表示するためのリスト(ModelIndex.txt)の中で指定する。第二は、メタファイルの中で指定する。第三は、データファイルの中で指定しておき、これをメタファイルで読み出して指定する。

② メタファイルへの登録

表示すべき建物等の三次元データの記録形式を解読するためのメタファイル（形式定義ファイル）を準備して、データファイルとのペアで、一覧ファイルに登録することにより、任意のデータ形式の記録を現場で再生することができる。

メタファイルには、データファイルを解読する方法を記述する以外に、固定的な形状を記述することもできる。GPS により計測される高さの精度が低い場合、あるいは現在の地盤面と異なる、嵩上げ前の過去の地盤面を用いて閲覧したい場合、あるいは手前にある現存建物に隠れる部分を消去して表示したい場合等に付加する情報は、データファイルの中に含めることもできるが、例えば隣接する建物は変化する可能性があるため、表示段階でのみ使用し、長期保存データには含めない方が良い場合もある。このような場合には、表示段階で使用するメタファイルの中に隣接建物の外形を固定的な形状として記述する方法が望ましい。同様に、記録された過去の町並とは異なる現在の地盤面から見下ろした風景を表示したい場合にも、メタファイルの中で現在の地盤面を記述する方法は有用である。

メタファイルの作成方法に関しては、２－４で解説する。

③ 表示選択用リスト・ファイルへの登録

わかりやすい見出し名、データファイル名、メタファイル名、および座標原点の緯度経度標高の計測値をリスト(ModelIndex.txt)に登録することにより、VC-3M で一覧表示の中から選択し、表示することができるようになる。この登録方法については、２－５で解説する。

２－４．メタファイルの作成

本節では、保存データに添付するメタファイルの作成方法を解説する。

（1）考え方

メタファイルは、一種のプログラム言語ないしスクリプト言語であり、以下の特徴をもつ。

① C言語に準拠した文法体系

・プログラムの中で変数および関数を定義して処理を記述する。
・プログラムの記述は、「;」により終了する。
・/*　*/によりコメントを挿入することができる。
・//によりコメント行を挿入することができる。
・main 関数から処理を開始する。
・if, switch, for, do, while 等により条件分岐、ループを記述する。
・if 文を簡略化した複合式「if(式１)?(式２):(式３)＞」の記法は使用しない。
・変数と一次元配列を使用する。
・exit 文により任意の場所で処理を中断・強制終了することができる。

一方で、以下の機能は省略されている

・fopen 関数を持たない。入力するデータファイルは最初から開かれている。
・入力するデータファイルには、固定長配列のようにアクセスすることができる。
・fscanf 関数は使用せず、代わりに scanf 関数（後述）によりデータファイルから読み込む。
・配列は１次元だけである
・構造体を定義し使用することはできない。固定長配列のみを使用する。
・malloc 関数等の動的メモリ管理を行わない。
（解読を行うデータがあらかじめ定められていることから、処理に必要な配列の長さは既知であり、大域変数として宣言する十分な長さの固定長配列により処理可能である）
・ポインタを使用することができない。
（セキュリティを高めるための配慮である）
・#define, #if, #include などのマクロは使用できない。
（「#」で始まる行は、コメントとして解釈される）
・リンカはなく、全ての処理を１本のメタファイルの中で記述する。

他方で、以下の機能が追加されている

・main 関数を省略する記述が可能である
（関数定義を行わない場合）
・宣言されない変数への代入が行われた場合、そのステップで宣言されたものとみなす
・scanf 関数の仕様が異なる
（代入する変数は１個のみとし、戻り値はパターンマッチの結果である。失敗した場合には、ポインタは動かない）

② LSS-G 形式のコマンドを部分集合として含むライブラリ関数

・LSS-G 形式のファイルを構成するコマンドは、C 言語の関数の書法である

変数名＝コマンド（引数）；　または

コマンド（引数１, 引数 2, ・・・）; という記述を基本とする

・LSS-G 形式では、例えば座標値を表す引数は数値（定数）であるが、ライブラリ関数においては、変数などから構成される数式を用いることができるように拡張した。

・LSS-G 形式では、引数の数を無制限とする記述を可としており、継承した。

・LSS-G 形式では、引数を省略する記述を可としており、継承した。

・メタファイルによる変換処理の記述を容易とするために、面の頂点列の末尾に頂点を追加する FACE_VERTEX コマンドを新たに追加した。

・厖大に蓄積されている LSS-G 形式のファイルを、記述している形状が既知でバグの無いメタファイルとしてコンパイル・実行することができ、これを用いて表示やデータベース処理などの利活用処理系の開発、テスト、デバッグを行うことができる。

③ OS に依存しない処理系の上でのコンパイルと実行

・コンパイラ、インタープリタのソースコードは C 言語で記述されているため、C コンパイラを幅広いプラットフォームに移植することができ、そこでメタファイルをコンパイルしてデータファイルを読み込む処理を実行することができる。

（２）メタファイルの種類

メタファイルは次のレベルのものが作成可能である

① 固定形状

データファイルを参照しない固定形状をメタファイルだけで記述することもできる。具体的には、FILE 関数しない固定的な（パラメトリックではない）LSS-G 形式のファイルは、固定形状のメタファイルとして処理できる。任意形式のファイルを入力するメタファイルとしては無意味であるが、様々な処理系のテストデータを簡単に用意することができる。長期保存されたデータの解読結果を利活用する新たな処理系の開発にあたって、初期の段階でのテストとデバッグのために、既存の LSS-G 形式のデータをサンプルとして用いる方法は、極めて有用である。

② パラメトリックな形状

小数のパラメータからパラメトリックな部品の生成をメタファイルで記述できる。景観シミュレータのために作成された外部関数のソースコードを修正（簡略化）することにより、関数としてメタファイルのメイン関数から呼び出すことができる。このとき、データファイルとはパラメータのみを定義する小さなファイルである。

③ 高度な文法形式をもつデータ形式の解読処理

景観シミュレータの外部関数として実装されているファイルコンバータが利用可能な場合には、ソースコードを修正することにより、当該データ形式のためのメタファイルを作成することができ、本処理系を用いて Windows 以外の環境においても利用することができ

る。重要な特徴は、このようにして作成したデータファイルとメタファイルの組は、景観シミュレータから独立して、未来の様々な処理系で利活用することができる、という点である。このことについては、利活用システムの開発を解説した第３章で詳しく解説する。

CAD や GIS アプリケーションを使用して入力された多くの三次元データは、高度な文法形式を有するデータ形式で保存されている。保存データに添付するメタファイル作成に際して、当該データ形式の全ての仕様に対応する必要はなく、あくまで保存データの中で参照されているキーワード等に対して対処できれば十分である。このため、メタファイルの中で解析に用いる固定長配列等も、必要最小限のサイズとすれば十分である。

（３）保存データの基本的な構造の把握

① 文字コード

２バイト系の文字に関して、様々な表記法が存在する。例えば、データの中に直接埋め込まれている場合（例 LandXML）と、ascii 文字を用いて間接的にコーディングされている場合がある（例 IFC）。

データの中に直接２バイト系コードが埋め込まれている場合には、ヘッダー部分に文字コードが表記されている場合がある（例えば XML 系）。しかし、例えば「メモ帳」エディターを用いて、ヘッダー部分を保ちながら異なる文字コードに簡単に変換することができる。従って、保存に際しては、ヘッダー部分の表記が正しいかを確認しておく必要がある。

利活用に用いるシステムで使用する文字コードが、原データとは異なる場合で、２バイト系文字コードを利活用システムでメッセージの表示などに使用したい場合、コード変換を行う必要がある。

コンパイラ処理系においては、コンパイラのソースコードに定数で文字コードを埋め込んでおき、これをバイナリで判定することにより、利活用処理系の文字コードを判別する方法を採っている。しかし、これは、メタファイルやデータファイルの文字コードと必ずしも一致することを意味するわけではない。例えば、Windows 上の開発環境である VS2005 の初期設定においては、utf-8 形式で保存されたソースコード中に埋め込まれたリテラル定数であっても、fprintf 関数等でファイルに出力した場合、Shift-JIS コードが出力される。

あるファイルの文字コードを明らかにするために、冒頭部分に、既知の２バイト系文字列を埋め込んでおく方法が有効である。ヘッダー部分に使用する文字コードを記載する方法がしばしば用いられているが、データファイルを編集・再保存が行われると、ヘッダー部分がそのままで、データ内部の２バイト系の文字列のコードが変換されているような場合があるので注意を要する。データファイルを保存処理する場合には、冒頭部分に記載された文字コードで保存するのが親切である。メタファイルの文字コードは、上記のように日本語などの２バイト系の文字コードを含むコメントやメッセージ出力が記述に含まれる場合に意味をもつ。処理の最初に logf(“仮想コンバータ”); といったメッセージ出力をメタファイルの処理の冒頭に含めておくことが、以後の保存処理と、後世の利活用にとって有効である。

② ブロック構成

多くのフォーマットにおいては、データ全体はヘッダー部分といくつかのデータブロックにより構成されている。この構造を把握し、例えばヘッダーを解読する header 関数と、データ本体を解読する main 関数から、header 関数と必要回数だけ data 関数を呼び出すようにメタファイルを構成するのがわかりやすい。

③ コマンドとパラメータの抽出

各種フォーマットにおいては、多くのキーワードやコマンドが定義されているが、実際の保存すべきデータにおいては、その一部しか使用されていない場合が多い。使用されていないキーワードを処理する関数を削除したメタファイルであっても、添付保存するメタファイルとしては十分である。

利用頻度が高いデータ形式の場合には、仕様書・定義書に定められたコマンド群の全てに対応した共通の親メタファイルから出発し、添付ファイルを作成する段階で、参照されない関数等を削除するような方法が能率的である。

本処理系においては、scanf(“文字列)”の形式でキーワードやコマンドを識別する。標準 C 言語の fscanf 関数においては、値を代入する変数がない場合には、この文字列のパターンが一致するところまでポインタが進み、戻り値は成否にかかわらず一定である。本処理系の scanf 関数においては、パターンが完全に一致した場合のみポインタが進み戻り値 1 を返し、そうでなければポインタは不変で戻り値 0 を返す。これにより、キーワードを抽出処理における gets 関数等を用いたバッファリングを省略している。

④ シンボル・テーブル

高度なデータ形式においては、シンボル（例えば繰り返し使用される図形の名称）を定義し、それを後の処理の中で参照することが行われる。このため、通常の場合にはシンボル・テーブルを作成する処理が行われる。

本処理系の場合には、解読対象とするデータファイルが固定されていることから、シンボルが最初に検出された位置の先頭アドレス（整数値）をもって、あるシンボルを特定することにより、長さが無制限のシンボルを、char[]配列を用いたバッファリングや、malloc/free 関数を用いたメモリ管理を行うことなく処理している。

（4）プログラミングとデバッグ

現在までに試作した 4 種類の実装形態に共通して、データファイルの利活用段階における解読処理は、メタファイルのコンパイル処理と、生成したコンバータの実行段階からなる。よって、エラーはコンパイル・エラーと、実行エラーに分けられる。コンパイル・エラーが生じた場合には、実行段階に移行することなく処理が終了する。

テキスト・エディタを用いて、メタファイルを編集し、データファイルと共にシステムに読み込むことにより、テスト･デバッグを行うことができる。VC-3M の場合には、Android 上で動作するテキスト・エディタにより現場で編集・修正を行い、直ちにコンパイルと実行（読み込み→表示）を行うこともできる。

保存データを現場での再生閲覧するために使用する処理系は、なるべくシンプルで軽く、バグの少ないものが望ましいことから、現場での閲覧のための処理系に多くのデバッグ機能を追加する計画はない。現時点では、デスクトップ PC の上で動作する VC-1C または VC-2V を用いて、メタファイル完成直前までのメタファイルのプログラムとデバッグ作業を行うのが能率的である。

① コンパイル時のエラーメッセージ

コンパイル・エラーを、表 2-4-1 に一覧する。エラー処理は、仮想コンバータに含まれる err_ss 関数(cci_misc.c)を通じて出力される。この関数は、二つの文字列（エラーメッセージ）を引数として取得し、エラー内容と、デバッグの参考となる文字列を出力する。また、二つ目の文字列を空とする err_s 関数、二つ目の文字列を数値とする err_fi 関数もエラーメッセージ出力に用いている。

エラーの原因となったメタファイル文法に関しては資料４－１に、またライブラリ関数の使用方法の詳細については、資料４－２に解説する。更に、新たなデータ形式に対するメタファイルの作成のための参考として、資料４－６に各種三次元データ形式とメタファイル作成例を解説する。

表 2-4-1 コンパイル・エラー　メッセージ一覧

(err_ss 関数が出力するメッセージ)
未定義の二項演算子(Int) (code:132)
未定義の二項演算子(Float) (code:150)
未定義の演算子(Quat) (code:171)
未定義の関数 (code:319)
scanf 書式(書式文字列)(file:161,173,253,263)
ミスマッチ発生(文字列型書式文字列) (file:446)
// scanf コマンドで、文字列型入力を含む書式文字列に従った入力に失敗した場合, EOF 到達など
（がない(引数が ON または OFF のコマンド) (pars:123)
ON でも OFF でもない(引数が ON または OFF のコマンド) (pars:129)
）がない(引数が ON または OFF のコマンド) (pars:131)
；がない(引数が ON または OFF のコマンド) (pars:133)
構文エラー（解釈できないトークン） (pars:237)
識別子が重複している(宣言された関数名) (pars:274)
関数が再定義されている(宣言された関数名) (pars:319)
関数プロトタイプが不一致(宣言された関数名) (pars:323)
｝を検出(main()関数が無い形式) (pars:412)
ReadOnly モードで書き込み statement(コマンド) (pars:599,992)
//データベースから出力ファイルを生成する形式定義ファイルに、データベースへの書き込みコマンドを検出
未定義の関数を使用(要求された関数名) (pars:650)

不正な記述(文として解釈不能な記述) (pars:678,1177)
型が不適切(=代入) (pars:786)
未解決の数値型の組み合わせ FI (pars:850)
//異なる数値型への変換が解決できない場合
引数の数値型が不適(三角関数) (pars:879,888)
void 型関数が式の中で使われている(使われた関数名)(pars:975,1063)
不明な型の変数が参照された(変数名) (pars:1074)
関数の引数個数が不一致(関数名)(pars:1214)
argsV 引数に二重の＊あり token.text (pars:1248)
argsV 引数がシンボルでない token.text (pars:1253)
argsV 引数が変数でない token.text (pars:1257)
argsV 引数が整数型の変数でない token.text (pars:1260)
//変数名を引数とする関数における引数の指定方法にエラーがある場合
引数の型が不正(LINK_XFORM) (pars:1342)
第二引数｛LOAD,PRE,POST｝がない(LINK_XFORM)(pars:1355)
第二引数｛LOAD,PRE,POST｝が不正(LINK_XFORM)（pars:1372）
第三引数｛IDENTITY,TRANSLATE・・・｝が不足(LINK_XFORM)(pars:1377)
第三引数｛IDENTITY,TRANSLATE・・・｝が不正(LINK_XFORM) (pars:1397)
数値の数が不整合(LINK_XFORM) (pars:1408)
printf %n に対応する引数が変数でない(引数部分の記述)(pars:1478)
代入がない scanf の書式文字列が不正（書式文字列）(pars:1585)
scanf の書式文字列の次のトークンが不正(トークン) (pars:1589)
scanf 引数が変数でない(引数部分の記述)(pars:1606)
scanf 引数が整数変数でも浮動小数変数でも四元数でもない(引数部分の記述) (pars:1610)
scanf 型指定と代入する変数の型が異なる(書式文字列) (pars:1630)
四元数型引数が二以上(COORD) (pars:1782)
）が無い（COORD）(pars:1783)
引数がない（COORD）(pars:1789)
浮動小数型、整数型の引数の数が３に不足（COORD） (pars:1796)
浮動小数型、整数型の引数の数が３を超過（COORD） (pars:1797)
引数がない（NORMAL）(pars:1834)
引数の数が３に不足（NORMAL） (pars:1836)
引数の数が３を超過（NORMAL）(pars:1837)
引数がない（TCOORD）(pars:1890)
引数の数が２に不足（TCOORD） (pars:1892)
引数の数が２を超過（TCOORD） (pars:1893)

引数がない（COLOR）(pars:1920)
引数の数が３に不足（COLOR）(pars:1922)
引数の数が４を超過（COLOR）(pars:1923)
引数の数が２ではない（FACE_COLOR）(pars:2019)
引数の数が２ではない（FACE_TEXTURE）(pars:2023)
引数の数が２ではない（FACE_NORMAL）(pars:2027)
引数の数が２ではない（FACE_MATERIAL）(pars:2030)
引数の数が２ではない（LINE_COLOR）(pars:2040)
引数の数が２ではない（LINE_NORMAL）(pars:2045)
引数の数が２ではない（LINE_TEXTURE）(pars:2049)
引数の数が２ではない（LINE_MATERIAL）(pars:2053)
引数の数が２ではない（GROUP_MATERIAL）(pars:2057)
引数の数が２ではない（GROUP_TEXTURE）(pars:2061)
引数の数が２ではない（GROUP_ATTRIBUTE）(pars:2065)
引数の数が２ではない（LINK）(pars:2069)
引数の数が１ではない（TIME）(pars:2098,2101)
引数の数が１３ではない（CAMERA）(pars:2108)
引数の数が８ではない（LIGHT）(pars:2113)
引数の数が８を超過（LIGHT_GROUP）(pars:2118)
引数の数が１ではない（MODEL）(pars:2130)
引数が文字列型ではない (pars:2132)
引数の数が１ではない（IMAGE）(pars:2144)
引数が文字列型ではない（IMAGE）(pars:2146)
第一引数：種別が IKE ではない（EFFECT）(pars:2152)
第二引数がない（EFFECT(IKE)）(pars:2156)
緯度の数値はダブルクォーテーションマークで括る（EFFECT(IKE)）(pars:2158)
第三引数がない（EFFECT(IKE)）(pars:2161)
経度の数値はダブルクォーテーションマークで括る（EFFECT(IKE)）(pars:2163)
第四引数がない（EFFECT(IKE)）(pars:2166)
高さの数値はダブルクォーテーションマークで括る（EFFECT(IKE)）(pars:2168)
第五引数がない（EFFECT(IKE)）(pars:2171)
データファイル名メタファイル名はダブルクォーテーションマークで括る EFFECT(IKE) (pars:2173)
引数の数が６を超過（EFFECT(IKE)）(pars:2196)
引数の数が１ではない（EFFECT_GROUP）(pars:2201)
引数の数が７ではない（SCENE）(pars:2210)

引数がある LEN (pars:2216)
引数がある SIORI (pars:2220)
引数の数が 1 ではない SEEK (pars:2224)
引数が整数型ではない SEEK (pars:2225)
引数がある GETC (pars:2229)
引数がある GETS (pars:2233)
引数の数が 1 ではない GETINT (pars:2237)
引数の数が 1 ではない GETFLOAT (pars:2241)
文字列引数の数が 1 ではない（ _Q()） (pars:2252)
引数がない _Q() (pars:2271)
引数の数が 4 未満（ _Q()）(pars:2273)
引数の数が 4 を超過（ _Q()） (pars:2274)
引数の数が 1 ではない（四元数成分取得）(pars:2284)
引数が四元数型ではない（四元数成分取得）(pars:2285)
引数の数が 1 ではない（DES⇔IKE）(pars:2295)
引数が四元数ではない（DES⇔IKE）(pars:2296)
引数が不足（座標系変換） (pars:2305)
第一引数が整数型でない（座標系変換）(pars:2306)
引数の数が 2 を超過（座標系変換）(pars:2309)
第二引数が四元数型でない（座標系変換(pars:2310)
関数に引数がある（SQL 関数）(pars:2326)
型指定誤り（型名称）(pars:2396)
名称が不適切：予約語を用いた宣言等（変数、関数等の名称）(pars:2408)
多次元配列は宣言できない（[）(pars:2434)
関数の引数の型が不適（引数名）(pars:2463)
%c 変換で幅が 2 以上 （scanf）(pars:2646)
未定義の識別子（名称）(tbl:128)
識別子が重複している（名称） (tbl:168)
エスケープ文字の 16 進表現が不正（¥x）(tkn:362)
不正なトークン（トークン）(tkn412)
入力ファイルが開いていない nextCh() (tkn:440)

(err_s 関数への参照)
浮動小数二項演算結果の数値型が不明 (code:160)
四元数二項演算結果の数値型が不明 (code:181)
不正な左辺値 (code:192)

ゼロ除算(code:275)
main 関数がない (code:313)
関数の；または｛がない (pars:306)
関数の引数の型が不正 (pars:342)
本処理系では非 void 型関数の末尾には単独の return 文が必要 (pars:356)
main 関数が int 型でない (pars:368)
不正な break (pars:437)
case 式が定数式でない (pars:452)
case 式の値が重複している (pars:457)
対応する switch 文がない (pars:468)
default が重複している (pars:470)
do 終端の while がない (pars:550)
return 文に戻り値がない (pars:567)
void 型関数に値を返す return 文がある (pars:570)
関数の型と戻り値の型が異なる (pars:571)
シンボルテーブルパンク (pars:633)
COMMENT 文の位置が不正 (pars:667)
意図しない終了。‘}’が不足か？(pars:674)
continue がループ文内にない (pars:705)
同じ型の間で型変換しようとした (pars:738)
未実装の型変換 (pars:745)
二項演算の型調整不能 (pars:870)
&の次がアイデンティファイアでない (pars:897)
未登録のアイデンティファイア (pars:905)
＆の後の配列の数値型が不正 (pars:919)
添字指定がない (pars:924, 1105,1493)
演算子[+-*/]=の数値型が未対応 (pars:1125)
演算子[+-*/]=の前後の数値型が違う (pars:1131)
演算子[+-*/]=の種類が不正 (pars:1141,1152,1163)
不明な型 (pars:1169)
添字指定がない (pars:1293,1667,2420)
printf, logf の第一引数に書式文字列がない (pars:1430)
書式文字列の代入に対応する引数が無い (pars:1441)
printf の引数は２個まで(pars:1502,1549)
printf の第１引数が不正 (pars:1573)
scanf の引数は２個まで(pars:1672)

scanf の整数系書式が不正 (pars:1677)
scanf の浮動小数系書式が不正 (pars:1681)
scanf の QUAT 系書式が不正 (pars:1685)
TEXTURE(*[変数名でないもの]) (pars:1709)
TEXTURE(*[未定義の変数]) (pars:1717)
TEXTURE の引数が不正(pars:1720)
TEXTURE の引数の数が 1 ではない(pars:1728)
MATERIAL(*[変数名でないもの]) (pars:1744)
MATERIAL(*[未定義の変数]) (pars:1753)
MATERIAL の引数が不正(pars:1756)
MATERIAL の引数の数が 1 ではない(pars:1764)
想定外の EOF (pars:1960)
不正な添字 (pars:2427)
配列幅指定が整数定数式ではない (pars:2430)
不明な型 (pars:2448)
書式文字列：QUAT (pars:2560)
変数型誤り(void) (tbl:31)
記号表 over (tbl:37)
定数の文字数が過大 (tkn:306)
不正な文字定数(tkn:312)
文字列リテラルが長すぎる (tkn:392)
文字列リテラルが閉じていない (tkn:393)
不正な文字が使われている (tkn:405)
/*に対応する*/がない (tkn:468)
/*がないのに*/を検出した(tkn:474)
（文字列）の前に（文字）が必要 (tkn:526)
//トークン１が現れる前に、トークン２があるべきところ
(err_fi 関数への参照)
#コードが%d ステップを超えた (code:81)
メモリ不足：残り (code:222)
整数型引数を異なる書式で出力しようとした (code:1453)
浮動小数型引数を異なる書式で出力しようとした(code:1459)
四元数型引数を異なる書式で出力しようとした(code:1465)
printf の出力数値型が不適 (pars:1470)
(war_ss 関数への参照)
main()がない形式として再解釈 関数の外で変数が左辺値 (pars:230)

main()がない形式として再解釈 プログラムの最終行に到達 (pars:244)
右辺が NON_T conv (pars:734)
＊の後が整数型ではない（factor）(pars:947)
配列の添字が FLOAT 値 (pars:1091)
switch(kd)において kd が不正(pars:1194)
代入先がない scanf の書式文字列に変換%が指定されている(pars:1582)
FILE 関数はサポートしません cci_pars (pars:2006)
整数型の第%d 引数を浮動小数型に変換,_Q (pars:2265)

② 実行時のエラーメッセージ

コンパイルが成功し、データの読み込みを開始してから発生し、ログファイルに出力される実行時エラーを、表 2-4-2 に一覧する。

表 2-4-2 実行時エラー一覧

ゼロ除算(code:347)
ゼロ浮動除算(code:348)
プログラムカウンタが範囲外(code:415)
スタック・オーバフロー(code:426)
スタック・アンダーフロー(code:430)
time over(code:435)
スタックメモリオーバー：フレーム確保失敗(code:463)
配列の添字が負(code:618)
配列の添字が過大(code:619)
outfile が開いていない(code:705,736)
引数の無い PRINTF で、書式文字列に出力項目がある(code:718)
SCANF の QUAT 書式が不正(code:822)
PRINTF の QUAT 書式が不正(code:829)
printf(“%s”,adr)で、メモリ範囲外の番地が要求された(code:836)
GN テーブルパンク(dml:156,192)
オーバーフロー,fCOORD(dml:256)
オーバーフロー,COORD (dml:380)
オーバーフロー,TEXTURE (dml:425)
テクスチャ画像ファイルの入力に失敗 (dml:433)
オーバーフロー,TCOORD (dml:493)
オーバーフロー,NORMAL (dml:525)
オーバーフロー,COLOR (dml:537,565)
GetG<0 (code:617)
NG<=GetG (code:621)

GetG 失敗(code:626)
group_face 処理する面のテクスチャ ID が登録範囲外(dml:718)
group_face 処理する面の頂点 ID が登録範囲外(dml:739)
オーバーフロー,LINK (code:827,843)
LINK で親グループが見つからない(dml:850)
LINK で子グループが見つからない(dml:856)
LINK_XFORM の数値の数が不整合(dml:890)
LINK_XFORM の引数の数が指定方法と不整合 (dml:893)
LINK_XFORM の第二引数（順序）が不正 (dml:900)
LINK_XFORM の第三引数（パラメータ型）が不正 (dml:915)
TRANSLATE 引数の個数が不整合(dml:926)
ROTATE_X 引数の個数が不整合(dml:933)
ROTATE_Y 引数の個数が不整合(dml:938)
ROTATE_Z 引数の個数が不整合(dml:943)
ROTATE_A 引数の個数が不整合(dml:948)
SCALE 引数の個数が不整合(dml:956)
MATRIX 引数の個数が不整合(dml:963)
LINK の type が不正(dml:968)
LINK の order が不正(dml:974)
F()関数使用前準備不足(dml:1190)
Ft()で頂点が 3 未満(dml:1260)
Ft()準備不足(dml:1282)
Nx()関数使用前準備不足 (dml:1526)
Ny()関数使用前準備不足 (dml:1536)
Nz()関数使用前準備不足 (dml:1546)
Cr()関数使用前準備不足 (dml:1672)
Cg()関数使用前準備不足 (dml:1682)
Cb()関数使用前準備不足 (dml:1692)
Ca()関数使用前準備不足 (dml:1702)
範囲外(Lm) (dml:1767)
リンクが未選択(Li) (dml:1779)
リンクが未選択(Ls) (dml:1801)
リンクが未選択(Lr) (dml:1817)
リンクが未選択(Lt) (dml:1838)
リンクの子グループの ID が負(Lc)(dml:1918, sql:1005))
リンクの子グループの ID が過大(Lc)(dml:1919,sql:1006)

VERTEX に５番目以降の引数（face:91）
s (face(97,102)
頂点リストを格納するメモリ取得失敗(face:126)
FACE コマンドでスタック・アンダーフロー(face:184)
FACE,LINE のリスト FL 伸長失敗(face:224)
FACE_VERTEX の引数が不足（face:250)
FACE_VERTEX 失敗（face:275)
GROUP_FACE でスタック・アンダーフロー（face:318)
GROUP_FACE 処理で、参照するグループ ID が過大（face:328)
GROUP_FACE 処理で、参照するグループ ID が過小（face:334)
GROUP_FACE 処理で、参照する面の ID が過小（face:344)
GROUP_FACE 処理で、参照する面の ID が過大（face:350)
GROUP_FACE 処理で、参照する面の頂点 ID が不正（face:368)
GROUP_FACE 処理で、参照する面の頂点 ID が未定義（face:377)
GROUP_FACE 処理で、頂点リストが空（face:398)
未定義の面[ID]を参照(FACE_COLOR), NF=面総数(face:424)
未定義の色[ID]を参照(FACE_COLOR),NC=色総数(face:427)
未定義の面を参照(FACE_NORMAL) (face(454)
未定義の法線を参照(FACE_NORMAL) (face:455)
未定義の面を参照(FACE_TEXTURE)(face(468)
未定義のテクスチャを参照(FACE_TEXTURE) (face:469)
未定義の面を参照(FACE_MATERIAL) (face:486)
FP==NULL,GETC() (file:71)
FP==NULL,WATC() (file:80)
FP==NULL,INTEGER() (file:92)
FP==NULL,FLOATER() (file:108)
FP==NULL,SIORI() (file:122)
FP==NULL,SEEK() (file:130)
書式文字列への参照アドレスが NULL(inputdata) (file:222)
書式文字列の参照が「%」ではない (file:226)
FP==NULL,SCANF() (file:400)
scanf で、取得した整数値の格納先がない（file:415,425)
scanf で、取得した浮動小数値の格納先がない（file:430)
scanf で、文字列型の格納先がない（file:435)
シンボルテーブルパンク(file:458)
scanf で、数値を取得しないのに格納先がある（file473)

SCANF の QUAT 書式が不正(file:540)
scanf で、取得した四元数値の格納先がない(file:551)
オーバーフロー,COORD (ip:176)
LINK_XFORM のタイプ指定が不正(ip:501)
#LINK_XFORM の引数とタイプが不整合(ip:504,sql:1411)
LINK_XFORM の第二引数（順序）が不正(ip:510,sql:1417)
LINK_XFORM の第三引数（パラメータ型）が不正(ip:522,sql:1430)
//'Strdup fail (misc:207)
Free(NULL) (misc:311)
LINK_XFORM の数値の数が不整合(sql:1408)
TRANSLATE 引数の個数が不整合(sql:1437)
ROTATE_X 引数の個数が不整合(sql:1444)
ROTATE_Y 引数の個数が不整合(sql:1449)
ROTATE_Z 引数の個数が不整合(sql:1454)
ROTATE_A 引数の個数が不整合(sql:1459)
SCALE 引数の個数が不整合(sql:1467)
MATRIX 引数の個数が不整合(sql:1474)
LINK の type が不正(sql:1479)
LINK の order が不正(sql:1488)

③ メタファイル中の logf 関数によるメッセージ出力

メタファイルのプログラマは、logf 関数を用いて、実行中の処理状態を確認するメッセージをログファイルに出力することができる。デバッグが終了し最終的に完成したメタファイルからは削除される。logf 関数の引数は、書式文字列と出力データから成り、printf 関数とほぼ同等である。利活用形態として、VC-2V、VC-4D においてメモリ上あるいはデータベース上の町並み、建物等を読み出して、ファイル出力を行う場合には、printf 関数が最終的にファイル出力されるデータファイルへの出力を行うのに対して、logf 関数は、デバッグのためのログファイルへの出力を行う。

（５）配列の長さ等の最適化

本処理系におけるメタファイルは、あるファイル形式で記述された様々な不特定のデータファイルを処理するための汎用のものである必要はなく、添付対象とする特定のデータファイルを変換できれば十分であり、むしろ無用なメモリの浪費は少ない方が望ましい。

このため、処理中に生成する配列等の必要幅は既知であり、メタファイルの冒頭で宣言する定数値も最小限とすることが望ましい。

よって変換処理が正しく行われたことを確認した上で、処理終了時点でのリストの長さ等を把握し、これに基づいて冒頭の宣言において配列幅を設定する定数に反映させ修正することにより、過大なメモリを配列の領域確保のために浪費することを防ぐことができる。

（６）不要な関数等の削除

仕様書・定義書に定められているデータ形式を用いた保存ファイルであっても、通常はその中で使用されているキーワードやコマンドは、用意されている利用可能なものの一部にすぎない。従って、変換処理を行った結果、参照されることが一度もなかったようなキーワードやコマンドに対応するメタファイルの処理（関数等）は、メタファイルから削除することにより、メタファイルを小さく簡潔にすることができる。

逆に、新たにメタファイルを作成するデータ形式に関しては、最初にまず参照されているコマンドの一覧を作成し、それらについて処理内容を実装する方法が効率的である。データファイルを作成する際に使用したアプリケーション（CAD、モデラー等）が自動的に出力するコマンドで、モデリング作業に無関係にデフォルト値を宣言しているだけのような、意味の無いコマンドに関しては、そのコマンドに関連した引数パート等を最後まで読み飛ばすだけの処理を作成すれば良い。

（７）データファイルの真贋判定、改竄検知

メタファイルは、添付するデータファイルが特定のものであることから、真贋性に関する検査の処理を付け加えることができる。ファイルの長さ、チェックサム、あるいはより高度なデータファイル固有の値をデータファイルに関して解析し、その値がオリジナルのデータファイルと一致することを確認することにより、メタファイルとデータファイルの対応関係を確認することができる。この固有の値は、定数値としてメタファイルの中に書き込むことができる。ごく簡単なファイルサイズの検査、チェックサムの処理例をリスト 2-4-1 に示す。より高度な検査をメタファイルで記述することも可能である。

但しこの方法が有効であるためには、データファイルとメタファイルの双方が同時に偽造されていないことを保証する必要があり、そのためにはメタファイルをデータファイルと関連づけられた別の場所に非公開で保管するような管理が必要であろう。一般にメタファイルは、データファイルよりも遥かにサイズが小さい。

リスト 2-4-1 データファイルの偽造・改竄の検出処理

```
int check(){
  int c,sum;
  if( LEN() != 固有の値) return 0; //長さの検査
  for(sum=0; 0 <= (c=GETC()); sum += c); //チェックサム検査
  if( sum != 固有の値) return 0;
  ・・・その他の高度な検査・・・
  return 1;
}

int main(){
  ・・・・
```

```
    if( !check() ) logf("データファイルは改竄されている");
    ・・・・
}
```

２－５．三次元アーカイブスの作成

（１）携帯端末用ロードデータの作成

① 選択用リスト modelindex.txt の作成

保存データと、これを解読するための処理を定義したメタファイルの組を、必要な数だけ modelindex.txt ファイルに登録することにより、VC-3M（むかしめがね）を用いて現場で表示することが可能となる。その際に、モデルを記述する原点の緯度・経度・標高を併せて指定することにより、記録された建物等を画面上で過去に存在していた位置あるいは建設が計画されていた位置に表示することができる。この一覧データは、csv 形式で記録されており、１つの行は、

表示名,メタファイル名,データファイル名,緯度,経度,標高

を含んでいる。表示名は、モデル選択画面（「見たい場所」）で一覧表示される、各場所の名称である。緯度、経度、標高は、モデルを記述している座標系の原点の位置を示している。

奥尻島の３地区を登録したデータの例を、表 2-5-1 に示す。

表 2-5-1 奥尻島の表示データのための、modelindex.txt の例

みさきこうえん,LSSG.cmm,misaki.geo,42.0524,139.4505,2.0
みさきこうえん津波２,LSSG.cmm,misakitsunami2.geo,42.0524,139.4505,2.0
はまかぜこうえん０,LSSG.cmm,hamakaze2.geo,42.060198,139.4494,0.0
はまかぜこうえん６津波,LSSG.cmm,hamakaze6t.geo,42.060198,139.4494,0.0
よねおかけんえい,LSSG.cmm,yoneoka.geo,42.0682,139.44435,0.0

2014 年 10 月 8 日に現地体験教室に使用した際のファイルであり、図 2-1-3 に示した一覧表示画面に対応している。この緯度、経度は、あらかじめ図上で計測した上で、現地において地図上で確認できる場所に赴き、VC-3M をセットアップした携帯端末を用いて計測を行って確認した。

古写真から復原したデータの場合には、テクスチャファイルを texture ディレクトリの下に用意しておく必要がある。

なお、建物等を記述するデータの作成に使用した座標軸の原点位置の緯度・経度・標高は、この一覧ファイルに記述する以外に、モデルデータの中に記録されている数値を利用することも、メタファイルの中で記録することもできる。また、シャッターを操作した際に記録され、mobile.ja.scn ファイルとして保存されるデータの中にも、撮影場所の緯度、経度、標高が記録される。詳細は、３－５で解説する。

② データファイルおよびメタファイルの格納

携帯端末の内蔵 SD カードの下に、VirtualConverter という名称のディレクトリを作成し、この直下に上記の modelindex.txt を格納する。このディレクトリには、シャッターボタンが押された場合の時刻、場所、姿勢（カメラアングル）を記録した mobile.ja.scn ファイルが作成される。また、データをロードする際のエラーメッセージ等を記録したログフ

ファイル log.txt が作成される。

便宜のためアプリをセットアップするための VC-3M.apk ファイルもここに置いている。

このディレクトリの配下に、サブディレクトリとして以下のものを作成し、それぞれに必要なデータを格納する。

1) data：データファイルを格納する。一つの建物が一つのファイルである。それぞれの建物を記述するデータファイルは、異なる形式であってもかまわない。上記の modelindex.txt に登録されているメタファイルとデータファイルが対応していれば読み込むことができる。
2) meta：メタファイルを格納する。データファイルの形式が同一であれば、一つのメタファイルで複数のデータファイルに共通に使用することができる。
3) texture：データファイルの表示にテクスチャを使用する場合には、画像ファイルを格納する。
4) sensorvalue：デバッグのために、センサーの計測値を記録したファイルを格納するためのディレクトリである。初期状態で空であって構わない。
5) image：シャッターボタンが押された場合に、背景画像を JPEG 形式のファイルとして保存する。初期状態で空であっても構わない。
6) thumbnail：保存した背景画像を縮小したサムネイル画像を保存する。記録再生を行おうとする場合に、マトリクス表示する背景画像として使用する。ディレクトリが存在していれば、初期状態で空であっても構わない。

（２）アーカイブスを使用した現場活動の記録の保存

現場での閲覧の後に、以下のファイルが作成される。

① mobile.ja.scn

背景画像名、シャッター時点での GPS 座標、端末の姿勢、キャリブレーション値を記録する。VirtualConverter ディレクトリ直下に作成される。

初期状態で存在しなければ、最初の終了時点で新たに作成される。

既存のものがあれば、シャッター操作により、末尾にデータが追記され、アプリ終了時点で保存される。終了前に異常終了（電池切れ等）した場合には、記録は失われる。また、記録再生画面で削除操作が行われた場合には、対応する記録は削除される。但し、関連する背景画像ファイル等は温存される。

② Images ディレクトリの下に作成される背景画像

シャッターボタンが押された時点で記録される背景画像(jpg 形式)である。

ファイル名称は、撮影日時・時刻に基づき、以下のように自動的に決定される。

[例]　BI20141009.095919.jpg

[解釈]　2014 年 10 月 9 日 9 時 59 分 19 秒に保存された背景画像(Back Image)

③ thumbnail ディレクトリの下に格納される見出し用縮小画像

TBI20141009.095919.jpg

2014 年 10 月 9 日 9 時 59 分 19 秒に保存された画像の縮小ファイル(40-50KB 程度)

④ log.txt

仮想コンバータが起動する度に、言い換えると「見たい場所」が選択される度に作成される。メタファイルのコンパイル・エラーや、実行形式のランタイム・エラーを記録する。

utf-8 形式で作成されており、エラーが生じた場合には、このログファイルを表示して終了する。

エラーが存在しない場合でも、作成されるが表示はせず、そのまま表示処理に進む。動作状況を把握するために使用することができる。

（３）WEB 配信用の圧縮ファイルの作成

公開可能なアーカイブスに関して、携帯端末に実装するデータを配信するためには、必要とするファイルを一つのファイルに圧縮・梱包したファイルを作成するのが便利である。圧縮・解凍の作業を行うためには、携帯端末側で利用するファイル・マネージャ等に、ディレクトリ単位の圧縮と解凍を行う機能が備わっており、ZIP 圧縮が用いられている。

携帯端末で VC-3M と三次元アーカイブスをロードし実行するためには、VirtualConverter ディレクトリの内部に、以下のファイルが用意されている必要がある。

① ModelIndex.txt（見たい場所、建物などの一覧）

VirtualConverter ディレクトリ直下

② Meta ディレクトリの下に格納されたメタファイル

データファイルのそれぞれを解読するための処理を定義したメタファイルである。

一つのメタファイルを、同一形式の複数のデータファイルに共通に用いることができる。

また、固定形状のデータをメタファイルだけで記述することもできる。

③ Data ディレクトリの下に格納されたデータファイル

様々な記録形式で建物等の三次元形状を定義したファイルである。

④ Texture ディレクトリの下に格納されたテクスチャファイル

建物の屋根面、壁面等にテクスチャ画像を定義する場合には、このディレクトリに格納する。

⑤ VC-3M.apk

アプリのセットアップファイルで、VirtualConverter ディレクトリ直下に置かれる。但し、このファイルがどこにあってもセットアップは可能であり、既にセットアップが行われている場合には、新たなデータのロードに際して、再セットアップは不要である。

（４）長期保存媒体への記録と再生

上記のように、建物や町並を記録した三次元データの解読方法がメタファイルにより記述され、さらにローカル座標系の原点位置の緯度・経度・標高が明らかであれば、将来時点において、同じ場所にデータ表示することが可能であり、表示以外にも様々な利活用方法が想定できる。そのためには、長期保存が可能な記録媒体への書き込み・読み出しが必要である。

本研究においては、２－６、２－７において、RAID１等によりメンテナンスが保証されているサーバー上に保存する方法を解説すると共に、頻繁なアクセスが期待されない、いわゆるコールドデータに関しては、ガラス等の長期保存可能な媒体に記録する方法をテストした。具体的には、Housemap.dll という、住宅に関する画像データ等を扱う、景観シミュレータのためのプラグインを作成し、その機能の一部として、データファイルとメタファイルの組を、二次元のビットパターンに変換して、レーザー加工機を駆動するためのデータとして出力する処理を行った。この記録を含む記憶媒体（電子棟札、電子定礎等とも呼ぶべきもの）を将来読み出す時点では、イメージスキャナでビットパターンを取り出す。これについては更に実用化に向けて研究・試作中であり、本書では割愛する。

２－６．サーバとデータベースの管理

（１）はじめに

仮想コンバータ VC-4D の開発を行った 2012 年当初、OS として Windows2003Server（32bit）を搭載した WEB サーバ上でのコーディングと動作テストを実施し、任意形式の三次元データファイルと、データ記録形式を記述したメタファイルの組をアップロードし、SQL サーバ上に座標値、図形、属性等を展開するアップロード動作を確認した。更に、ユーザがリクエストに添付したメタファイルに従い、別の形式の三次元データとして再構築して配信する動作を確認した。

Microsoft 社による Windows2003Server のサポートが 2015 年 7 月 15 日に終了したため、国総研が景観シミュレーションシステムのダウンロードサービス(1996-)や、まちづくり・コミュニケーション・システム(2001-)およびこの機能を活用した三次元アーカイブスの公開、VC-4D の開発(2012-)を行っていた電算室の公開サーバの OS を Windows2012Server（x64）に更新すると共に、SQL データベースも MSSQL7 から SQL2012 に更新した。この移植に際して、OS のセットアップを仮想ハードディスク（VHD）上とし、可搬性を高めた。この移植そのものは一般的な手順書に解説されているものであって特殊性はないが、古い処理系とデータを引き継ぎながら運用しているサーバ機能の新たな OS への移植に際して遭遇した問題点と解決手段は、今後の新たなサーバへの開発成果の導入や、既存機能とサービスを維持しながらの OS の更新に際して有益な経験となると考え、以下に記録しておく。

（２）仮想化

①概念と用語

Virtual は通常「仮想」と訳される。「空想」や「虚構」という意味はなく、異なるハードウェアや OS 環境下でも、同じようにソフトウェアが動作する可搬性を実現することを意味し、実体はどこかに必ず存在する。以下のような用語がよく用いられる。

１）パーティション

物理的なハードディスクの上に複数の領域を確保し、それぞれが別のドライブとして OS からアクセスできるようにする方法である。ハードディスクの先頭 512 バイトにある MBR(Master Boot Record)にパーティションの定義が記録されている。MBR で定義できるパーティションは４までである。

なお、MBR による方法では最大２TB までしか管理できないが、これに代わる GPT(GUID (Global Unique IDentifier)Partition Table)を導入したハードディスク装置では 128 のパーティションを作成できる。但し、Windows Vista 以後の 64 ビット OS が適合する。

a. システム・パーティション

保護のためエクスプローラでは表示されず、ドライブ名も無い。

ハード関連のブート処理に使用する。

コンピュータに一つだけあり、起動する複数の OS の選択リスト等が置かれる。

b. ブート・パーティション

セットアップした仮想マシン（OS）の数だけ作成される。

各 OS の起動に必要なシステム・ファイル群を格納する。

c. アクティブ・パーティション

選択され起動された OS のブート・パーティションである。

「ディスクの管理」（GUI）では、パーティションをアクティブとしてマークする。

「diskpart」（コマンドライン）では、セレクトした上で、アクティブにする。

```
>select volume 0
>active
```

d. プライマリ・パーティションと拡張パーティション

新しいパーティションを作成する際に選択する。

プライマリ・パーティションは、システム起動が可能なパーティションである。

作成すると、一つのドライブ名が割り当てられる。

拡張パーティションは、システム起動ができないパーティションである。

あるパーティションの領域を「拡張パーティション」として位置づけた上で、その内部に複数の論理ドライブを作成することができる。拡張パーティションを作成しただけではドライブとして認識されず、その内部に論理ドライブを作成すると、ドライブ名が割り当てられる。数の制限はないが、名前に使用できるアルファベット文字の数で制約される。

２）BIOS　Basic Input/Output System の略

1980 年代の PC においては、ディスクドライブの読み書きなどの基本処理をライブラリとして EPROM に格納したものを BIOS と呼び、プログラムから呼び出すと共に、起動時の OS の読込み（IPL）もこの中に記述していた。その後 OS が拡大し基本処理の大半が ROM に置かれなくなってからも BIOS の名称が存続しているが、実質的な内容はセルフテストや起動条件設定などを含むファームウェアとして、書き換え可能なフラッシュメモリに記憶しているハードが多い。更にこれに代わる UEFI(Unified Extensible Firmware Interface)を搭載したボードも製品化され、3TB のドライブからの起動を可能としている。

OS 自体のロードは、FDD、HDD（その各パーティション）、CD/DVD-ROM、Ethernet、更にはフラッシュメモリなどの USB デバイスからも選択的に行うことができ、起動を可とするデバイスと優先順位を BIOS の設定画面で変更することができる。

３）Boot Menu 起動デバイスの選択

OS をロードするためのプログラムをロードする方法を、靴のつまみに喩えて bootstrap（略して boot）と呼ばれる。通常の起動では BIOS で設定した優先順位に従い自動的にロードされるが、前回異常終了した場合や、起動デバイスとして選択されているデバイスが壊れている場合などに、オペレータに選択を求めるメニューがデバイスの一覧として表示される。メンテナンスのため起動時に指示表示されるファンクション・キー等を操作する

ことにより選択可能なドライブの一覧が表示され選択可能となる。例を示すと、

1. CD-ROM Drive

2. Removable Devices

3 +Hard Drive（内蔵 HDD からの起動）

4. MBA V7.7.5 Slot 0200　（MBA: Multiboot Agent を用いたネットワークからの起動）

４）Network Boot

BIOS による起動方法選択の一つで、内蔵ドライブではなくネットワークを介してマシンを起動する。ハードディスクが高価であった 1980 年代の端末起動技術を、セキュリティの高い端末起動やリモート操作でのＯＳ導入に活用したもの。

５）PXE　：(Preboot Execution Environment)

ネットワークからの起動のための規格。

Broadcom UNDI PXE-2.1 v7.7.5 を選択した場合、

CLIENT MAC ADDR:xxxxxx　GUID:yyyyyy

DHCP　と表示して、OS 取得先のホストを探しに行く。

(Broadcom は会社名, UNDI は Universal Network Device Interface)

６）Multi Boot　マルチブート

マシンにセットアップされている複数の OS から一つを選んで起動する。Windows NT では NTLDR、Vista では Windows Boot Manager が標準装備されていた。それぞれの OS は一つのパーティションを占有するため、OS 毎に用意する必要があった。

７）VHD（Virtual Hard Disk、仮想ハードディスク、②で解説）

Windows7 以降は、一つのバイナリ・ファイル（拡張子.vhd）としてハードディスクを仮想的に表現することにより、一つの物理的ドライブ（パーティション）の中に、容量が許す限りいくつでも新たな仮想ハードディスクを追加することができ、より多くの OS を選択的に起動することができるようになった。最大 2 TB。

ある OS 上で、一つの VHD ファイルをマウント（アタッチ）すると、あたかも一つのドライブを接続したように内部のファイルにアクセスすることができる。

８）Virtual PC

一つの 64 ビット OS の上のアプリケーションとして別の 32 ビット OS を起動し同時に実行させる。Windows7、2008R2 から導入され、Hyper-V への移行に伴い廃止された。

９）Hyper-V は、64 ビットのマルチブート環境を実現する。

Windows8、2012 から、VirtualPC に代わり導入されたハードウェアの上の階層（仮想環境）で複数の VHD 毎に OS をセットアップし、同時に起動し実行することができる。VHD は最大 64TB に拡張された（拡張子.vhdx）。ホストとなる管理 OS 自身も、この仮想環境の上に載っている点が VirtualPC とは異なる。

一つの OS はマウント（アタッチ）された複数の VHD にアクセスでき、複数の OS 間で VHD を共有することもできる。一方 Hyper-V でマシンを仮想化した場合、ディスク以外

の古いハード（シリアルポート、マウスポート、IDE 接続の FDD、CD/DVD 等）はエミュレーション化の影響で効率が下がる点がマルチブートの起動の場合とは異なる。

１０）Power Shell

MS-DOS の command.com→Windows NT の cmd.exe と類似した外観のコンソール画面を提供する実行形式(PowerShell.exe)。仮想マシンをブートメニューに追加する処理などを実行できる。

１１）Sysprep （sysprep.exe）

OS からマシン固有の ID である SID 等を除去して一般化した VHD を作成する。別マシンの複製に際し新たに固有の値を与え、ドメイン参加に際しての重複障害を回避する。

②仮想ハードディスク

ハードディスクの上に作成された一つのパーティション（ドライブ）は、通常はディスク管理ツールにより作成する。Windows では一つのパーティションに複数の OS を並存させることはできないため、新たなブートパーティション(または物理的なドライブ)を増設しなければ、一台のサーバ上に複数の OS を併存させ、移植前の OS と移植後の OS によるマルチブートの環境を実現することはできない。

これに対して、Windows7, Server2008 以降に利用可能となった仮想ハードディスク(VHD: Virtual Hard Disk)では、既存のドライブ上に大きなバイナリ・ファイルを作成し、その内部を読み書きすることにより、恰も独立した増設ドライブのように扱うことができる。このような仮想ハードディスクへの OS のセットアップ方法は、Windows7、2008Server 以降で可能となった。具体的には、インストール DVD-ROM をドライブに装着して一連の作業を開始し、必要なファイルが一時的なディレクトリに解凍された後、セットアップ実行形式を開始する前のタイミングで選択を行い、DVD が提供するコマンドライン・コンソールを用いて、DISKPART ユーティリティを起動し、これを用いて新たな VHD を作成し、セットアップ中の仮設的な OS にこれをアタッチして、そこに新たな OS を導入する。

仮想ハードディスクは、例えば VC-3M の開発・デバッグに用いた Windows マシン上での Android 携帯端末のエミュレータの内部の記憶装置(実際の携帯端末では SD カードに相当する、４－５参照）としても使用されており、また、Windows におけるハードディスクのバックアップもまた同様の形式のバイナリ・ファイルとして実現されている。

一つの仮想ハードディスクのファイルをデタッチして別のマシン（ディスク）にファイルとしてコピーし、そこで再びアタッチすることにより、物理的なドライブや記憶媒体をあるマシンから取り外して、別のマシンに装着するような結果を、コンソールで行う作業により得ることができる。

③実施例

具体的には、Windows7、Windows2008R2 以降の OS において、コンソールから diskpart コマンドで作業を行うか、ディスク管理ツールを用いて作業を行う。固定長と可変長のドライブを作成することができる。なお、Windows7 以降では、既存のパーティションのファ

イルを維持したまま、物理的なパーティション（ドライブ）のサイズを変更することもできるようになった。

Windows7 のセットアップディスクからブートし､ディスクの修復メニューでコマンドラインを選択すると､X ドライブ上に導入されたコマンド群が利用できるようになる。ここから diskpart コマンドで VHD を作成し、そこに OS をセットアップすることができる。

マルチブートを追加するためには、

a. diskpart ユーティリティにより VHD を作成し、そこに OS をセットアップする

b. bcdedit によりブートリストに追加する(BCD: Boot Configuration Data)

Windows7 マシンに、既存の OS を残したまま VHD を作成し、Windows2012Server を VHD 上に導入することができた。その手順を以下に例示する。

1）DVD－R から起動できるように BIOS を設定して、Windows2012 サーバセットアップ用 DVD から起動する。

2）セットアップ開始前の画面で、ディスクの修復を選択し、コマンド画面を開く。

3）DISKPART を起動する

4）空きディスクに VHD を作成する（CREATE コマンド)。

5）作成したディスクを選択する。

6）VDISK をアタッチする。

7）コマンドラインまで戻り、DVD>setup

setup は、コマンドラインとして Windows 2012 Server の上で動作しており、アタッチした VDISK（VHD）にアクセスすることができ、警告は出るものの、ここを選択して OS をセットアップすることができる。以後の手順は、通常のセットアップと同様である。

重要な点は、setup.exe が実行されている OS において、セットアップ先の VHD が見えていることが必要である点であり、このために、setup の直前に、セットアップに用いている OS のコマンドラインでアタッチ操作を行った後、再起動することなく、その OS の上で引き続きセットアップを完了させる。

④効果など

VHD は、これを使用していない OS から見ると、その OS にアタッチしない限り単なる一つのバイナリ・ファイルであり、その中にある構成要素を直接操作することはできない。よって、セキュリティ上の分離は高まっている。

VHD をファイルとして別システムに移動し、そこでアタッチすることにより、容易に移築することができる。

セットアップにおいて、VHD を作成し、そこに新たな OS を導入する方法は、ハードディスクを増設したり、新たなパーティションを作成したりできない場合でも実行できる。Windows 7 、2008R2 以降の OS がセットアップできれば、OS 導入後に、これらの機能を用いて既存のパーティションの内容を保持したまま、サイズだけを変更することもできる。

更に、過去に固定的なパーティションに導入した OS に関しても、これを VHD の上にコ

ピーした上で、そこから起動するように再構成することができる。

このような操作により、過去のテキスト、画像、プログラムおよびそれらにより構築された WEB サイトをアーカイブした上で新たな OS 上に移植する際に、古い OS を含めた古い環境全体を再実行可能な古い動体として凍結保存することが容易になった。言い換えると、一つの環境全体が記憶媒体の上での一つのファイル VHD として記録保存される。

（３）FTP

①概念と用語

FTP とは、File Transfer Protocol で、本処理系においては、IIS(Internet Information Service, Ver.8)によりサービスが提供される。HTTP プロトコルと並んで、古くから使用されてきたネットワーク上のファイル転送方法である。便利な機能ではあるが、近年では利便性よりもむしろ、セキュリティ上の脆弱性が問題となっており、不正なアクセス等を防止することが課題となっている。

FTPS とは、SSH File Transfer Protocol で、FTP において暗号化されていない状態において送信されるユーザ名とパスワードが盗聴されることを防ぐ方法である。

Explicit モードでは、ハンドシェイク完了後に暗号化された通信を行う。

Implicit モードでは、ハンドシェイクから暗号化された通信を行う。

FTPS サービスは、サーバ側で IIS により提供することができるが、クライアント側で Windows コマンドの FTP ではサービスを受けられない。FFFTP などにより初めて可能。UNIX 系の lftp をクライアントとして使用することは可能。また、FTPS サービスを実行するためには、サーバ側で SSL 証明書（有償）を購入する必要がある。ただし、SSL サイトと共通の証明書をワイルドカード証明書として利用することは可能。

通常は、クライアント側から PORT コマンドに引数としてクライアントのＩＰアドレスとポート番号を指定して送付し、データ転送を開始する。パッシブ接続では、クライアント側から PASV コマンドを送付して返されたサーバのＩＰアドレスとポート番号を用いてデータ転送を開始する。NAT や IP マスカレード処理に際して、PASV コマンドに対するレスポンスは変換対象とならないため、バグになりにくい。

NAT(Network Address Translation)は、ルータやゲートウェイが IP アドレスを変換する処理である。ルータは異なるネットワークの間を中継する。ゲートウェイはプロトコル変換を行う。たとえば無線 LAN では IPX/SPX プロトコルと TCP/IP プロトコルの間の変換を行う。

②ＩＩＳの設定

http、ftp プロトコルで Web コンテンツなどのファイル配信を行う IIS(Internet Information Service)サービスの導入は、Windows2012Sever においては「サーバの管理」画面から「機能と役割の追加」画面を開いて行う。IIS と、その下の HTTP、FTP 等を選択した上で実行すると、セットアップ DVD を使用することなくサービスを開始することができる。

FTP サイトを開設した上で、以下の設定を行う。

1）IP アドレスとドメインの制限

機能設定で「許可」を指定すれば、特定のＩＰアドレス以外からのアクセスはすべて許可される。本処理系では「拒否」を指定した上で、隣接するマシン（デバッグ用）の IP アドレスと、ファイアーウォールの IP アドレスを許可する。ファイアーウォールが複数あって負荷分散している場合には、すべて記述しておかないと、大きなファイルの転送に支障がある。

2）SSL 接続

FTPS を使用しない場合は、「許可」とするだけでよい。「必要」を選択すると、暗号化しない FTP 接続ができない。

3）承認規則

接続するユーザを指定する。「すべてのユーザ」に対して、読み取りと書き込み許可を与えるか否かを指定する。特定のユーザに対して特別な許可を与えたり、禁止したりすることができる。

4）ファイアーウォールのサポート

FTPS を使用する場合、またはファイアーウォールでパケットフィルターを処理していない場合に、パッシブ接続を受け入れるには、ファイアーウォールの外部 IP アドレスを設定する。このアドレスは、外部から見たときのファイアーウォールのアドレスである。

5）ユーザの分離：特に行っていない

6）認証

基本認証：有効（規定のドメインは空白のまま）

匿名認証：無効

（有効にする場合、アカウントとパスワードを設定する）

IIS への設定は、FTP サービスを再起動した時点で有効となる。

③アカウントの設定

FTP 接続する際に、クライアント側で入力する管理用のユーザ名とパスワードを決め、これをサーバのユーザとして登録する。FTP 接続専用に、管理権限の弱いアカウントとパスワードを一つ作成し、ファイル転送を行うディレクトリのファイル送受信（つまり読み取り・書き込み・変更）に関してのみ強い権限を与えるのが便利で安全である。

匿名接続を許可すると、任意のユーザ名＋パスワードで進入できるため、安全ではない。基本接続を使用する場合には、あらかじめ用意したアカウントとパスワードでのみアクセスが可能となる。但し、接続時の認証過程で、交信されるアカウントとパスワードが暗号化されないために、回線の状況によっては盗聴される危険性が存在する。このためには SSL 接続が勧奨されている。SSL 接続を行うためには、サーバの認証をプロバイダから有償で取得する必要があるが、自己認証を用いて無償で実現する方法は存在する。

④ディレクトリ・セキュリティの設定

FTP 接続する際に、クライアント側で入力するユーザ名の、サーバ上の各ディレクトリに対するアクセス権を設定する。FTP 接続でログインするユーザ名に対して、読み取り・書き込み・変更を許す権限を設定する。システム構築／再構築段階では、様々な要因により通信が成立しない状況が想定されるため、問題を単純化するためにまず、Everyone-Full Control 状態で動作を確認した後に、Everyone アカウントを削除して FTP 専用のアカウントだけを進入させるように変更する。

⑤ファイアーウォールの設定

Windows2012Server の場合、「セキュリティが強化されたファイアーウォール」ツールを使用する。起動方法は、[コントロールパネル]→[セキュリティ]→[ウィンドウファイアウォール]→[詳細設定]という経路、または[管理ツール]→[セキュリティが強化されたファイアーウォール]という経路を辿る。

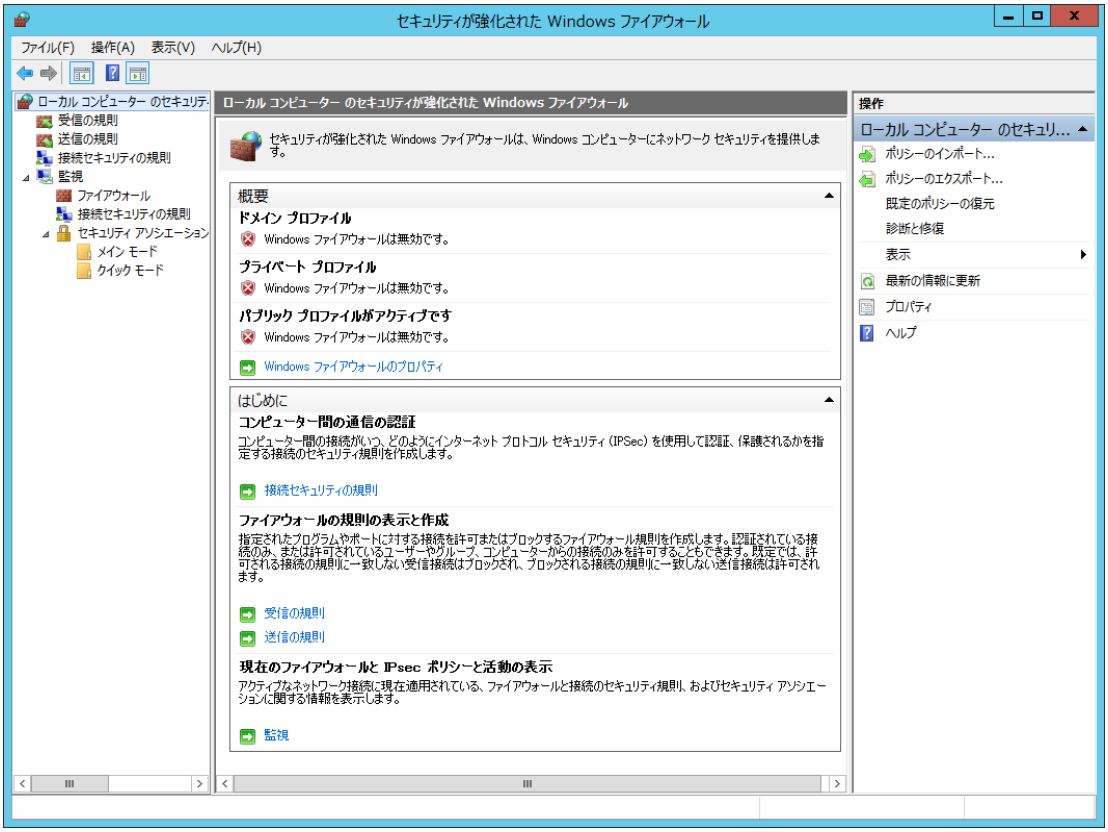


図 2-6-1 ローカルコンピュータのセキュリティ設定画面（Windows2012Server）

ファイアーウォールの有効・無効の切り替えは、右側のプロパティで行う。

FTP の受信に関する規則では、左ペインで「受信の規則」を選択した上で、中央ペインで FTP に関連する項目をダブルクリックまたは右クリック→プロパティまたは左クリックで選択状態にした上で、操作メニューのプルダウンからプロパティ選択し、編集画面をポ

ップアップする。

FTP 受信接続に関しては、[1]FTP サーバ（FTP トラフィック）、[2]FTP サーバセキュリティ（FTP SSL トラフィック）、[3]FTP サーバパッシブ（FTP パッシブトラフィック）を設定する。

FTP の送信に関する規則では、左ペインで「送信の規則」を選択したうえで、同様の手順で詳細設定画面を開く。

FTP 送信の規則に関しては、[1]FTP サーバ（FTP トラフィック送信）、[2]FTP サーバセキュリティ（FTP　SSL トラフィック）を設定する。

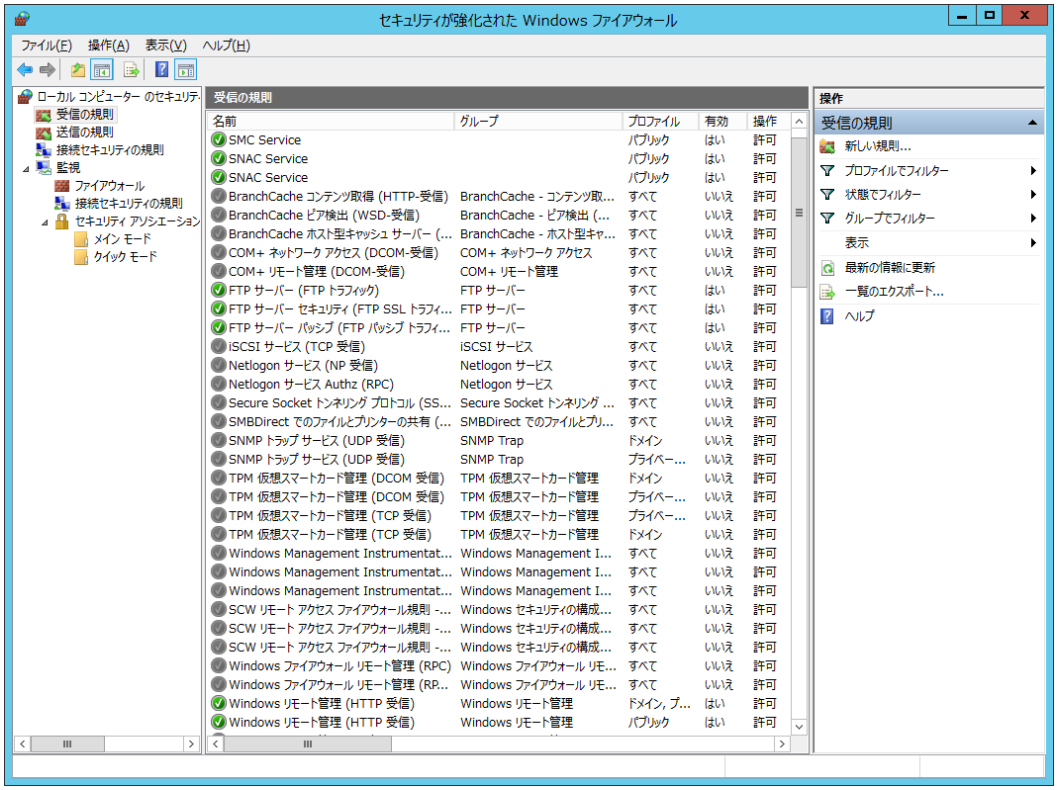


図 2-6-2 受信の規則の設定画面（Windows2012Server）

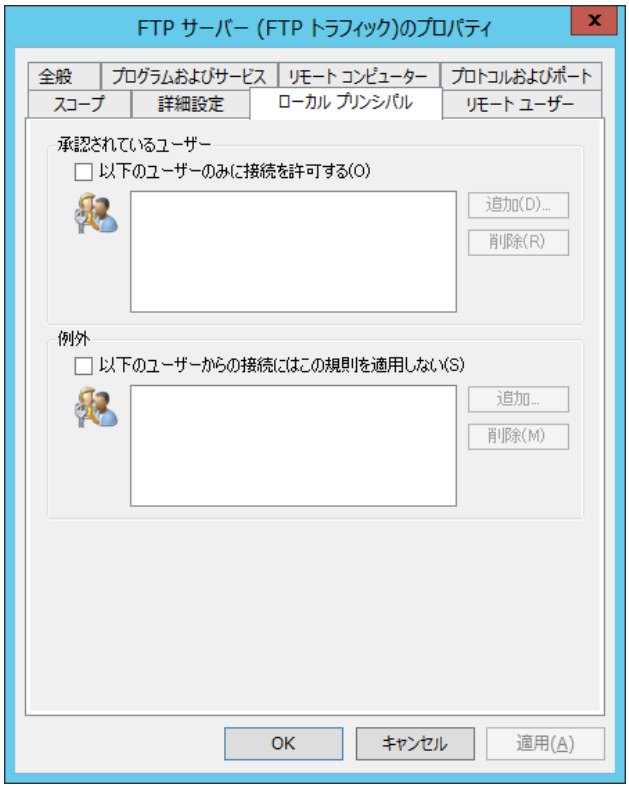


図 2-6-3 承認された FTP ユーザの設定（Windows2012Server）

ログインすることができるアカウントを制限することができる。

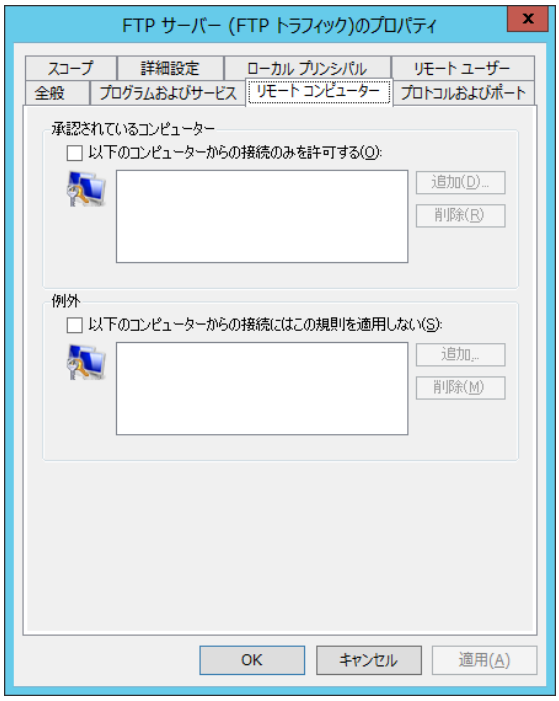


図 2-6-4 承認された FTP アクセス用コンピュータの設定（Windows2012Server）

このサーバにアクセスすることのできるコンピュータを制限することができる。

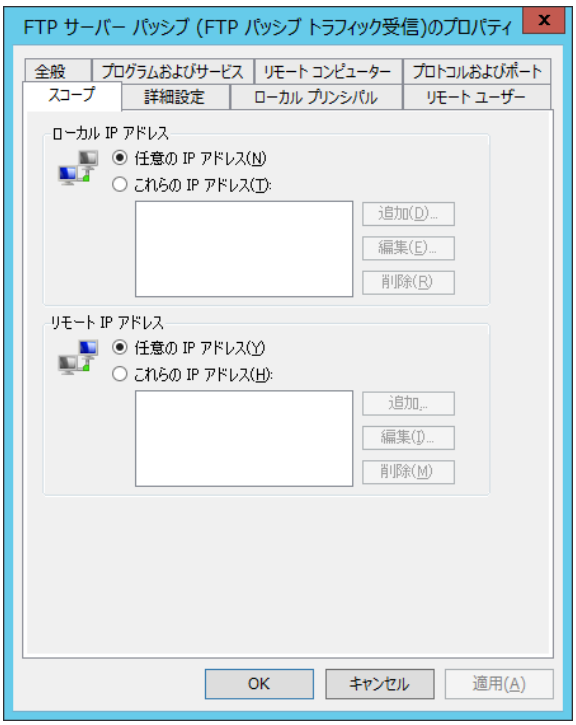


図 2-6-5 承認された FTP アクセス用 IP アドレスの設定（Windows2012Server）

このサーバ自身の IP アドレスと、このサーバにアクセスすることのできる IP アドレスを制限する。

なお、プロキシを介してアクセスする場合には、プロキシのアドレスをここに登録する必要があり、さらに複数のプロキシを用いて負荷分散している場合には、全てのプロキシを登録する必要がある。

ファイアーウォールには、ドメイン、プライベート、パブリックの 3 種類が用意されている。現在の接続方法は、実際の物理的接続にかかわらず、[管理ツール]→[ローカルセキュリティポリシー] →[セキュリティの設定] →[ネットワークリストマネージャポリシー](W7 の「ネットワークと共有センター」に相当)の設定画面により選択されている接続方法である。現在選択されている接続方法は、「セキュリティが強化された Windows ファイアーウォール」の監視画面で確認することができる。現在選択されている接続方法に対応したファイアーウォールの設定が、サーバの動作に影響する。この設定で選択を行っても、「識別されていないネットワーク：ネットワークに問題が発生したか識別可能な特性が不足していることが原因で、識別できないネットワークです」と表示されることがある。

設定対象となる FTP 受信に関しては、

[1]FTP サーバ（FTP トラフィック）

[2]FTP サーバセキュリティ（FTP　SSL トラフィック）

[3]FTP サーバパッシブ（FTP パッシブトラフィック）

の３種類がある。[1]FTP トラフィックだけ許可され、[3]FTP サーバパッシブトラフィックが禁止されている状態では、FTP 接続はできるが、LS コマンド(ファイル一覧表示の要求)などで応答が来ない。

⑥サーバ側のセキュリティに関する小結

Windows2012Server への移行に際して、各種設定内容の基本は従前 OS と変わらないが、設定に使用する操作画面や、設定結果を保存するファイルなどが大きく変化している。

FTP 接続に関しては、セキュリティ確保のために以下の制約条件設定が可能である。

1) 接続するクライアント・マシンの IP アドレス
   - IIS のサーバ全般のプロパティ設定における「IP アドレスとドメインの制限」の機能設定で「拒否」を指定しておき、特定の IP アドレスだけを例外的に許可することにより、アクセスできるマシンを制限することができる。この設定は、配下の FTP サイトの各ディレクトリに関する個別の設定のデフォルト値となる。
   - 必要であれば、個別の FTP サイトに許可する IP アドレスを増補することができる。
   - ファイアーウォールで、FTP 着信のスコープとして、クライアントの IP アドレスを制限することができる。
2) 接続するアカウント
   - IIS のサーバ全般のプロパティ設定で、「アカウントの認証規則」によりユーザを制限することができる。
   - ファイアーウォールの FTP 着信で、リモートユーザを制限することができる。
   - Windows のディレクトリまたはファイル毎のセキュリティ設定で、ユーザを制限することができる。
3) その他
   - 最大ファイルサイズを制限することができる。
   - アクセスすることができるファイルの種類（拡張子）を制限することができる。

⑦FFFTP

サーバの管理に使用するクライアント側マシンで FTP の接続確認や、ディレクトリ一覧を確認するだけであれば、コマンドラインからの FTP コマンドを使用するのが便利である。多くのファイルを一括して転送する場合には、エクスプローラや FFFTP を使用するのが便利である。エクスプローラでは、大量のファイルを送付中に途中でエラーが生じた場合に、以後の動作がキャンセルされてしまうため、最初からやり直す必要が生じる。

FFFTP においては、メニューの[接続]以下で、サーバ毎、ないし同一サーバ上のプロジェクト／サイト毎に接続の単位を作成する。ファイルの送受信に使用するクライアント（ミラーサーバ）との間にファイアーウォール（プロキシ・サーバ）が存在する場合には、メニューの[オプション]以下でファイアーウォールの設定を行った上で、個別の接続単位に関してファイアーウォールを使用するように設定する。

[例]プロキシ・サーバのクライアント側からの IP アドレスを<ip1>、プロキシ・サーバから

見た WEB サーバの IP アドレスを<ip2>とすると、ftp コマンドでこのサーバにアクセスする場合には、cmd.exe、PowerShell.exe 等のコマンドラインツールで

```
open <ip1>
user ユーザ名@<ip2>
pass WEB サーバにおけるこのユーザのパスワード
```

とタイプする。同じ接続条件を FFFTP に指示するためには、図 2-6-6～8 に示した操作画面での設定を行う。

⑧まとめ

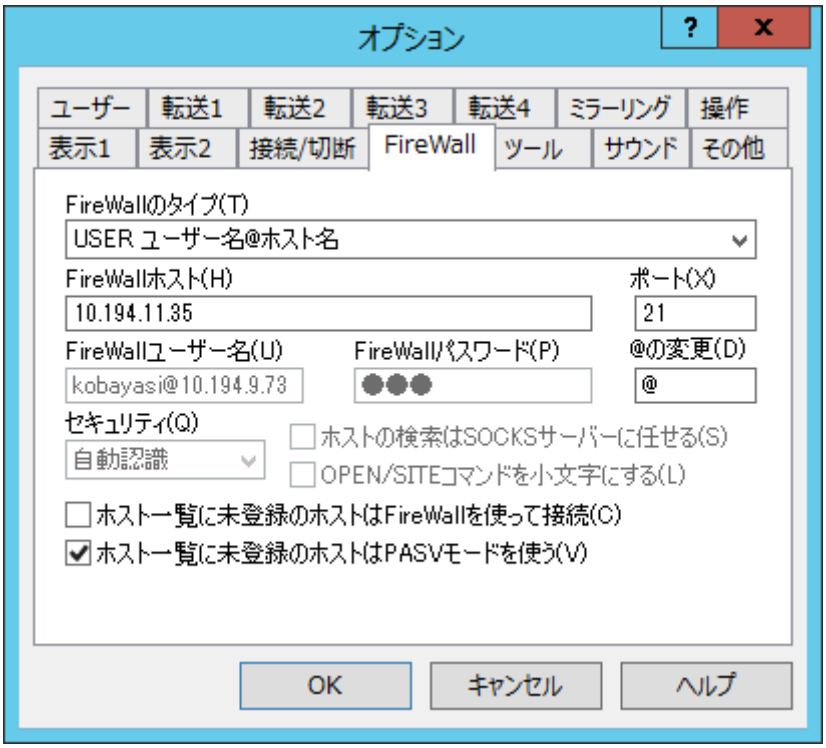


図 2-6-6 ファイアーウォールの設定（FFFTP）

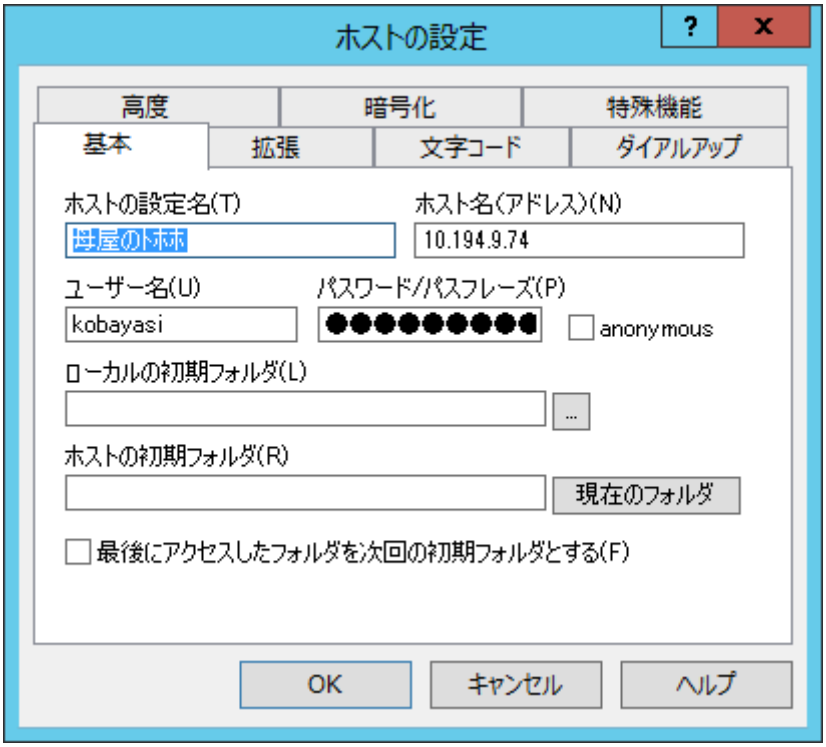


図 2-6-7 アクセス先サーバの設定（FFFTP）

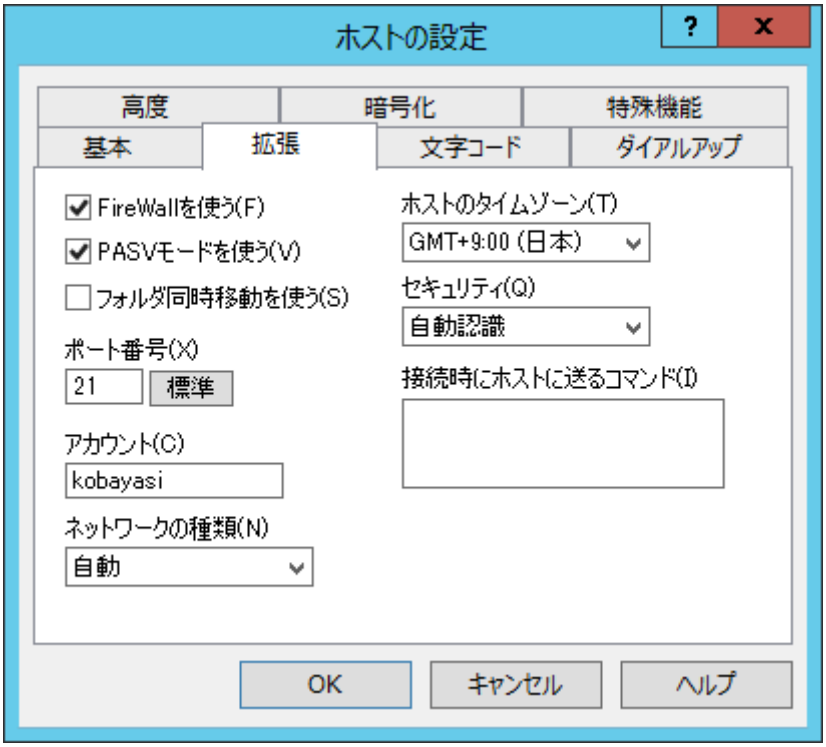


図 2-6-8 PASV モードの設定（Windows2012Server）

（４）データベース（SQL サーバ）

①データベースの歴史と展望

現在「データベース」と呼ばれるものは、単なるデータではなく、動的に維持更新するようなシステムであって、メモリに制約されずに大量のデータをファイルで管理するものである。基本的には突然の電源遮断やシステム・ダウン、あるいはバグ障害が生じても、複数のテーブルが一体に連携したデータの部分的変更により整合性が失われないようなアルゴリズム（トランザクション、コミット、ロールバック）が用意されている。

景観シミュレーション・システム（1993～）において使用している景観データベースは、主に検索機能の実現を目的としてゼロベースで開発したプログラムである（巻末文献 11、建築研究資料 92)。入力用のエディタを用いて三次元データ本体に加えて、様々な検索のための補足情報を登録し、入力されたデータベースは、COM.TXT というテキストファイルに保管した。データベースの入力は、editor.exe という入力用の Windows 実行形式を用いて行い、検索は三種類の実行形式（yuu.exe:優良景観事例、zai.exe:景観材料、kou.exe:景観構成要素）を用いて検索した。データベースはこの段階では小規模であったため、editor.exe の動作中は、全てメモリ上に展開され、入力が終了した時点で、外部ファイルを更新・保存する。ファイル保存中にシステム・ダウンが生じた場合に過去の累積データが失われないように、バックアップを自動的に作成するようにした。よって、損失はメモリ上に全ロードされたデータに対する更新部分のみである。

まちづくり・コミュニケーション・システムの開発(2001～)に際しては、このデータベースを WEB サーバ上で運用するシステムを実現すると共に、データベースの構成（スキーマ）をプログラムで固定的に扱うのではなく、DEF.CSV という外部ファイルで定義することにより、様々な形式のデータを扱うようにした（巻末文献 21、国総研資料 134)。この処理系においては、MSDE という無償で再配布することができる Microsoft 社が提供するデータベースエンジンを使用し、上記の補足情報（属性情報）を管理すると共に、三次元データ自体は、固定的なファイルとして、サーバ上のディレクトリに置いて管理した。このシステムにおいては、三次元形状を記述したデータは、アップロードされたファイルのまま蓄積し、その内部を図形的に解析してテーブルに展開するような処理は行っていない。

２－７、３－６で解説する VC-4D(三次元データ保管庫、2012～)においては、三次元データ自体も、頂点座標、線、面、立体、色彩等の要素に分解する。これにより、異なるデータ形式への変換を可能にしている。

データベースエンジンとして使用した SQL サーバにおいては、複数の異なる形式のデータの集まりを一つのレコードとし、任意の数のレコードの集まりをテーブルとして管理している。C 言語風に表現すれば、構造体の配列であり、構造体の定義がスキーマである。

このテーブルの構造は、Lotus, Excel のような表計算ソフトにおける表と類似しているが、列の数がスキーマで予め定義された数に限定されており、また集計式等を項目の中に埋め込むことはできない。

レコードの構成要素は、データそのものであっても、別のテーブルの要素への参照であっても良い（リレーショナルデータベース)。例えば、ある物件のテーブルに所在地都道府県の項目がある時に、都道府県のテーブルが別途用意してあれば、全ての物件に関して文字列で所在地を記入する代わりに、都道府県のテーブルの ID を指示するだけで済ませることにより、メモリやファイルサイズを節約することができる。

このような一連のテーブルを束ねたものを「データベース」として、一つのファイル(MSSQL の場合、拡張子.mdf)として動的に管理している。ファイルを元にデータベースエンジンを使用することにより、停電による全データ消失を防ぐことができる。

但し、例えばある頂点座標を定義した上で、その ID を用いて面を定義するような処理を行っている途中でエラーが生じた場合には、例えば参照されない頂点座標や、参照先のない面のデータが生成する恐れがある。そこで、このような一連の処理を TRANSACTION として束ね、一連の処理が終了するまえに不測の中断が生じた場合には、外部ファイルとして保存されるデータベースに反映しないように保留することができる。TRANSACTION が終了した段階で、問題が無ければ COMMIT することにより結果を確定することができ、また問題があれば ROLLBACK することにより処理前の状態に戻すことができる。これにより、たとえ 1 件のデータ投稿が不成立に終わっても、データベース全体の一貫性が失われないように管理することができる。

②利用可能なデータベースエンジン

MSSQL で、無償で再配布することのできるデータベースエンジンは、2001 年頃の MSDE から、SQL EXPRESS と名称が変更され、2005, 2008, 2012 等のバージョンに改良されている。

プログラム開発途上にあっては、データベースの内容を例えば表形式で表示し訂正したり、各種のサンプルデータを手入力したりする道具があると便利である。MSDE の頃には、このような道具として、Enterprise Manager というアプリケーションが提供されていた。その後、この道具は、Development Studio という名称になり、開発環境と一体で提供されるようになった。

開発環境 VS2005 を搭載した Windows8.1 にセットアップされている、上記の条件に該当する SQL データベースは表 2-6-0 の通りである。

表 2-6-1　無償で利用可能な MSSQL サーバ(2015 年時点)

2005
2005 Mobile [JPN] Developer Tools
2008 R2 管理オブジェクト
2008 セットアップサポートファイル
2012 (64 ビット)
2012 Express LocalDB
2012 Native Client

データベースへの操作は、sqlcmd.exe(isql.exe, osql.exe の後継)によって SQL 文をコマンドラインで実行することができる。コマンドライン(cmd.exe または powershell.exe)の画面から sqlcmd とタイプすることにより起動できる。sqlcmd は ODBC インターフェースを

用いてデータベースエンジンにアクセスするため、ODBC に使用するデータベースが登録されている必要がある。

Windows2012 Server (standard edition)をサーバの OS として、この上に、現時点で最新の無償 SQL サーバをダウンロードし、セットアップをテストした結果は、以下の通りである。データファイルとして同じデータベース(拡張子.mdf)を利用するために、使用できるエンジン、接続するためのプロバイダ、アクセス方法などが選択可能である。過去 20 年の流れで見ると、多様な設定やアクセス方法が可能となった一方、初期導入には手間と時間がかかるようになった。

１）2012 Express LocalDB (ローカル DB)

LocalDB は、従来の MSSQL サーバがサービスとして動作するのとは異なり、実行形式（インスタンス）の形で動作する。セットアップした後に、コマンドラインで動作する sqllocaldb.exe を起動し、コマンドを入力することにより、様々な設定を行う。

＞create 名称 : 名称のインスタンスを作成する。デフォルトで、v.11 というインスタンスが一つ起動しており、複数のインスタンスを作成することができる。

＞start 名称 : 名称で指定したインスタンスを開始する。

＞stop 名称 : 名称で指定したインスタンスを終了する。

＞delete 名称 : 名称で指定したインスタンスを削除する。

＞info : 現在作成されているインスタンスの一覧を表示する

＞info インスタンス名称 : 名称で指定したインスタンスの状態を表示する

このインスタンスが開始され実行中である場合に、リストの末尾に「インスタンス　パイプ名」として表示される「名前付きパイプ」の名前がアクセスに必要である。このパイプ名称は、例えば

「np:¥¥.¥pipe¥LOCALDB#F3A1FC0B¥tsql¥query」

といった比較的長い名称である。

開始している実行されているインスタンスについて、コマンドラインツール sqlcmd.exe を起動し、

＞sqlcmd –S パイプ名称 –E

により、ログインが成立すると、１＞のプロンプトでコマンドラインから SQL 文を受け付け実行することができるようになり、以後データベースの作成や検索・更新など各種の操作を行うことができる。

LocalDB のセットアップは３６MB 程度のサイズであり、ダウンロードは短時間で終わり、セットアップ全体は数分以内で完了する。

なお、sqlcmd は、ODBC ドライバを使用しているため、LocalDB にアクセスするためには、事前の登録が必要である。「コントロールパネル」から（[全てのコントロールパネル項目］－）［管理ツール］－［データソース(ODBC)］の順に選択し、LocalDB を［追加］登録する。

２）SQL2012 とマネジメント・スタジオ

開発・デバッグ用としてデータベースの内容の確認や、バックアップ条件等各種の設定を試行錯誤的に行うための Management Studio の機能を含む SQL 2012 のダウンロード(http プロトコル)とセットアップには、相当の時間（2015 年 7 月時点、筆者の作業環境では約半日）を要した。そのほとんどは、コマンドで指示した処理の終了待ちである。

マネジメント・スタジオ(Management Studio)は Enterprise Manager の後継で、グラフィカルな画面でデータベースの管理やデータの操作を行うことができる。これが使用する.NET Framework3.5 は、共通言語基盤(CLI, common language Infra. JIS, ISO 等として標準化されている）として、各種言語のソースコードからコンパイルした結果を中間言語として処理する。CLI は、基本クラスライブラリ（BCL）と、仮想実行システム（VES）から成っている。これにより、C#や VB.NET 等のソースコードから共通の中間言語（CIL）を生成し、これを更に共通のコンパイラ(JIT、just in-time コンパイラ)により、実行時にコンパイルすることにより各処理系のネイティブコードにコンパイルして実行する、二段階構成のコンパイラシステムを実現している。

このため、Management Studio を導入するためには、それが依存する.NET Framework の先行導入が要求されており、条件が満たされないと、SQL サーバのセットアップは失敗終了となる。なお、場合によっては、SQL2012 のセットアップの途中で NET Framework のインストールには、「Windows の機能(Windows features)」のダイアログで、「この機能をインストールします。」という選択肢が示される。

.NET Framework は、コントロールパネルの「プログラムと機能」の中に存在し、選択してアンインストールすることができる。また、左側に表示されている「Windows の機能の有効化または無効化」をクリックすると、インストールされている各機能について

有効／無効をセットするためのチェックボックス群が縦に表示される。

＊Windows2012Server の場合には、上記の操作を行おうとすると、「サーバマネージャ」が起動される。左欄でローカルサーバを選択して右画面を下の方にスクロールすると、.NET Framework 4.5 および .NET Framework 4.5 Features(機械翻訳では、Features は通常「機能」と訳される)の項目がある。これは表示のみで、有効／無効を切り替えるインターフェースは見られない。セットアップは失敗と表示されるものの、より上位の.NET Framework 4.5 が動作しているためか、サーバマネージャなどの設定画面は開くことができ、セットアップしたサーバの設定を行うことができる。但し、たとえこれが有効であっても、より古いバージョンの.NET Framework3.5 がセットアップされていなければ、Management Studio をセットアップすることができない。

.NET Framework 3.5 は OS 導入時に自動的にセットアップされていないため、新たに導入する必要がある。このためには、サーバマネージャのダッシュボードで、役割と機能の追加を選択し、役割と機能の追加ウィザードを開く。「機能選択」画面で、NET Framework 3.5 Features の左のチェックボックスにチェックを入れる。サーバが Windows Update（インターネット）に接続できない環境（バリアセグメント等）にある場合、セットアップがプログレスインジケータの表示４０％程度まで進行した所でエラー終了する。この場合、Windows2012Server（セットアップディスク）を DVD ドライブにセットし、例示に従って、E:¥sources¥install.wim を代替サーバとして指定する（タイプ入力）。Install.wim とは、Microsoft Windows Imaging Format(WIM)形式による OS のインストール元である。

インストール媒体（DVD-ROM）を使用する場合には、次の設定コマンドをコマンドラインから管理者権限で実行し、イメージの展開を行う必要がある。

```
Dism /online /enable-feature /featurename:NetFx3 /All /Source:E:¥sources¥sxs /LimitAccess
```

ここで、Dism とは、展開イメージのサービスと管理を行うコマンドである。
/online は、Windows の機能に関する
/enable-Feature は、削除された Windows の機能を復元する
/featurename: は機能の名称　NetFx3 は、NET Framework 3

「機能を有効にしています」の次の行にコマンドラインのプログレスインジケータが表示され、66.2%で長く止まるが、100%まで到達する。なお、ソースの指定が誤っていても、途中まで進行して、エラーで終了する。この処理に約 1 時間。失敗した場合に参照可能な DISM ログファイルは c:¥Windows¥Logs¥DISM¥dism.log にある。

恐らく、Windows Update をインターネットで許可してあれば、この辺の処理は自動的に行われるのであろう。保護された環境下での作業を行う方法は用意されているが容易ではない。成功すると、「操作は正常に終了しました。」と表示。

これにより、セットアップが前に進む。この作業には 1 時間程度を要する。この作業の間、システムはネットワークから受信を断続的に行っているが、CPU 時間はあまり消費していない（１０%未満）。セットアップや試行錯誤の終了後、無効化ないしアンインストールしておいた方が安全である。

SQL サーバのセットアップに際して、既定のインスタンス（名称：MSSQLSERVER）か、あるいは名前付きインスタンス（名称：任意）かを選択するダイアログが表示される。（この点は、SQL Server Express は、常に名前付きインスタンスでインストールされるのと異なる。）名前付きインスタンスで名前を指定しなかった場合には、名前の無い名前付きインスタンスとしてインストールされる。この場合、デフォルトの SQLExpress が自動的に、インスタンス名となる。名前付きの場合には、「サーバ¥インスタンス名」でアクセスする必要がある（例えば、「.¥SQLExpress」）。

インスタンスとはメモリ上にロードされた実行形式のことである。同一のプログラムの複数個がメモリ上の別の場所にロードされた場合には、異なるインスタンスとなる。

認証モードについて、Windows 認証か、混合認証かを選択できるダイアログが表示される。Windows 認証の場合にはパスワードはないが、混合認証の場合には s a にパスワード設定が求められる。以前のバージョンとは異なり、SQL Server 2012 においては、パスワード無し、あるいは単純なパスワードはエラーとなる。覚えやすい複雑なパスワードを設定するのが便利である。ここで「複雑」とは、アルファベットと数字と記号が含まれていることを示す。よって、「n=3」は複雑なパスワードとして評価される。

③ コマンドラインを用いた、ＳＱＬサーバの検証

データベース・システムがセットアップされた後に、簡単にコマンドラインから接続して検証することができる。

`sqlcmd –E` で認証プロセスなしにアクセスできる。

`sqlcmd –U sa` でユーザを sa とすると、次行のプロンプトでパスワードを入力し、アク

セスできる。パスワードが設定されていなければ、改行するだけでよい。

データベース名を指定する場合には、-S でデータベース名を指定する

sqlcmd –S （SQL サーバ名） -E

SQL サーバ名は、セットアップ時に既定のインスタンスとして指定した場合には、データベース名のみでよく、またデータベース名自体を省略してもよい。sqlcmd(改行)と入力するだけで、既定のデータベースが起動する。

一般には、マシン名￥SQL サーバ名 の形式で SQL サーバ名を指定する。

起動が成功すると、1> のように SQL の入力を求めるプロンプトが表示される。

更に、この前にプレフィクスを付けることにより、名前付きパイプ、IP などの接続方法を指定することができる。

ローカルマシンの場合には、マシン名の部分に当該マシン名を入力する代わりに、”.”、“(loal)”、”localhost”等の一般名とすることができる。

sqlcmd -? で、各種のパラメータ設定の一覧を表示することができる。

```
1 >select @@version
2 >go
```

で、システムのバージョンを表示することができる。

```
1>exit
```

で終了する。

このサーバ上にデータベースを一つ構築するためには、

```
1>CREATE DATABASE 名称
2>GO
```

と入力する。データベースの名称や場所等を引数で詳細に指定することができるが、省略することもでき、その場合には同一の名称のファイルが標準の場所に作成される。

このデータベースを選択して、内容を表示し変更するためには、

```
1>USE 名称
2>GO
```

と入力する。

作成されているデータベースを削除するためには、

```
1>DROP DATABASE 名称
2>GO
```

と入力する。このデータベースが指定されている状態ではエラーとなる。

現在選択されているデータベースのテーブルを全表示するためには、

```
1>SELECT * FROM テーブル名
2>GO
```

と入力する。

システムデータベースとして最初から存在している master と同一名称のデータベース

を作成することはできない。また、この master データベースを削除することはできない。

SQL サーバに接続する際に、データベース名を指定しなかった場合には、接続が確立した段階で、master データベースが選択されている。新たなデータベースを作成した場合には、対応する保存ファイルが作成されると共に、これらの情報が master データベースに登録されている。

```
1>SELECT name FROM sys.databases
2>GO
```

というコマンドにより、現在のデータベースの一覧を表示することができる。この一覧の中には、master 自身も含まれている。

SQL サーバへの接続のプロトコルには、共有メモリ、TCP/IP、名前付きパイプ、VIA があり、SQL Server Configuration Manager によってそれぞれの有効／無効を設定することができる。

一方、接続するクライアントの側では、SQL サーバ名にプレフィクスを付けることにより、プロトコルを指定することができる。

1) 名前付きパイプ

np:¥¥<コンピュータ名>¥<パイプ名>

2) TCP/IP

tcp:<コンピュータ名>,<ポート番号>

3) 共有メモリ（主にトラブルシューティングに用いる、ローカルな接続）

lpc:<コンピュータ名>¥<インスタンス名>

4) VIA プロトコル（仮想インターフェースアダプタ）

ネットワークインターフェースカード番号（NIC 番号）とポート番号を指定する

via:<SQL サーバ名> [¥<インスタンス名>],<NIC 番号>:<ポート番号>

NIC 番号が省略された場合には、1433 番がデフォルト番号となる。

複数のインスタンスがある場合に推奨されない。

```
現在の接続方法を確認するためには、
SELECT net_transport FROM sys.dm_exec_connections WHERE session_id=@SPID;
```

基本的には、コマンドラインだけで複雑な操作を実現することができ、またそのような手順をスクリプトとして保存しておき、異なるデータベースに対して繰り返して適用するようなことも可能であるが、途中でエラーが生じた時の処理結果のログやメッセージを作成し、中断終了の分岐処理などを記述することは困難であるため、より高度なスクリプト言語を用いて処理を記述するのが便利である。

データベース作成処理の内容は、プロジェクト毎のデータベースの構築と、初期状態（空）のテーブル群の作成である。

まちづくり・コミュニケーション・システムの場合にも、VBScript を用いて、データベースを構築する処理を記述しており、その過程での各処理の成否については、ログファイ

ルを出力しているので、結果を確認するのに便利である。

バックアップ方法等は、SQL サーバにより異なるため、以前の Enterprise Manager や、上記の Management Studio などの専用の設定ツールを用いて作業を行うのが便利である。これらのツールを用いて、データベースの各テーブルの内容を閲覧し修正することもできるが、大量の一括処理を行うことは実用的ではない。

このようなツールをセットアップせずとも、データベースエンジンと、実用システムの一部として作成してある専用のデータベース構築・削除ツールがあればシステムを構築し運用し廃止する作業は行うことができる。しかしながら、システムを運用する中で、悪意のある攻撃、システムのバグによる不適切なデータの生成などの想定外の状態が生じる場合があり、その検査と緊急対応的な作業ツールとしては便利である。

一方、このようなツールが存在することは、SE 等の立場の人達がまちづくりに関する意見（テキスト）を改ざんしたり、あるいは一部の意見だけを削除したり、といった不正を容易に行えるようにするような負の側面も有している。セットアップや試行錯誤の終了後、無効化ないしアンインストールしておいた方が安全である。

まちづくりコミュニケーション・システム(2001)においては、プログラム的要素を含む意見投稿などのハッカー的な攻撃に関しては、SE や管理者（いわば事務方）による主観的な管理ではなく、審査システム（いわば外部査読者）を通じて不合格判定とし、公開される段階に移行しない仕組とした点に特徴がある。技術面を支えるべきシステム管理者等による主観的な評価だけで、反対意見を封じこめるようなデータ介入は行えない事が望ましい。

④地区別 WEB サイトのためのＳＱＬサーバの設定と検証

１）アクセス文字列

従来の VB スクリプトにおける古いスタイルのデータベースアクセス：

```
SET DB=CreateObject(“ADODB.Connection”)
Cstrg=”Provider=MSDASQL;DRIVER:sql server; SERVER=(local);
DATABASE=”&dbname&”; UID=”&dbuser”;PWD=”&dbpass
DB.Open Cstrg
```

修正後のスクリプトにおけるデータベースアクセス：

```
SET DB=CreateObject(“ADODB Connection”);
Cstrg=”Driver={SQL Server}; server=; uid=”&dbuser&”; pwd=”&dbpass&”;
DB.Open Cstrg
```

なお、古いスタイルは、勧奨されていないが、接続はまだ可能である。

ここで、ADODB.Connection とは、Active Data Object を接続プログラムの組込み方法として用いることを示す。 Provider は、接続のために使用するプログラムであり、MSDASQL.dll を使用している。Driver は、ODBC(Open Data Base Connection)の中で使用するドライバである。どのドライバが利用可能であるかは、ODBC 設定ツールである odbcad32 つまり OpenDataBaseConnectionAdministrator32bit 形式を用いて、確認し追

加することができる。コマンドライン窓から odbcad32 とタイプすることにより起動する。図の例では、SQL Native Client、SQL Server、SQL Server Native Client 11.0 の 3 種類が、ドライバとしてセットアップされていることがわかる。Server は、アクセスするサーバであり、記述方法はデータベースのセットアップ方法に依存している。

(local)は、古いスタイルで、コンピュータ名¥サーバ名が、完全な記述である。ローカルコンピュータの場合には、.¥サーバ名 の形で省略することができる。

サーバ名が省略されると、Native Client サーバへのアクセスが試みられ、失敗するとタイムアウトする。

利用可能なサーバは、Microsoft SQL Server Management Studio を起動して、接続する際に参照を選択すると、一覧として見ることができる。

なお、ODBC データソースアドミニストレータを用いて、DSN（データソース名）を定義し、”DSN=・・・”という接続文字列を用いて SQL サーバにログインする方法も可能であるが、セットアップ作業が一工程増えることとなるため、今回は現実的・実用的観点からこのような理念的対応を棄却した。何故ならば、そのような互換性は過去に遡ったプログラムの修正により将来にわたるプログラムの互換性を保証する、というアプローチであり、現実的には過去に遡る必要はなく、また将来 SQL サーバが供給され続ける保証もないからである。

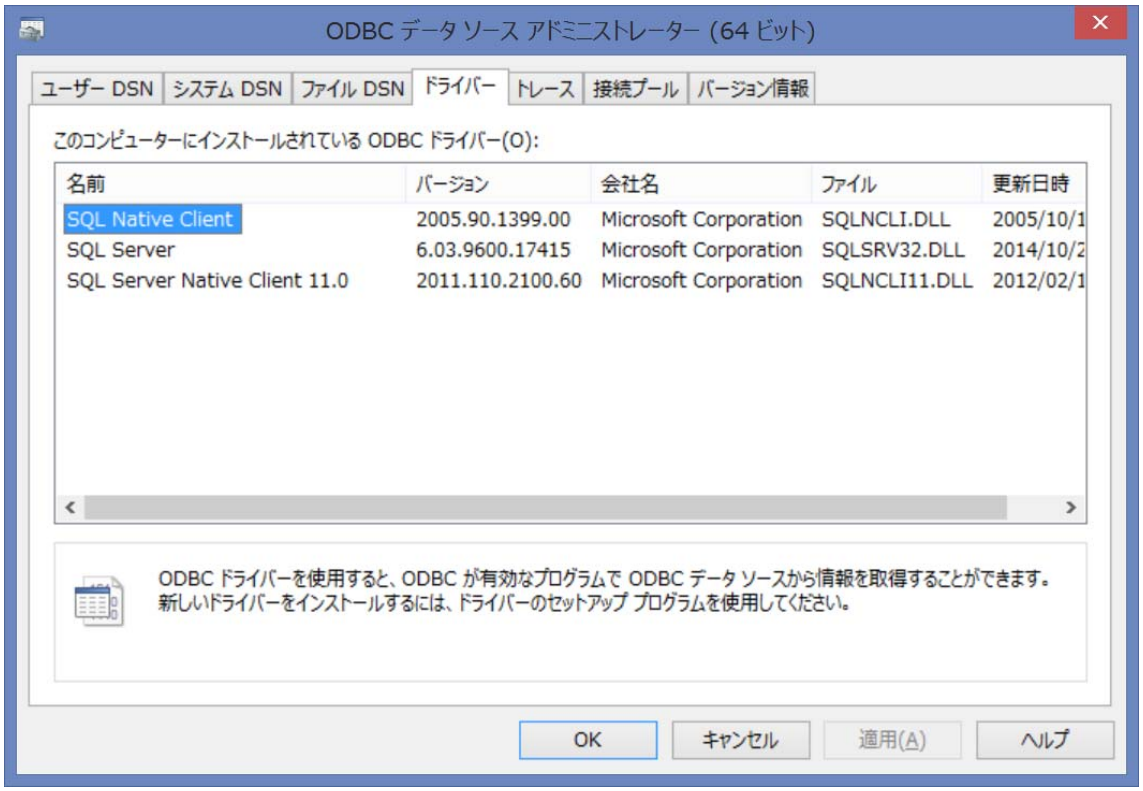


図 2-6-9 データベースドライバの選択と設定（Windows2012Server）

２）asp のエラーメッセージの表示

asp は、サーバ側のプログラムを記述する方法であり、本処理系においては VB スクリプトを用いている。html 形式のコンテンツの中に、<% ・・・%>で囲んだ中に、BASIC 言語による記述を行う。

IIS の設定で、奥尻サイトの asp のプロパティを開き、デバッグのプロパティの下にある、ブラウザへのエラー送信を、false から true に変更する。

データベース名は、ダイアログから入力されているものを使用している。新たなデータベースを作成する際には、この名称のデータベースはまだ存在していない筈である。その場合であっても、DB.Open は実行され、master データベースが選択された状態になる。master には、ユーザが作成(create)したデータベースが登録されている。

スクリプトの実行がコンパイル失敗により E500 となる場合（ブラウザ側での表示)、引用符”の不適切な使用が原因である場合が、経験的には多い。

コンパイルが成功し、スクリプトが実行された後に発生したエラーの個数は、DB.Errors.Number メンバ変数によって知ることができる。但し、Open に際しては、データベース名が指定されていないか、指定された名称のデータベースが無い場合であって、master データベースが代わりに開かれたような場合の警告、および言語を日本語に変更した警告も DB.Errors.Number としてカウントされる。よって、警告を除いた実害のあるエラー数を数えるためには、全てのエラーにアクセスして、内容を調べる必要がある。

リスト 2-6-1 クラシック asp による SQL コマンドエラーの表示関数

```
Function sdb_error(DB)
                dim count
                dim nerr
                count = 0
                Response.Write "【エラーチェック開始】<br>"
        For Each errLoop In DB.Errors
            If 0 <> errLoop.Number Then
              count = count + 1
            End If
            strError = "Error #" & errLoop.Number & "<br>" & _
                "     " & errLoop.Description & "<br>" & _
                "     (Source:" & errLoop.Source & ")" & "<br>" & _
                "     (SQL State:" & errLoop.SQLState & ")" & "<br>" & _
                "     (NativeError:" & errLoop.NativeError & ")" & "<br>"
            If errLoop.HelpFile = "" Then
                strError = strError & _
                    "     No Help file available" & _
                    "<br><br>"
            Else
                strError = strError & _
                    "     (HelpFile:" & errLoop.HelpFile & ")" & "<br>" & _
                    "     (HelpContext:" & errLoop.HelpContext & ")" & _
                    "<br><br>"
            End If

            Response.Write("<p>" & strError & "</p>")
        Next
        sdb_error = count
        DB.Errors.Clear
        Response.Write "【エラーチェック終了】<br>"
End Function
```

例えばリスト 2-6-1 に示した sdb_error 関数によってエラーの原因を表示させることができる。このようなエラー表示機能は、サーバの更新に伴う障害の原因を特定するツールとしては有効であるが、設定が終了した後には削除するか不活性化しておくことにより、セキュリティ上の弱点とならないように対策しておく必要がある。

⑤ 掲示板機能のためのテーブルの構築

個々の地区別のコミュニケーション・サイトのための SQL データベース（空）を作成した上で、その中に事業別の初期状態の各種テーブルを作成する。テーブル構成はスキーマである def.csv ファイルによる定義に従う。この作業はコマンドラインや Management Studio などの管理ツールから行なうこともできるが、手間を要する。このため「まちづくり・コミュニケーション・システム」のために開発したデータベースとテーブルの自動作成のためのスクリプトが、新しい OS と SQL サーバにおいても使用できれば便利である。

Windows2000Server 上で開発した初版については文献 21（pp.42～56）で解説した。その後、Windows2003Server に移植した。この方法をそのまま Windows2012Server 上で実行しても旧 OS 上のように正しくテーブルを構築することができない。そこで、奥尻サイトに含まれる掲示板機能を例として、デバッグ情報の取得・参照方法と、それを用いて修正したプログラム（スクリプト）や作業手順を以下に解説する。

１）テーブルの自動構築のためのスクリプト

[1] アクセス情報等を記録するテーブル群の作成

WEB サイトを構成する各ページ(.asp)へのアクセスを SQL サーバに記録する機能である。必要とするテーブル群を作成するためには、WEB ブラウザでアクセスしたサーバ上の hdbmk_counter.asp を起動すると、ここから呼び出されるインクルードファイル tdbmk_database.inc で定義された関数が、データベースを作成し、tdbmk_counter.inc で定義された関数がアクセスカウンタに必要なテーブルを作成する。この単純なテーブルは、ページのファイル名、累計ヒット数、最後のアクセス、最後の更新日を記録している。作成するデータベース名、ローカルファイル名、ユーザ名、パスワードは tdbm,_database.inc のプログラム中に直接に記述しているため、運用するシステムに合わせて書き換える。

この構築過程の途中結果は、成否にかかわらず全てテキスト形式で記録され、終了時点でブラウザ画面に表示される。 更に、エラーが生じた場合には、Windows 配下の LogFiles の中に、W3SVC のログとしても記録されるため、デバッグの手がかりを得ることができる。

データベースとテーブルの作成が完了した後に、新たなページを(.asp)作成し最下行に

<--#include FILE=”include/t_counter2.inc” -->

というインクルードでスクリプトを追加すると、このページに最初にアクセスした時点でこのページ（ファイル名）が上記のテーブルに追加されると共に、カウンター１にセットされる。カウンターの値はアクセスの度にインクリメントされ、ページの中に表示される。

カウンターはデータベースが動いているかどうかの簡便なチェックとしても有効である。

[2] 意見投稿と審査のためのテーブル群の作成

ユーザからの意見や資料の投稿を受け付け、審査員によるオンライン審査の上で公開する機能である。この機能において作成するテーブル構成は、スキーマに相当する def.csv 形式のファイルによって目的に応じて自由に定義できる。テーブル群の作成は、WEB ブラウザからアクセスしたデータベース作成ページ(HP_dbmk.asp)でパラメータを設定し実行ボタンを操作することにより行なう。このスクリプトは、スキーマで定義された投稿意見のテーブル以外に、審査の進行状況や各審査員への依頼数を管理するテーブルも作成する。

移植後のデータベース名、アカウントなどは、tdb_user.asp の中で定義するため、移植先のシステムに合わせて修正する。

テーブル構成を定義するスキーマである def.csv 形式のファイルの中では、データを配置するローカルアドレスも定義しているため、移植先に合わせて書き換える必要がある。なお、def.csv 形式のファイルは複数個システム上に存在していてもよく、名称は、def.csv とは異なっていてもよい。よって移植前の def.csv を別名または別場所に残しておくことができる。使用するファイルは、VB スクリプトが参照する asp-cfg.txt ファイルと、tehaishi サービスが参照する cfg.txt によって、フルパスで特定する。

asp ファイルに含まれる VB スクリプトをデバッグするためには、「idebug=1 」といコマンドでデバッグ用のフラグを立て、豊富なメッセージを出力させる。必要であればバッグ用のコマンドを追加すると共に、エラー発生箇所付近に、Response.End コマンドを追加してそこで処理を中断することにより、問題箇所を特定する。

上記[1]と[2]の作業を通じて、結論的に W2000Server 上で開発し、その後 W2003Server に移植し稼働していた全ての機能を、W2012Server 上に移植することができた。なお、[1]と[2]を同じデータベース上に共存させることも別のデータベースとすることもできる。同名のデータベース上に共存させるためには、[2]→[1]の順で行なう必要があるため、デバッグを伴う移植に際しては単純な[1]のテストを通過した後、一度削除して[2]に進むのが速い。

２）SQL2012 への移行における修正点と留意事項

Windows2003Server 以前の OS におけるテーブル作成のためのスクリプトや作業手順を W2012Server＋SQL2012 に移行するための修正点と留意事項を、以下にまとめる。

[1] デバッグのため、idebug という変数に 1 を代入しておくと、デバッグモードで処理が行われ、詳細なエラーをクライアント側に表示するようになる。プログラムからメッセージを作成し、これをクライアント側に返送する方法は、IIS の仕様変更等にかかわらず有効であるため、システムの載せ替えが楽である。しかしながら、審査機能付き掲示板系のデータベース作成のコールバック（form の action=”tdbmk_main.asp”）におけるエントリポイント tdbmake_main 関数から呼び出す tdbmk_database 関数は、インクルード(tdbmk_database.inc)の中で定義されている。tdbmake_main 関数では idebug 変数に 1 を代入しているにも関わらず、tdbmk_database 関数の中で idebug により処理を切り替えているケースでは、多くの場合、idebug=0 の側に分岐している。つまりインクルードにより参照したファイルの VB スクリプトにおけるグローバル変数の外縁は不明確である。よって、

各種関数には、idebug を引数として明示的に渡すのが好ましい。そこで、tdbmk_database 関数に引数を一つ追加し、呼び出し元から idebug を引数として与え、呼び出し先で引数 idebug として受け取る方法により、この不整合を解決した。

[2] クライアント側にエラーメッセージを返送する方法として、Response.Write 関数で直接メッセージを作成する方法、tAdd_Serverlog(文字列)で、サーバログにエラー情報を追記する方法、tMkKekkaOutcom_html(文字列)により、最終的に作成する処理終了後の一次的な表示頁にエラー情報を追記する方法が用意されている。後２者の方法では、最後に転送したメッセージのみが表示されるため、途中で生じたエラーに関する情報は掻き消されている。このような諸点に関して、設定手続きのためのプログラムは改良の余地がある。

[3] 上記 tdb_makedatbase.inc および tdbmk_database.asp の中で SQL コマンド文字列を発行してデータベースの構築している。コマンド文字列の中で指定する mdf ファイル(データベースを格納する)のサイズとして従来、固定的に 1 MB を指定していたが、SQL2012 では、最低５MB を求めており、これより小さな値を指定すると、エラー終了となる（データベースは作成されない)。その際のエラーコードが、データベースが既に存在する場合と同じであるため、スクリプトからは「その名前のデータベースは既に存在」という表現のエラーメッセージが表示されるが、「データベース指定における数値の過小」とすべきである。

エラーコード番号ではなくメッセージ文字列を返すのは親切である。しかし、上記のようにコードの意味がバージョンにより変化し混乱する場合もあるため、固定的な（古い）変換表で文字列に変換せずに、更新後の最新システムにおけるメッセージを文字列として取得した上でそれを返送する方が、SQL や IIS の仕様変更に対して柔軟に対応ができる。

[4] データベースを作成するディレクトリとして、存在しない場所の名称を入力した場合には、ルートにデータベースが作成され、成功裡に終了するので注意が必要である。

データベースが既に存在する場合、内容が上書きされるように解説されているが、実際にはエラー終了する。よって、予め前述の方法で、カウンターやアクセスログのためのデータベースがプロジェクト名称を用いて作成されている場合には、それに各種のテーブルを追加する処理には進まない。

[5] IIS8.0 においては、asp スクリプトの中でのファイル関連処理において、親ディレクトリへのアクセスを規定（初期状態）で禁止しており、本処理系ではこれを設定で許可する必要がある。しかし、この設定は、IIS を再起動するまで反映されない。

親ディレクトリ参照ができるようにすることにより、プログラムの可搬性を高めることができる。実際には、まちづくりコミュニケーション・サイトのそれぞれの動作を定義する DEF.CSV ファイルの中で、ディレクトリを指定する際に、「../」から始まる相対アドレスを用いることにより、サーバの更新等に伴う修正箇所を大幅に減らすことができる。

[6] エラーメッセージの表示方法、エラーログの記述内容に関して、全般に、エラーメッセージは攻撃者に対して有用な情報を提供するリスクが大きいため、クライアント側には返送せず、サーバ上のログファイルとしてのみ記録を残すような方法が勧奨されるようにな

った。このため、OS の更新や SQL データベースの変更等に伴う作業において、エラーに関する情報は取得しにくくなっている。

現在では、障害無く確実に動作するシステムを完成させることよりもむしろ、悪意ある攻撃に対する安全性が問われている。セキュリティを高めるために屋上屋を重ねるような対応もありうるが、テストに使用する管理機能などは、設定終了後には、たとえ SE であっても不正アクセスできないように封鎖ないし削除しておくのが単純な回避策となる。

⑥ basp21 について

2001 年に本処理系を運用開始した段階で、サーバ側の ASP（2015 年現在では Classic ASP と呼ばれている）におけるファイル処理機能を追加するプラグインとして、有用であった。同じ実行形式は、Windows2012 上でも動作することを確認したが、３２ビットの実行形式を、６４ビットの OS 上で実行するために、IIS のアプリケーション・プールにおいて、３２ビットのプラグインの実行を許可するような設定を行わなければ正常に動作しない。

⑦ 巡回サービス

投稿状況をチェックし、新規投稿があれば査読委員に査読を依頼し、締切が迫った時に査読結果が届いていなければ督促を行い、査読結果が集まれば集計して掲載の合否を決定する、tehaishi という名称のサービスである。

Windows2012Server においても、サービスを登録して起動することができる。コマンドラインで「.¥tehaishiserviccconfig」とタイプして TehaishiServiceConfig.exe を起動すると、同じディレクトリに存在する cfg.txt で所在場所をフルパス指定した、事業別の設定を記述した def.csv の定義に従って、サービスを登録する。サービスの名称はプロジェクト名称を用いる。なお、def.csv はデフォルト名称であり、Tehaishi 用の cfg.txt、VBScript 用の asp-cfg.txt により、異なる名称のファイルを用いることができる。

初代の MSDE(2001 年)においては、パスワード無しの”sa”アカウントを設定することができた（デフォルト状態）。第二世代の MSSQL7 においては、パスワードを設定することが必須となったが、そのパスワード文字列に関する制約はなかった。第三世代の SQL2012 においては、このパスワードが「強いパスワード」であることを求められるようになった。

人間がこれらのパスワードをキーボード入力する環境であれば、パスワードを紙に記録しておき、必要な時に打鍵すれば良いが、上記のようにプログラムから設定を行い、設定後にサービスやサーバ上のスクリプトで実行するためには、パスワードをプログラム中で固定的に記述するか、外部ファイルにより指定する必要がある。リモートで設定作業を行うためには、外部ファイルはリモートでアクセスできる場所にある必要がある。このことは、セキュリティを保つことが次第に難しくなってきていることを意味する。複雑なパスワードは、コンピュータにとっては入力が簡単であるが、人間にとっては記憶したり間違いなく入力することはより困難である。逆説的であるが、パスワード無しでログインできる条件とし、SQL サーバにはローカルなアクセスのみを許可する方法が最も安全であるように見える。第二世代から、パスワードは、def.csv の中に記述するようになった。これを

利用できるのは、第二世代以後の tehaishi サービスである。

パスワードが不適切であるために処理に失敗した場合、その原因箇所を特定することは手間を要する。このデバッグ作業を容易にするためにアカウントやパスワードを含む豊富なエラーメッセージを出力すると、攻撃者に対して情報を提供する危険性が高まる。

第四世代である三次元データ保管庫においては、専用の検証プログラム（VC-4D、セットアップが終了後は撤去）の一部に組み込んだ機能）により SQL へのプログラムからのログイン条件を検証できるようにし、セットアップが完了した後はパスワードを暗号化したバイナリ形式のファイルの中に埋め込み、プログラムから解読して SQL サーバへのログインに使用している。

リスト 2-6-2 SQL データベース用コネクション文字列

```
"Provider=sqloledb;Data Source=" + su + ";Initial Catalog=" + pDBname;
//これは(local)の場合６階でのみうまく行く。
"Provider=MSDASQL;DRIVER=sql server;SERVER="+su+";DATABASE="+pDBname;//これはＡＳＰと同じ方法。
"Provider=MSDASQL;DRIVER=sql server;SERVER="+su;//データベースを指定せずにつないでみる。
"Provider=MSDASQL;DRIVER={sql server};SERVER="+su+";DATABASE="+pDBname;//これはＡＳＰデータベース作成と同じ方法。
"DRIVER={sql server};SERVER="+su+";DATABASE="+pDBname;//これはＡＳＰと同じ方法。
```

Provider とは、データベースへのアクセスのためにクライアント側でログインに使用する DLL の名称であり、本処理系においては、MSDASQL.dll を使用している。これは記述で省略された場合に使用されるデフォルトの dll である。

Driver とは、ODBC で使用されるデータベースアクセス用のクライアントである。

Server とは、アクセス対象である SQL サーバと、そのシステムがセットアップされているマシン名称を組み合わせたものである。マシン名だけを明記した場合には、SQL サーバとしては既定のサーバが存在すれば、それが指定されたことになる。SQL サーバ名だけを明記した場合、ローカルマシンの上の SQL サーバが指定される。全てが省略された場合には、ローカルマシン上の既定のサーバが指定される。

⑧ SE のための支援機能

設定作業における失敗は、様々な段階で発生する。解決のために、Windows の設定変更、SQL サーバの設定変更、本処理系における定義ファイルの修正で済む場合には、大きな変更は必要ない。この見極めのためには、設定時、修正時、事業終了時に使用する管理者用の VB スクリプト、および Windows 実行形式（サービス等）の設定・デバッグに一時的使用する EXE 形式の診断プログラムが有用である。これらは豊富なエラーメッセージを出す方が親切であるが、現在の状況(Windows2012+SQL2012)に詳細に対応する価値は余りない。しかし、少なくとも以前のシンプルなシステムであった時代に作成したエラーメッセージが、実際とは異なる状況を報告している場合がある点は、修正しておく必要がある。

DB オープンと、SQL コマンド実行のためには、共通のサブルーチンを作成すると共に、エラーに関して詳細にチェックを行い、必要であればログを出力するような処理がデバッグのために有効である。また、このようなサブルーチンを全てのプロジェクトで共通に使用することにより、将来の更なるサーバ間移植の手間を大幅に節約することができる。

しかしながら、便利な機能は、攻撃者のためにも有効なアプローチ手段を提供することにつながる危険性がある。インターネットの利便性よりも悪意ある攻撃のリスクが際立っている 2015 年時点の現状においては、移植に手間がかかる、かなり冗長性のある現行システムのままで当面運用を続けるしかない。

⑨ 簡単なデバッグの方法

デバッグのためのノウハウとして、以下の方法が有効である。

[1]最も基本的な SQL サーバの動作・接続を確認する段階では、単純な処理であるカウンター機能をテストに使用するのが便利である。
[2]プロジェクト毎（地区毎）のサイトを構築する際には、まずコミュニケーションのためのデータベースとそれを構成するテーブル群を、管理機能から作成しておき、同じデータベース名を用いてアクセスカウンタを作成することにより、同一名のデータベース内部にテーブル群を共存させることができる。これらの登録を一挙に実行する機能はまだ存在しない。
[3]複雑な処理を組み合わせた意見投稿コールバックの最後に、Respnse.End の行を挿入し、メッセージをクライアント側に表示させる。
[4]スクリプトが作成するログファイルの内容を調べる（仮登録配下の LogFile ディレクトリ）
[5]W3SVC のログを確認する（コンパイルエラー）

⑩ データベースの保存と移行

第二世代のサーバでは、OS として Windows2003 サーバを使用し、データベースとしては、MSSQL2005 を使用していた。移行先の第三世代のサーバでは、OS として Windows2012 サーバを使用し、データベースとしては、MSSQL2012 を使用した。中間のバージョンを飛ばした移植ではあったが、バックアップファイルに関して、上位互換が保たれていたため、旧版のデータベースを停止することなくバックアップを作成し、これを新たなデータベースにおいて復元することができた。

バックアップに際しては、Microsoft SQL Server Management Studio を使用した。バックアップは、対象となるデータベース毎に行い、選択・右クリックで現れるポップアップから、タスク→バックアップで、操作画面を開く。次に、以下の設定を行う。

「全般」ページでの設定
・バックアップの種類：[完全]を選択
・バックアップコンポーネント：[データベース]にチェック
・バックアップ先：[ディスク]にチェックしバックアップを格納するディレクトリを指定
「オプション」ページでの設定
・メディアに上書きします
[既存のメディアセットにバックアップする]をチェック
→[既存の全てのバックアップセットを上書きする]をチェック
（追加する、という設定では、復元段階で障害が生じる）
・信頼性　[完了時にバックアップを検証する]にチェック
[メディアに書き込む前にチェックサムを行う]にチェック

以上の設定を行った上で、OK ボタンによりバックアップを行う。バックアップ処理は数秒で完了する。

記憶媒体もしくは FFFTP 等を用いて、バックアップを新たな環境にコピーする。

復元は、予めデータベースが作成してある状態から出発し、同様に Microsoft SQL Server Management Studio を使用して作業を行った。既に作成してあるデータベースを選択・右クリックし、タスク→復元→データベースで、操作画面を開く。

「全般」ページで、
・復元するバックアップセットの復元元ファイルと場所：デバイスからとして、ファイルを選択する。
「オプション」ページで、
・復元オプション：既存のデータベースを上書きする（WITH　REPLACE）
・復元先：（新たなデータベースの）mdf ファイルと、ldf ファイルをフルパスで記述

以上の設定の上で、OK を押すと、現在のデータベースの上に、復元が行われる。復元は、バックアップよりも時間を要する。

このとき、クエリ等が開いていると、「アクセス権が取得できませんでした」というエラー終了となる。また、上記の「上書き」が選択されていない場合には、「ログの末尾が　アックアップされませんでした。」というエラー終了となる。

バックアップファイルの名称は随意であるが、.bak 等の拡張子が用いられている。

⑪ まとめ

仮想化を伴う、異なる OS への移築による保存継承の手順は、W2003→W2012 の移行を例とすると、以下のような流れとなる。

[1] DVD-ROM や、USB メモリ等のメディアから新たな OS を導入し、その途中でコマンドラインから VHD を作成してそこに新たな OS を導入する。
[2] 新たな OS 上で、IIS を導入する。
[3] 新たな OS 上で、SQL エンジンを導入する。
[4] 新たな IIS 上で、既存のプロジェクト別の仮想ディレクトリを作成する。
[5] 新たな IIS 上で、ASP の実行環境を設定する（エラー処理等）。
[6] 古いデータベースをバックアップし、新たな SQL エンジン上に復元する。
[7] 新たな IIS 上で、FTP のアクセス環境を作成する。
[8] 古い実行環境を VHD に変換し、ファイルとしてアーカイブする。

W2003+SQL7 から W2012＋SQL2012 への移植にあたって、設定変更（条件緩和）により旧システムとの互換性を実現することができた項目は、以下の通りである。

[1] 32 ビットのサービスを許容するように IIS を設定する
[2] 上位ディレクトリへの参照を許容するように設定する
親ディレクトリの参照方法として、”..”を使用することは初期状態ではできなくなった。使用するためには IIS での設定を行い、再起動する手順が求められる。これにより、インクルードファイルがアクセスできないことによるコンパイルエラーを防ぐことができ、ま

た移植に伴い必要となる設定ファイル def.csv の修正箇所を少なくすることができる。
[3] VBScript のコンパイルエラーをクライアント側に表示する方法は、可能と解説されているが、いずれの方法も失敗した。サーバ側に開発・デバッグ環境をセットアップしておき、サーバ上で直接見る以外の方法はなさそうである。

コンパイルが成功した後には、スクリプトでクライアント側に返すメッセージとして、あるいはログファイルとして、デバッグのために有用な情報を得ることができる。

結果的に、アプリケーション側で以下のようなプログラムないしアプリケーション内部の設定ファイルの修正を行った。

[1] asp-cfg.txt を修正し、def.csv が存在する場所をスクリプトに正しく伝える（２か所）
[2] def.csv の内容を修正する（１か所）
[3]SQL サーバのアカウントとパスワードを記述する tdb_user.asp を修正する（３か所）
SQL パスワードの修正箇所は、VB スクリプト用３か所、サービス用１か所である。 SQL サーバへのパスワードを修正する。パスワードは必ず設定しなければ動作しない。しかも、強いパスワードである必要がある。MSDE ではパスワード無しで動作した。MSSQL7 では、弱いパスワードでも動作した。
[4]データベースを新規に作成する VB スクリプト（dbmake、SQL 文字列固定記述）において、MDF ファイルのサイズを１MB から５MB に増量する。修正前の失敗に関して、従来のシステムでは、「データベースは既にあります」というメッセージを発するため、設定者は混乱する。
[5]投稿用のコールバック関数 VB スクリプトを修正する。
コメントアウトされている on Error Resume Next を有効にする（２か所）

２－７．ＶＣ－４Ｄ　三次元データ保管庫

（１）概要

仮想コンバータを用いたデータベース入出力プログラム（以下、本システム）は、サーブレットエンジン（Apache Tomcat）上で動作する Web アプリケーションである。

本システムでは、主に以下の３機能を有する。

① アップロード受付機能

ユーザが作成した形式記述ファイル（メタファイル）と形状記述ファイル（データファイル）を受け付け、サーバ上の所定のフォルダ下に保存する。

保存した各ファイルを仮想コンバータへ受け渡し、３次元データをデータベース上に展開する。

② ダウンロード受付機能

ユーザが作成した形式記述ファイル（メタファイル）と登録済データ（選択式）を受け付け、サーバ上の所定のフォルダ下に保存する。

保存したファイルと選択した登録済データを仮想コンバータへ受け渡し、サーバ

上に形状記述ファイルを生成する。

③ 履歴参照機能

ユーザが行ったアップロード／ダウンロード要求を、一覧／詳細表示する機能。

各要求内容の進捗状況や、処理結果（合否）内容が参照できる。

ダウンロード要求の場合は、生成された形状記述ファイルのダウンロードが行える。

（２）動作環境

①動作環境

表 2-7-1 VC-4D の動作環境

項目	内容
サーバ OS	Windows 7, Windows XP
クライアント Web ブラウザ	IE 8 以降、Mozilla Firefox 18 以降

※ 本システムの動作試験を行った環境

②利用アプリケーション一覧

表 2-7-2 VC-4D の利用アプリケーション

項目	内容
サーブレットエンジン	Apache Tomcat 7.0.35
Java SDK	JDK 7
HTTP サーバ	Apache HTTP Server 2.2
データベース	Microsoft SQL Server 2008 R2 SP1 Express Edition （本システムの動作試験で使用）

（３）システムのインストール

①Java SDK

jdk-7-windows-i586.exe（64 ビット OS の場合は、jdk-7-windows-x64.exe）より、Java SDK のインストールを行う。

以下は、32 ビット OS 上でインストールを行う場合の例である。

1) jdk-7-windows-i586.exe をダブルクリックする。

2) インストールウィザードに従い、デフォルトで指定される内容で各画面を進める。

3) インストール終了。

②Apache Tomcat

1)インストール

apache-tomcat-7.0.35.exe より、Apache Tomcat のインストールを行う。

リスト 2-7-1 Apache Tomcat のインストール手順

(1) apache-tomcat-7.0.35.exe をダブルクリックし、インストールウィザードを表示させる。
(2) 「License Agreement」画面では、”I Agree” ボタンを選択する。
(3) 「Choose Components」画面では、デフォルトの内容で”Next” ボタンを選択する。

(4) 「Configuration」画面では、デフォルトの内容で”Next” ボタンを選択する。
(5) 「Java Virtual Machine」画面では、デフォルトの内容で”Next” ボタンを選択する。
(6) 「Choose Install Location」画面では、デフォルトで指定されるインストールフォルダ以外を選択する。
【例】C:¥Apps¥Apache Software Foundation¥Tomcat 7.0
【理由】
Windows Vista 以上の OS において、User Account Control（UAC）を有効化している場合は、OS 管理下におかれているフォルダ（例　C:¥Program Folder）下にインストールを行った場合、Web アプリケーションが正常に動作しない可能性があるため。
(7) ファイルが展開され、インストール終了。
(8) Windows ファイアーウォールにおいて、ポート番号 8080（HTTP）と 8009（AJP）でのアクセス許可を設定する。　※ 外部 Web サーバと連携する場合のみ

2) 起動／停止

Windows スタートメニューより、「Apache Tomcat 7.0 Tomcat7」→「Monitor Tomcat」を指定しダイアログを表示させる。

ダイアログ上部の「General」タブを指定し、”Service Status:” 下部にある “Start” ボタンで起動、“Stop” ボタンで停止を行う。

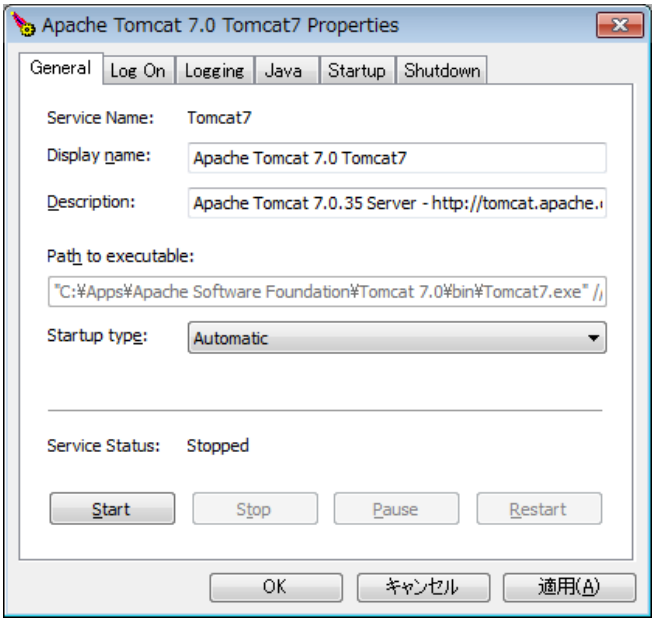


図 2-7-1 Apache Tomcat 起動／停止ダイアログ

③ Web アプリケーション（本システム）

1) インストール

vc4d.war を、サーブレットエンジンのアプリケーションフォルダ下にコピーし、リスト 2-7 の操作を行う。

リスト 2-7-2 WEB アプリケーションのインストール手順

(1) Apache Tomcat を停止する。
(2) Apache Tomcat インストールフォルダ中の、webapps フォルダ（アプリケーションフォルダ）を開く。
(3) 上記フォルダへ、vc4d.war ファイルをコピーする。
(4) Apache Tomcat を起動する。
(5) 起動後、アプリケーションフォルダ内に vc4d フォルダが生成されていることを確認する。

2) 設定変更

Web アプリケーションの設定変更を行うには、共通設定プロパティファイル（vc4d.properties）を編集する。

基本的に設定変更が必要な項目は、以下の通り。

- アップロード／ダウンロード要求で受け付けたファイルを保存するフォルダ。
- 仮想コンバータプログラムのパス指定。
- データベースの接続情報。

共通設定プロパティファイルは、

${TOMCAT_HOME}/webapps/vc4d/WEB-INF/classes

フォルダ下に存在する。${TOMCAT_HOME} はサーブレットエンジン インストールフォルダを示す。

共通設定プロパティファイルにおいて設定可能な項目の詳細は、(6)③1) を参照。

1. Apache Tomcat を停止する。
2. ${TOMCAT_HOME}/webapps/vc4d/WEB-INF/classes/vc4d.properties を編集し、保存する。
3. Apache Tomcat を起動する。

④データベース設定

本システムがデータベースへアクセスするために、Microsoft SQL Server 2008 R2 SP1 Express Edition の設定を変更する。

尚、設定変更は付属の SQL Server 2008 R2 SP1 Management Studio Express を利用する。

1) サーバ認証方法の変更

Windows スタートメニューより、「Microsoft SQL Server 2008 R2」→「SQL Server Manager Studio」を指定し起動する。

画面左部の「オブジェクト エクスプローラ」に表示されるインスタンス名称を右クリックし、プロパティを表示させる。

プロパティダイアログ中の「セキュリティ」ページを選択し、「サーバ認証」を “SQL Server 認証モードと Windows 認証モード” に変更する。

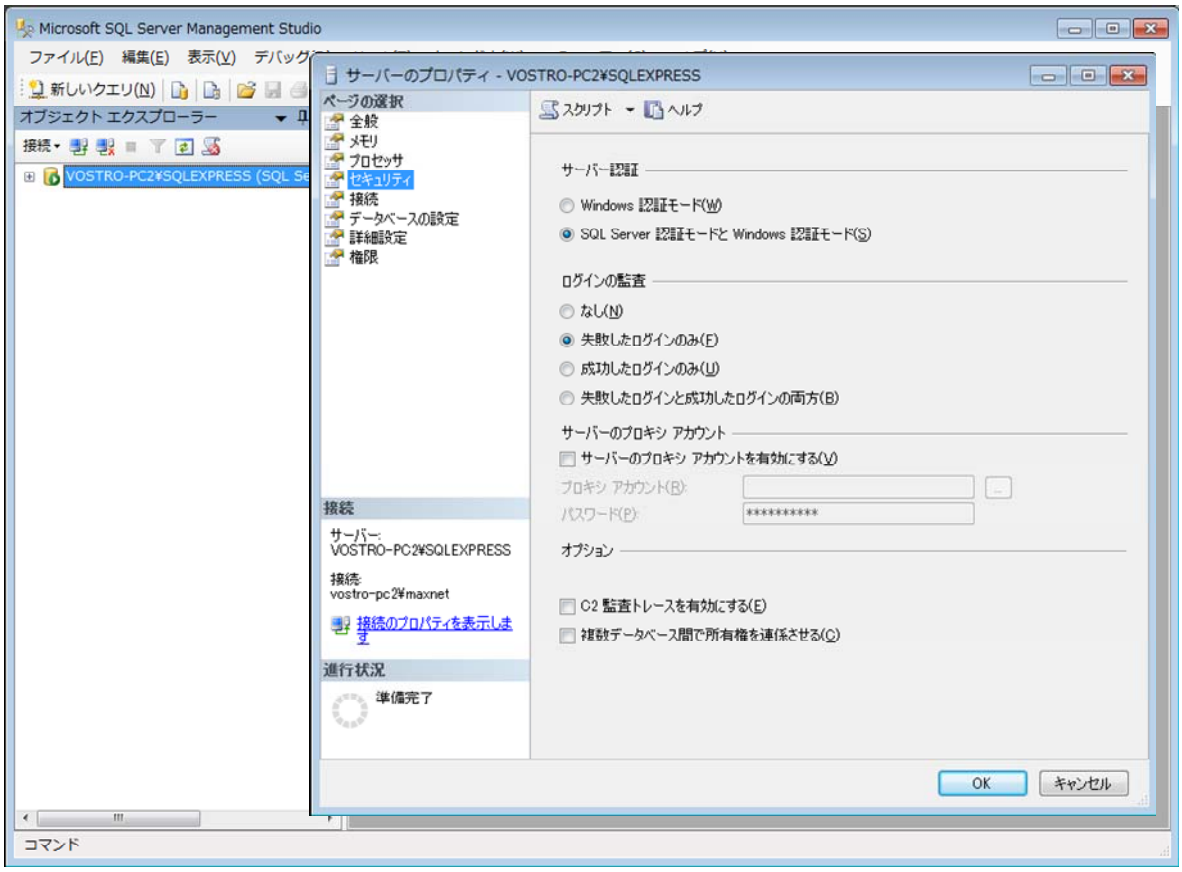


図 2-7-2 SQL サーバ認証方法変更

次に、データベースへのログインに使用する sa ユーザの有効化を行う。

画面左部の「オブジェクト エクスプローラ」より、「セキュリティ」→「ログイン」下に表示される sa ユーザ名称を右クリックし、プロパティを表示させる。

プロパティダイアログ中の「状態」ページを選択し、「データベースエンジンに接続する権限」項目を“許可”、「ログイン」項目を“有効”に変更する。

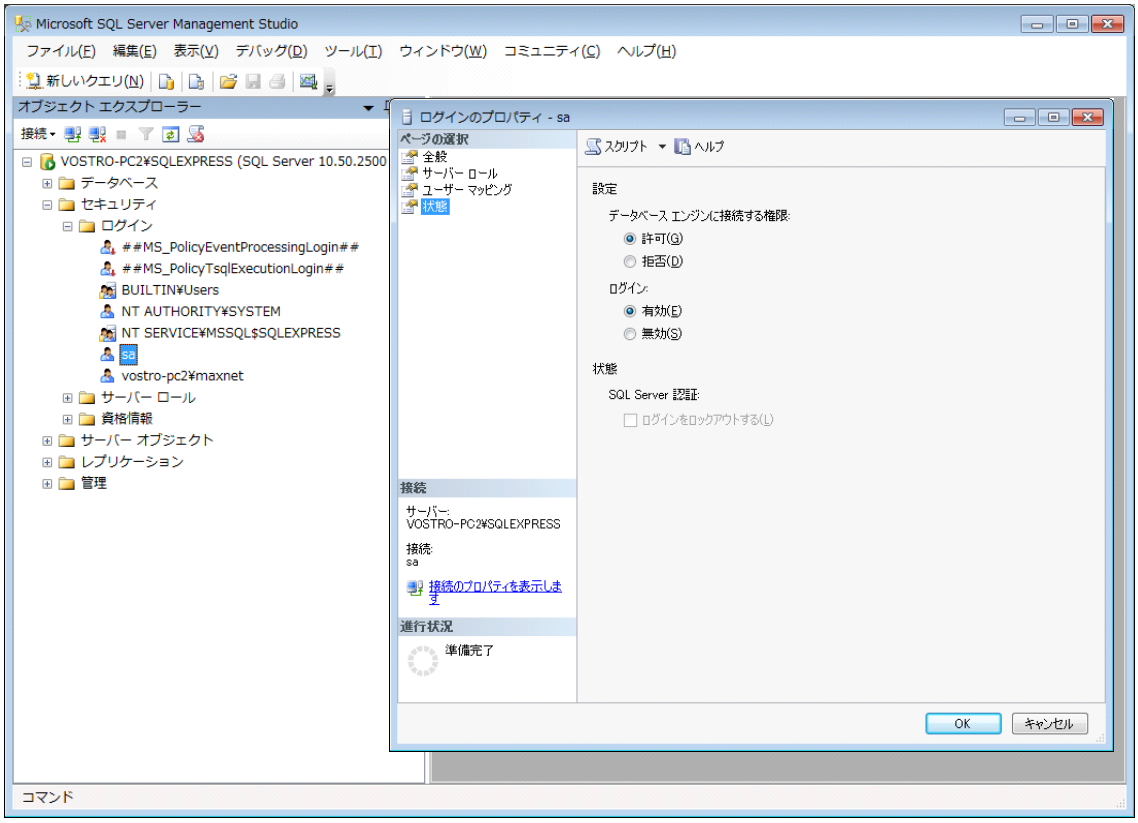


図 2-7-3 SQL サーバ sa ユーザの有効化

2) TCP／IP の変更

Windows スタートメニューより、「Microsoft SQL Server 2008 R2」→「構成ツール」→「SQL Server 構成マネージャ」を指定し起動する。

画面左部の「SQL Server ネットワーク構成」より、使用するインスタンスのプロトコルを選択する。

画面右部に表示される「TCP/IP」項目を右クリックし、「プロパティ」を選択する。

プロパティ画面上部の「IP アドレス」を選択し、SQL Server で接続を許可する IP アドレスを以下の内容で変更する。（localhost で接続する場合は、IP アドレス ”127.0.0.1” または ”::1” を変更）

表 2-7-3 SQL サーバの TCP/IP 設定(local host)

項目	設定内容
TCP ポート	1433
TCP 動的ポート	（空白）
アクティブ	はい
有効	はい

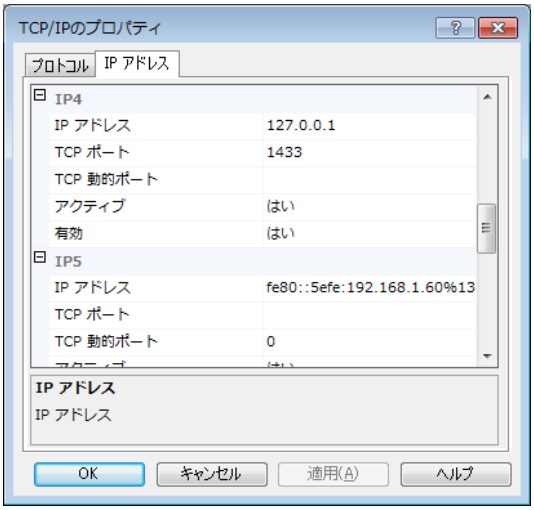


図 2-7-4 SQL サーバ TCP/IP プロパティ変更(local host)

続いて、プロパティ画面最下部にある“IPAll”の項目を、以下の内容で変更する。

表 2-7-4 SQL サーバの TCP/IP 設定(IPAll)

項目	設定内容
TCP ポート	1433
TCP 動的ポート	（空白）

図 2-7-5 SQL サーバ TCP/IP プロパティ変更（IPAll）

最後に、画面右部に表示される「TCP/IP」項目を右クリックし、「有効化」を選択する。

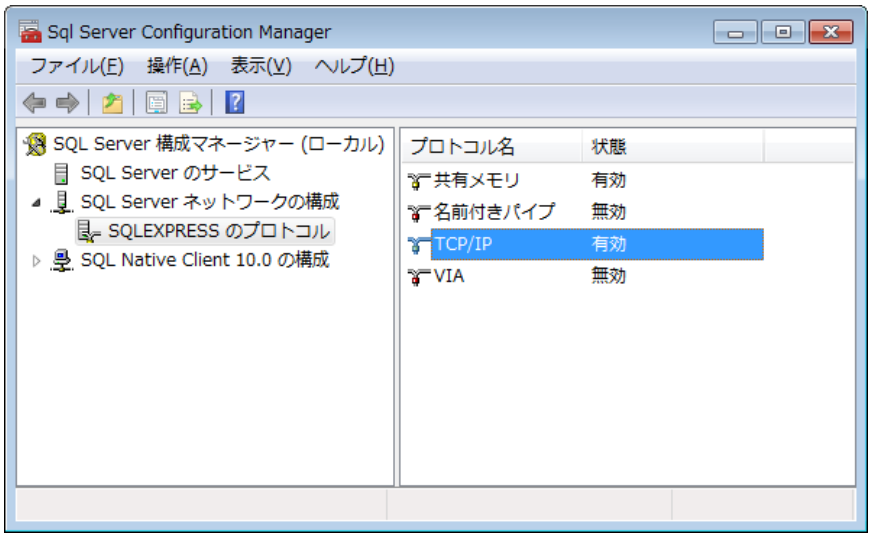


図 2-7-6 SQL サーバ TCP/IP 有効化

3) システム用データベース／テーブルの作成

SQL Server 上に本システムで利用するデータベースおよびテーブルを作成する。

データベース名称は vc4d とする。

利用するテーブルは Master と JobTask の 2 つで、その用途を表 2-9 に示す。

表2-7-5 VC-4Dシステム用テーブルの用途

テーブル名	用途
Master	仮想コンバータにおいて3次元データを展開したデータベース名を管理。
JobTask	仮想コンバータの実行スケジュールを管理。

Master テーブルの各カラム内容は、表 2-10 の通り。

表 2-7-6 VC-4D システム用 Master テーブルの列構成

項目名	項目 ID	属性	必須	説明
データベース ID	Id	Int	○	自動採番されるユニーク ID。
データベース名	Name	Varchar(256)		3次元データが展開されたデータベース名称。

JobTask テーブルの各カラム内容は、下記の通り。

表 2-7-7 VC-4D システム用 JobTask テーブルの列構成

項目名	項目 ID	属性	必須	説明
受付 ID	Id	Varchar(64)	○	アップロード／ダウンロード要求受付時に発行するユニーク ID。
セッション ID	Sid	Varchar(64)	○	ユーザの Web ブラウザに Cookie データとして登録されたセッション ID。
処理種別	JobType	Smallint	○	アップロード要求＝1、ダウンロード要

				求＝2。
処理状態	Status	Smallint	○	未処理＝0、処理中＝1、処理済＝2
処理結果	Result	Bit	○	仮想コンバータでの処理に成功した場合＝1
処理結果コード	ResultCode	Int	○	仮想コンバータからの返却値
処理メッセージ	Message	Varchar(1024)		エラーメッセージなど。
お名前	Name	Varchar(256)		ブラウザで入力した「お名前」
保存フォルダ	Folder	Varchar(512)	○	ファイルが保存されているフォルダ名
メタファイル名	MetaFile	Varchar(256)		サーバ内で保存される形式定義ファイル名
メタファイル名	MetaOrgFile	Varchar(256)		ブラウザで指定した「形式定義ファイル」のファイル名
データファイル名	DataFile	Varchar(256)		サーバ内で保存される形状記述ファイル名
データファイル名	DataOrgFile	Varchar(256)		ブラウザで指定した「形状記述ファイル」のファイル名
データベース名	DbName	Varchar(256)		(アップロード)仮想コンバータで作成するデータベース名称 (ダウンロード)ブラウザで選択した「登録済みデータ」
登録日時	RegistDate	DateTime	○	処理を受け付けた日時。
開始日時	StartDate	DateTime		仮想コンバータを実行した日時。
終了日時	FinishDate	DateTime		仮想コンバータの処理が終了した日時。

データベースは、SQL Server Manager Studio より作成する。

画面左部の「オブジェクト エクスプローラ」に表示されるインスタンス名称を選択し、下段に現れる「データベース」を右クリック→「新しいデータベース」を選択する。

データベース作成用ダイアログの画面右上部にある「データベース名称」へ ”vc4d” と入力し、OK ボタンを選択する。

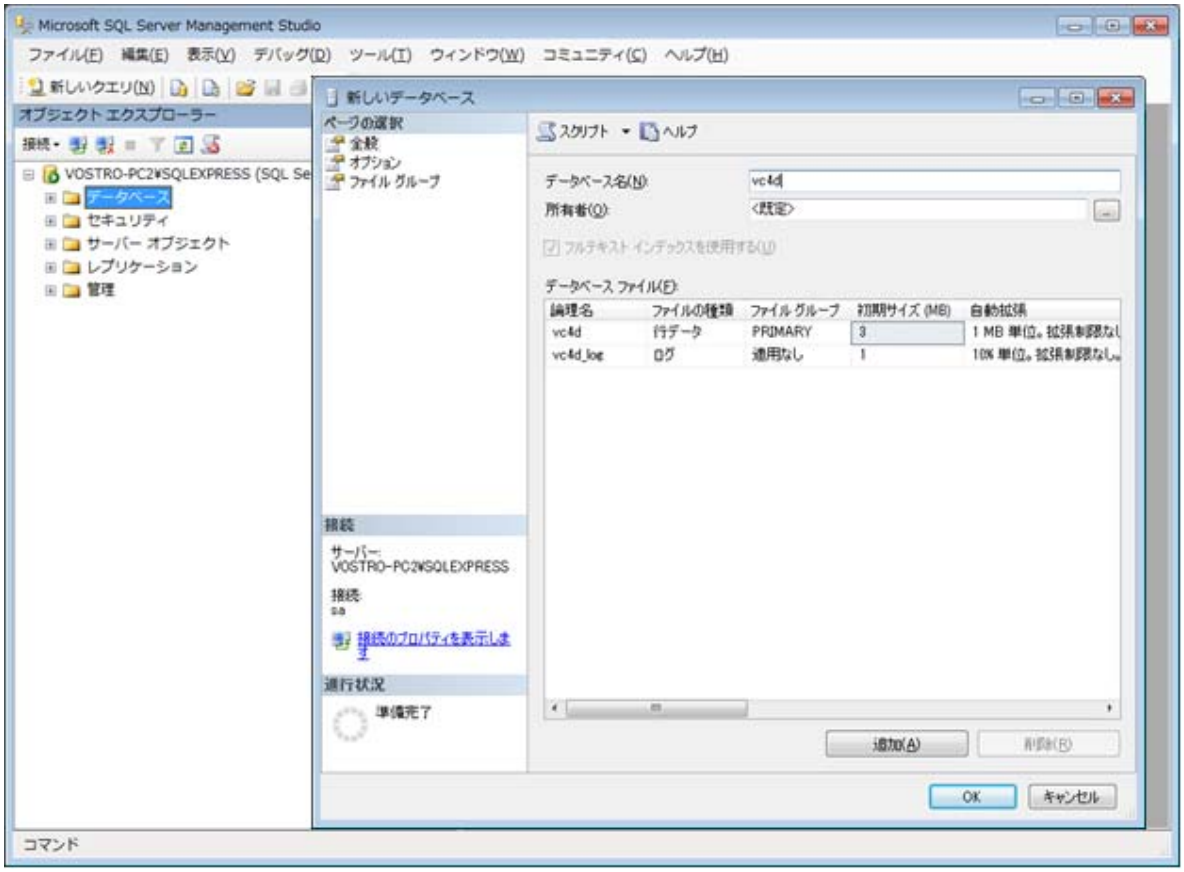


図 2-7-7 SQL サーバ　新規データベース作成

各テーブルは、SQL Server Manager Studio より、以下のクエリを実行して作成する。

リスト 2-7-3 テーブル作成用クエリ

```
USE [vc4d]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[Master](
	[Id] [int] IDENTITY(1,1) NOT NULL,
	[Name] [varchar](256) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[JobTask](
	[Id] [varchar](64) NOT NULL,
	[Sid] [varchar](64) NULL,
	[JobType] [smallint] NOT NULL,
	[Status] [smallint] NOT NULL,
	[Result] [bit] NOT NULL,
	[ResultCode] [int] NOT NULL,
	[Message] [varchar](1024) NULL,
	[Name] [varchar](256) NULL,
```

```
	[Folder] [varchar](512) NOT NULL,
	[MetaFile] [varchar](256) NULL,
	[MetaOrgFile] [varchar](256) NULL,
	[DataFile] [varchar](256) NULL,
	[DataOrgFile] [varchar](256) NULL,
	[DbName] [varchar](256) NULL,
	[RegistDate] [datetime] NOT NULL,
	[StartDate] [datetime] NULL,
	[FinishDate] [datetime] NULL
) ON [PRIMARY]
GO

SET ANSI_PADDING OFF
GO
```

⑤ Web サーバ

1) インストール

httpd-2.2.22-win32-x86-no_ssl.msi より、Web サーバ（Apache HTTP Server）のインストールを行う。

尚、インストールを行う環境上に別の Web サーバ（IIS 等）が導入されている場合はポート番号が衝突するため、サービスを停止する必要がある。

リスト 2-7-4 Web サーバ Apache のインストール手順

(1) httpd-2.2.22-win32-x86-no_ssl.msi をダブルクリックし、インストールウィザードを表示させる。
(2) 「License Agreement」画面では、"I Agree" ボタンを選択する。
(3) 「Reed This First」画面では、"Next" ボタンを選択する。
(4) 「Server Information」画面では、各項目に適切な値を入力し"Next" ボタンを選択する。
(5) 「Setup Type」画面では、デフォルトの内容で"Next" ボタンを選択する。
(6) 「Destination Folder」画面では、デフォルトで指定されるインストールフォルダ以外を選択する。
【例】C:¥Apps¥Apache Software Foundation¥Apache2.2
【理由】Windows Vista 以上の OS において、User Account Control（UAC）を有効化している場合は、OS 管理下におかれているフォルダ（例　C:¥Program Folder）下にインストールを行った場合、Web サーバの設定が反映されない可能性があるため。
(7) ファイルが展開され、インストール終了。
(8) Windows ファイアーウォールにおいて、ポート番号 80（HTTP）でのアクセス許可を設定する。

2) 設定変更

Windows スタートメニューより、「Apache HTTP Server 2.2」→「Configure Apache Server」→「Edit the Apache httpd.conf Configuration File」を選択し、テキストエディタを表示させる。

テキストエディタで行末までスクロールし、以下の内容を追加後に保存・終了する。

リスト 2-7-5 Apache 設定ファイル末尾への追加項目

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so

<Location /vc4d>
  ProxyPass ajp://localhost:8009/vc4d
</Location>
```

3) 起動／停止

Windows スタートメニューより、「Apache HTTP Server 2.2」→「Control Apache Server」を選択し、下に表示される「Start」「Stop」メニューより起動／停止を行う。

（4）システムのアンインストール

① Web アプリケーション（本システム）

以下に、本システムのアンインストール方法を記す。

リスト 2-7-6 本システムのアンインストール手順

(1) Apache Tomcat を停止する。
(2) Apache Tomcat インストールフォルダ中の、webapps フォルダ（アプリケーションフォルダ）を開く。
(3) 上記フォルダ中の、vc4d.war ファイルと vc4d フォルダを削除する。
(4) アップロードにより保存されたファイルは、共通設定プロパティファイルに指定したフォルダ下に存在する。
これらのファイルはアンインストールの対象とならないため、手動で削除を行う。
（バックアップが必要な場合は、任意の圧縮ツールを使用して行うこと）

②その他のアプリケーション

Java SDK／Apache HTTP Server／Apache Tomcat は、Windows コントロールパネルの「プログラムと機能」より、アンインストールを行う。

本システムで利用するデータベース（vc4d）や仮想コンバータより登録された 3 次元データベースは、SQL Server Manager Studio より手動で削除を行う。

画面左部の「オブジェクト エクスプローラ」に表示されるデータベース名称を選択し、右クリック→「削除」を指定する。

データベース削除用ダイアログにおいて、OK ボタンを選択する。

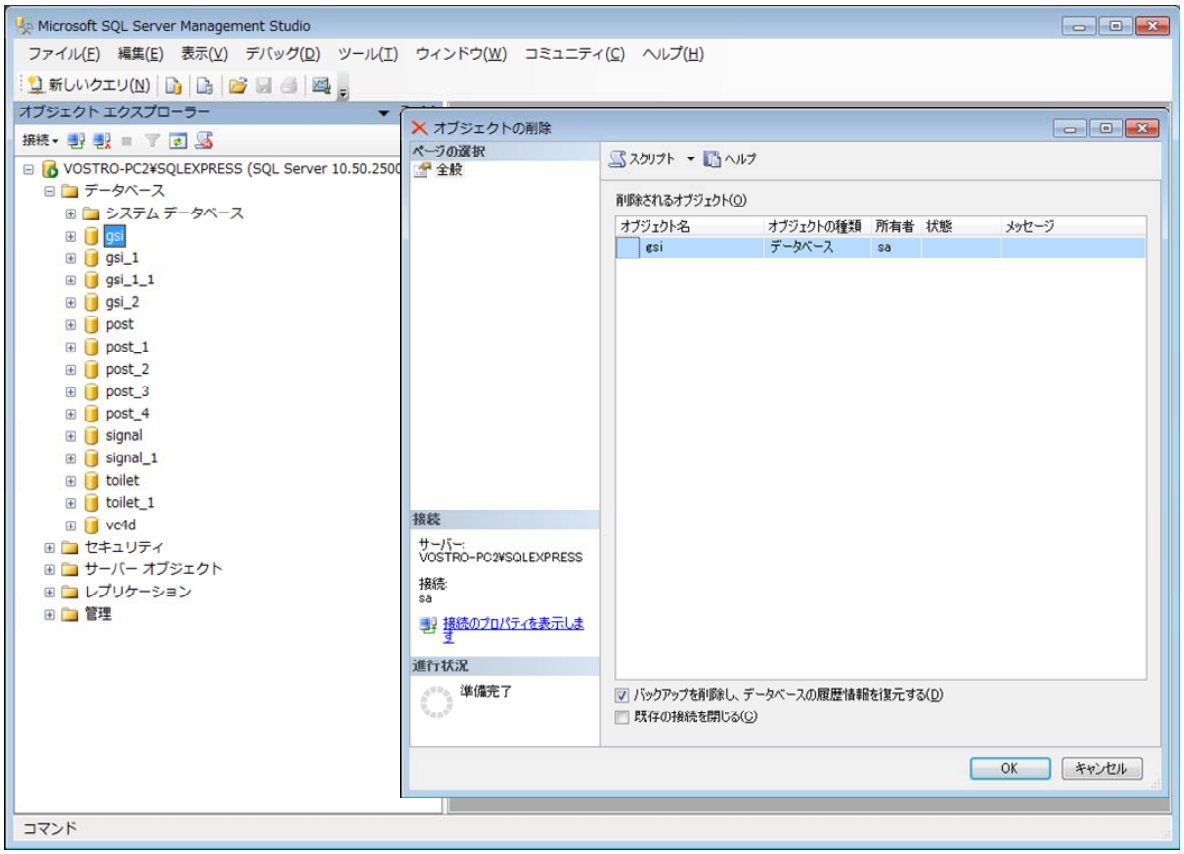


図 2-7-8 SQL サーバデータベース削除

（５）システム構成

①Web アプリケーション ファイル構成

１）全体構成

サーブレットエンジンのインストールフォルダ下に展開される、本システムのフォルダ／ファイル構成。

```
${TOMCAT_HOME}                                          Apache Tomcat インストールフォルダ
  ├ logs/                                               ログ出力フォルダ
  └ webapps/                                            Web アプリケーションフォルダ
     └ vc4d/                                            本システム アプリケーションフォルダ
        ├ META-INF/
        ├ resources/
        │  └ css/
        │     └ style.css                               スタイルシートファイル
        │
        └ WEB-INF/
           ├ classes/
           │  ├ vc4d/                                   クラスファイル格納フォルダ
           │  ├ vc4d.properties                         共通設定プロパティファイル
           │  ├ job-messages.properties                 仮想コンバータ用メッセージファイル
           │  ├ applicationContext.xml
           │  ├ dataAccessContext.xml
           │  ├ webContext.xml
           │  ├ messages-spring_jp.propertiesl
           │  └ logback.xml                             ログ出力定義 XML
           │
           ├ lib/
           ├ view/                                      JSP ファイル格納フォルダ
           │  ├ download-index.jsp
           │  ├ parts-footer.jsp
           │  ├ parts-header.jsp
           │  ├ parts-menu.jsp
           │  ├ parts-status-detail.jsp
           │  ├ status-detail.jsp
           │  ├ status-index.jsp
           │  ├ status-search.jsp
           │  ├ upload-index.jsp
           │  ├ http-404.jsp
           │  └ http-500.jsp
           │
           └ web.xml                                    Web アプリケーション配備記述子 XML


${仮想コンバータ用データフォルダ}
  ├ exe/                                                仮想コンバータコンソール格納フォルダ
  ├ data/                                               3 次元データベースファイル格納フォルダ
  └ file/                                               アップロードファイル格納フォルダ
```

図 2-7-9 本システムのフォルダ／ファイル構成

2）Web アプリケーション リソースファイル構成

本システムで使用する、各リソースファイルの構成。

各ファイルの内容を変更することで、システム動作の変更が行える。

基本的に内容変更が必要なファイルは、“vc4d.properties”と“job-messages.properties”の２ファイルのみ。

尚、Web アプリケーションをアンインストールした場合、これらのファイルも削除されるため、内容変更を行ったファイルは任意の場所にバックアップを行うこと。

```
${TOMCAT_HOME}/webapps/vc4d/WEB-INF/classes/
  ├ vc4d.properties                 共通設定プロパティファイル
  ├ job-messages.properties         仮想コンバータ用メッセージファイル
```

```
  ├ applicationContext.xml                    Springframework 定義 XML
  ├ dataAccessContext.xml                     （同上）
  ├ webContext.xml                （同上）
  ├ messages-spring_jp.propertiesl            メッセージプロパティファイル
  └ logback.xml                               ログ出力定義 XML
```

図 2-7-10 本システムのリソース構成

3）Web アプリケーション HTML 画面構成

Web ブラウザで表示される、各 HTML 画面の構成。

テキストエディタ等で編集することで、表示する HTML の内容が変更される。

尚、Web アプリケーションをアンインストールした場合、これらのファイルも削除されるため、内容変更を行ったファイルは任意の場所にバックアップを行うこと。

```
${TOMCAT_HOME}/webapps/vc4d/
  ├ WEB-INF/
  |   └ views/
  |       ├ download-index.jsp          ダウンロード受付用 JSP ファイル
  |       ├ parts-footer.jsp            （共通）フッタ用 JSP ファイル
  |       ├ parts-header.jsp            （共通）ヘッダ用 JSP ファイル
  |       ├ parts-menu.jsp                      （共通）メニュー用 JSP ファイル
  |       ├ parts-status-detail.jsp             （共通）履歴詳細用 JSP ファイル
  |       ├ status-detail.jsp           履歴一覧用 JSP ファイル
  |       ├ status-index.jsp            履歴詳細用 JSP ファイル
  |       ├ status-search.jsp           履歴検索用 JSP ファイル
  |       ├ upload-index.jsp            アップロード受付用 JSP ファイル
  |       ├ http-404.jsp                        HTTP ステータスコード 404 用 JSP ファイル
  |       └ http-500.jsp                        HTTP ステータスコード 500 用 JSP ファイル
  |
  └ resources/
      └ css/
          └ style.css                  スタイルシートファイル
```

図 2-7-11 HTML 画面構成

4）ログファイル構成

サーブレットエンジンおよび本システムで出力される、ログファイルの構成。

日単位にファイル名の“YYYY-MM-DD”が変更されて出力される。

```
${TOMCAT_HOME}/logs/
  ├ vc4d.YYYY-MM-DD.log                       本システムより出力されるログ。
  ├ catalina.YYYY-MM-DD.log          Apache Tomcat 全体のログ。
  ├ localhost.YYYY-MM-DD.log         本システムで発生した例外が出力されるログ。
  ├ localhost_access_log.YYYY-MM-DD.txt 本システムへのアクセスログ。
  └ tomcat7-stdout.YYYY-MM-DD.log             Apache Tomcat 標準エラー出力のログ
```

図 2-7-12 ログファイル構成

システムエラー等が発生した場合は、以下の順序で各ファイルの内容を調査する

① vc4d.YYYY-MM-DD.log

② localhost.YYYY-MM-DD.log

③ catalina.YYYY-MM-DD.log

④ tomcat7-stdout.YYYY-MM-DD.log

５）仮想コンバータ用データフォルダ構成（例）

仮想コンバータ用データフォルダの構成は、本システムの共通設定プロパティファイルで定義する。

「仮想コンバータコンソールアプリケーションパス」「３次元データベースファイル格納フォルダ」「ダウンロード要求ファイル格納フォルダ」「アップロード要求ファイル格納フォルダ」は、任意のパスが指定可能。

システムにアップロードされたファイルは、「ダウンロード要求ファイル格納フォルダ」「アップロード要求ファイル格納フォルダ」の配下に、「年／月／日／受付 ID」フォルダが作成されて保存される。

図 2-7-13 に、データフォルダの構成例を示す。

```
${仮想コンバータ用データフォルダ}
  ├ exe/                                  仮想コンバータコンソール格納フォルダ
  │   └ vc4d.exe                          仮想コンバータコンソールアプリケーション
  │
  ├ data/                                 ３次元データベースファイル格納フォルダ
  │   ├ gis.ldf                           ３次元データ展開データベースファイル
  │   └ gsi.mdf                                 （ファイル名は例）
  │
  └ file/
      ├ download/                         ダウンロード要求ファイル格納フォルダ
      │   └ YYYY/
      │       └ MM/
      │           └ DD/
      │               └ 受付 ID/
      └ upload/                           アップロード要求ファイル格納フォルダ
          └ YYYY/
              └ MM/
                  └ DD/
                      └ 受付 ID/
```

図 2-7-13 データフォルダ構成例

②ソースコード ファイル構成

vc4d-src.zip を、任意のフォルダ下に展開する。

展開後の各フォルダ構成は、下記の通り。

```
vc4d/                                     ソースコードフォルダ
  ├ build.sh                              ビルド用スクリプト
  ├ pom.xml                               Maven プロジェクト記述ファイル
  ├ db/                                   テスト用 DB 定義ファイル
  ├ dummy/                                テスト用 仮想コンバータプログラム
  └ src/
      ├ main/
      │   ├ java/                         Java ソースコード
      │   ├ resources/                    リソースファイル
      │   └ webapps/                      Web 画面ファイル
      │
      └ test/                             ユニットテスト用モジュール SET ANSI_PADDING OFF
```

図 2-7-14 ソースコードファイル構成

１）Java ソースコードファイル一覧

Java ソースコードファイルの構成。

```
vc4d/src/main/java/
  └vc4d/
    ├ core/
    │   ├ dao/                                  Data Access Object パッケージ
    │   │   ├ GenericDao.java
    │   │   ├ JobTaskDao.java
    │   │   ├ ShapeDao.java
    │   │   └ jpa/
    │   │       ├ GenericDaoImpl.java
    │   │       ├ JobTaskDaoImpl.java
    │   │       └ ShapeDaoImpl.java
    │   │
    │   ├ entity/                               データエンティティパッケージ
    │   │   ├ JobTask.java
    │   │   └ Shape.java
    │   │
    │   └ service/                          各サービスモジュールパッケージ
    │       ├ DataService.java
    │       ├ FileService.java
    │       └ impl/
    │           ├ DataServiceImpl.java
    │           └ FileServiceImpl.java
    │
    ├ job/                                  バックグラウンドジョブパッケージ
    │   └ CommandExecutor.java
    │
    └── web/
        ├ controller/                           Web アプリケーションパッケージ
        │   ├ AbstractCookieController.java
        │   ├ DownloadController.java
        │   ├ IndexController.java
        │   ├ StatusController.java
        │   └ UploadController.java
        │
        ├ form/                             Web フォームパッケージ
        │   ├ DownloadForm.java
        │   └ UploadForm.java
        │
        └ listener/                             Web リスナーパッケージ
            └ LocalContextCleanupListener.java
```

図 2-7-15 JAVA ソースコードファイル構成

2）リソースファイル一覧

各リソースファイルの構成。

```
vc4d/src/main/resources/
  ├ vc4d.properties                     共通プロパティファイル
  ├ job-message.properties                      仮想コンバータ用メッセージファイル
  ├ applicationContext.xml                      Springframework 定義 XML
  ├ dataAccessContext.xml                       （同上）
  ├ webContext.xml                  （同上）
  ├ messages-spring_jp.properties               メッセージプロパティファイル
  └ logback.xml                                 ログ出力定義 XML
```

図 2-7-16 リソースファイル構成

3）Web 画面ファイル一覧

Web ブラウザで表示される、各 HTML 画面の構成。

```
vc4d/src/main/webapp/
  ├ WEB-INF/
  │   ├ views/
  │   │   ├ download-index.jsp          ダウンロード受付用 JSP ファイル
  │   │   ├ parts-footer.jsp            （共通）フッタ用 JSP ファイル
  │   │   ├ parts-header.jsp            （共通）ヘッダ用 JSP ファイル
  │   │   ├ parts-menu.jsp                    （共通）メニュー用 JSP ファイル
  │   │   ├ parts-status-detail.jsp           （共通）履歴詳細用 JSP ファイル
  │   │   ├ status-detail.jsp           履歴一覧用 JSP ファイル
  │   │   ├ status-index.jsp            履歴詳細用 JSP ファイル
  │   │   ├ status-search.jsp           履歴検索用 JSP ファイル
  │   │   ├ upload-index.jsp            アップロード受付用 JSP ファイル
  │   │   ├ http-404.jsp                      HTTP ステータスコード 404 用 JSP ファイル
  │   │   └ http-500.jsp                      HTTP ステータスコード 500 用 JSP ファイル
  │   │
  │   └ web.xml                               Web アプリケーション配備記述子 XML
  │
  └ resources/
      └ css/
          └ style.css                   スタイルシートファイル
```

図 2-7-17 Web 画面ファイル構成

4）ソースコードのビルド

ソースコードのビルドは、Apache Maven ビルドツールを用いて行う。

以下、ビルドの手順を説明する。

リスト 2-7-7 ソースコードのビルド手順

```
(1) apache-maven-3.0.4-bin.zip を、任意のフォルダ下に展開する。
(2) 展開後、コマンドプロンプトを開き、ソースコードフォルダへ移動する。
(3) Java SDK をインストールしたフォルダを、環境変数 JAVA_HOME に設定する。
(4) mvn.bat を package オプションを付与して実行し、ソースコードのビルドを行う。
      mvn.bat は Apache Maven を展開したフォルダ下の、bin フォルダに存在する。
(5) ビルドが実行され、ソースコードフォルダ下の target フォルダに vc4d.war が生成される。
```

下の例は、C:¥vc4d-build フォルダ下に Apache Maven とソースコードを展開後、ビルドを行った例である。

リスト 2-7-8 ソースコードのビルドコマンド

```
C:¥Users> cd C:¥vc4d-build¥vc4d                                    ソースフォルダの移動
C:¥vc4d-build¥vc4d> set JAVA_HOME=C:¥Program Files¥Java¥jdk1.7.0   環境変数を設定
C:¥vc4d-build¥vc4d> C:¥vc4d-build¥apache-maven-3.0.4¥bin¥mvn.bat package   ビルド実行
    :
    :   ビルド実行メッセージ
    :
C:¥vc4d-build¥vc4d> dir target¥vc4d.war                            vc4d.war の生成確認
```

（６）操作説明

①各種設定

1）共通設定プロパティファイル

1-1）概要

本システム全般で利用するプロパティファイル。

ファイル名は “vc4d.properties” で固定とする

本ファイルは ${TOMCAT_HOME}/webapps/vc4d/WEB-INF/classes フォルダ下に存在する。

1-2) 設定項目一覧

表 2-7-8 VC-4D システムのプロパティ設定項目一覧

項目名	内容
db.driverClassName	使用する JDBC ドライバのクラス名称。
db.url	接続先データベースの URL。
db.username	データベースのユーザ名。
db.password	データベースのパスワード。
db.platform	データベースのプラットフォーム。
db.showSql	ログに SQL を出力するかを指定。
db.generateDdl	DDL (Data Definition Language)ファイルを出力するかを指定。
file.upload.dir	アップロード受付ファイルを保存するフォルダを指定。 本フォルダ下に「年／月／日／受付 ID」サブフォルダを生成し、各ファイルを格納する。
file.download.dir	ダウンロード受付ファイルを保存するフォルダを指定。 本フォルダ下に「年／月／日／受付 ID」サブフォルダを生成し、各ファイルを格納する。
file.system.db.names	システムデータベース名称を、カンマ区切りで指定。 アップロード処理において、本項目で指定したデータベース名と重複した場合は、${dbname}_data と変更される。
job.command	仮想コンバータプログラムのパス名を指定。
job.db.overwrite	アップロード処理において、データベースを上書きするかを指定。 “false”を設定した場合、データベース名は連番を付与して作成される。
job.command.env.S	仮想コンバータ用データベースのサーバ名称。
job.command.env.U	仮想コンバータ用データベースのユーザ名。
job.command.env.P	仮想コンバータ用データベースのパスワード。
job.command.env.D	仮想コンバータ用データベースファイルの格納フォルダ。
job.cron.expression	仮想コンバータの実行時間を指定。 UNIX cron ベースの書式で、“秒 分 時 日 月 週 年”を指定する。 尚、初期設定値は一分間隔で実行されるよう設定されている。

※ 基本的に**太字**の部分が、設定変更の必要な項目

1-3) 設定例

リスト 2-14 に、共通設定プロパティファイルの設定例を記す。

形式は［項目名］＝［値］となる。

設定する値の中で、“¥”はエスケープ文字と認識されるため、“¥”を設定する場合は“¥¥”と入力する。

（Windows 環境ではファイルパス区切子として、“¥”が使用される）

また、値の中で“${項目名}”とすると変数として認識され、該当する項目の値が展開される。

行頭が“#”の行はコメントとして認識される。

リスト 2-7-9 プロパティファイル vc4d.properties の設定

```
# DBMS settings.
db.driverClassName=net.sourceforge.jtds.jdbc.Driver
db.url=jdbc:jtds:sqlserver://localhost/vc4d
db.username=sa
db.password=password
db.platform=org.hibernate.dialect.SQLServer2008Dialect
db.showSql=false
db.generateDdl=false

# File store settings.
vc4d.dir=C:¥¥vc4d
file.upload.dir=${vc4d.dir}¥¥file¥¥upload
file.download.dir=${vc4d.dir}¥¥file¥¥download
file.system.db.names=master,model,msdb,tempdb,vc4d

# File store settings.
job.command=${vc4d.dir}¥¥exe¥¥vc4d.exe
job.db.overwrite=false
job.command.env.S=(local)¥¥SQLEXPRESS
job.command.env.U=${db.username}
job.command.env.P=${db.password}
job.command.env.D=${vc4d.dir}¥¥data
job.cron.expression=0 */1 * * * ?
```

1-4) ファイルの編集／システムへの反映

テキストエディタを用いて、動作環境に沿う内容で本ファイルを編集する。

ファイル編集後に Apache Tomcat を再起動することで、システムへ反映される。

２）仮想コンバータ用メッセージファイル

2-1) 概要

仮想コンバータの終了コードより、実行結果メッセージを設定するプロパティファイル。

ファイル名は “job-messages.properties” で固定とする

本ファイルは ${TOMCAT_HOME}/webapps/vc4d/WEB-INF/classes フォルダ下に存在する。

2-2) 設定内容一覧

表 2-7-9 VC-4D システムの実行結果メッセージ

項目名	内容
code_0	仮想コンバータ 終了コード 0 に該当するメッセージ。
code_1	仮想コンバータ 終了コード 1 に該当するメッセージ。
code_2	仮想コンバータ 終了コード 2 に該当するメッセージ。
code_3	仮想コンバータ 終了コード 3 に該当するメッセージ。
code_4	仮想コンバータ 終了コード 4 に該当するメッセージ。
code_5	仮想コンバータ 終了コード 5 に該当するメッセージ。
code_6	仮想コンバータ 終了コード 6 に該当するメッセージ。
code_-1	仮想コンバータ実行前にエラーが発生した時のメッセージ。 具体的には、共通設定プロパティファイルの job.command 項目において、仮想コンバータプログラムへのパスが間違っている場合など。
code_-2	仮想コンバータ実行後、result.txt が出力されない時のメッセージ。
extCode_9	result.txt に出力される、exe_ret_code=9 に該当するメッセージ。

2-3) ファイルの編集／システムへの反映

仮想コンバータの終了コードの意味を変更した場合は、本ファイルのメッセージ内容を変更する必要がある。

また、終了コードを増やした場合は、本ファイルに対して “code_（終了コード番号）＝メッセージ内容” の形式で追加を行う。

result.txt 中の exe_ret_code の内容を追加する場合は、“extCode_（exe_ret_code の番号）＝メッセージ内容” の形式とする。

ファイル編集後に Apache Tomcat を再起動することで、システムへ反映される。

②操作画面

１）本システムの URL

任意の Web ブラウザより、以下の URL にアクセスを行う。

http://（インストールサーバのホスト名）/vc4d/

２） ３次元データアップロード画面

画面左側メニューの「アップロード受付」より、３次元データアップロード受付画面が表示される。

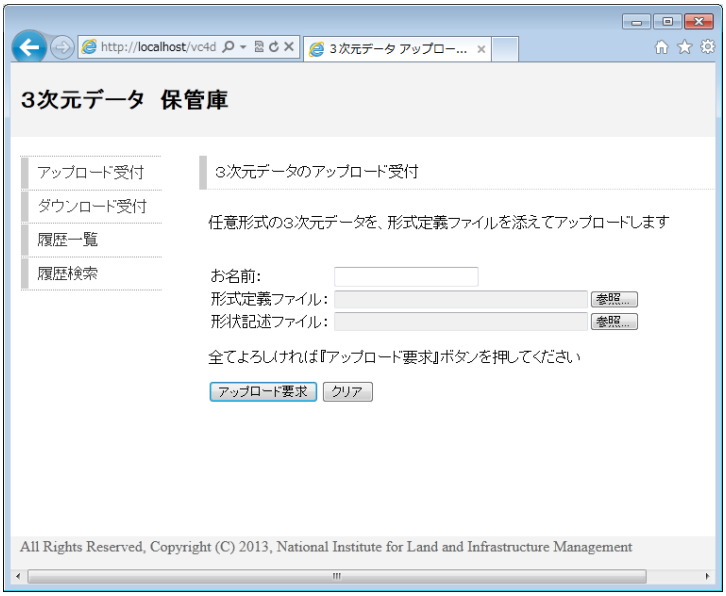


図 2-7-18 アップロード受付画面

受付用フォームに指定する内容を表 2-7-10 に示す。

表 2-7-10 VC-4D アップロード受付フォームの指定内容

項目名	内容
お名前	任意の文字列を入力する。
形式定義ファイル	3次元データの展開に利用する、メタ情報ファイルを指定する。
形状記述ファイル	3次元データの実データファイルを指定する。 仮想コンバータでは、本ファイルの拡張子を除いた名称でデータベースを作成し、3次元データを展開する。

フォームの各項目に文字列／ファイルを指定後、「アップロード要求」ボタンを押すことで、サーバ上に受け付けた内容が登録され、受付 ID が発行される。

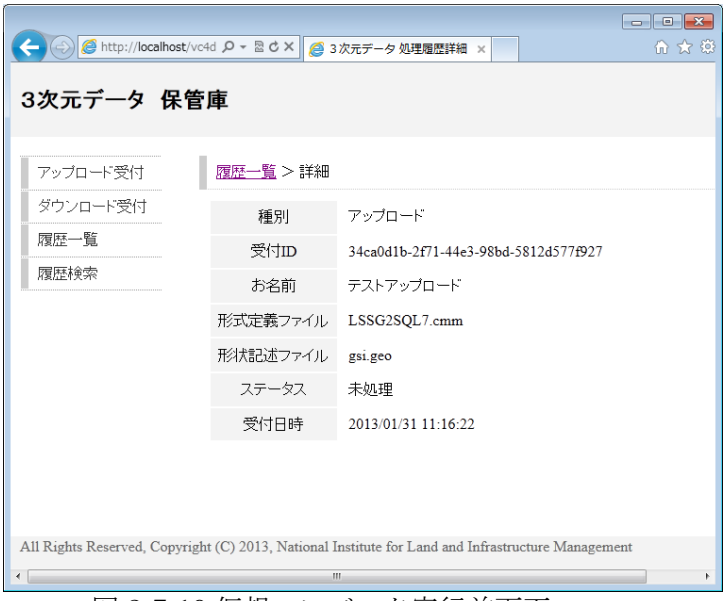


図 2-7-19 仮想コンバータ実行前画面

発行された ID は、後述する履歴検索において用いるため、ユーザは必要に応じてテキストファイル等に保存しておく必要がある。

アップロードを受け付けた後は、履歴詳細画面へ遷移する。

画面中の「ステータス」が“未処理”の場合、仮想コンバータでのデータ展開処理はまだ行われていない。「ステータス」が“未処理”または“処理中”の場合、本画面は１０秒ごとに自動リロードされる。

「ステータス」が“完了”の場合、仮想コンバータでの処理は終了しており、処理結果が表示される。

処理が成功した場合は、「登録データベース名称」に３次元データを展開したデータベース名が表示される。

図 2-7-20 仮想コンバータ実行完了画面

なお「登録データベース名称」について以下の禁則と対応を行っている。

2-1) ３次元データが展開されるデータベースの名称（登録データベース名称）には、受付画面で指定した「形状記述ファイル」のファイル名を使用し、通常は形状記述ファイルのファイル名から拡張子を除いた文字列を登録データベース名称として用いる。

2-2) 共通設定プロパティファイルにおいて、登録データベースの上書きを許可しない（job.db.overwrite=false）設定の場合は、データベース名が重複しないよう枝番号が付与される。

2-3) ファイル名に、データベース作成において使用できない記号が含まれる場合は、記号が全て‘_’に置き換えられる。

2-4) ファイル名の先頭文字が数字で始まる場合は、データベース名の先頭に‘_’が追加される。

登録データベース名称の変換例を表 2-7-11 に示す。

表 2-7-11 VC-4D 登録データベース名称の変換

形状記述ファイル名	登録データベース名称	条件	説明
gis.geo	gis	（通常）	（ファイル拡張子が除かれる）
gis.geo	gis_1	既に“gis”が登録済み	重複しないよう枝番号が付与される
gis-(new).geo	gis_new_	ファイル名に記号が含まれる	記号は‘_’に変換される
123-gis.geo	_123_gis	ファイル名の先頭文字が数字	先頭に‘_’が追加される

3） 3 次元データダウンロード画面

画面左側メニューの「ダウンロード受付」より、3 次元データダウンロード受付画面が表示される。

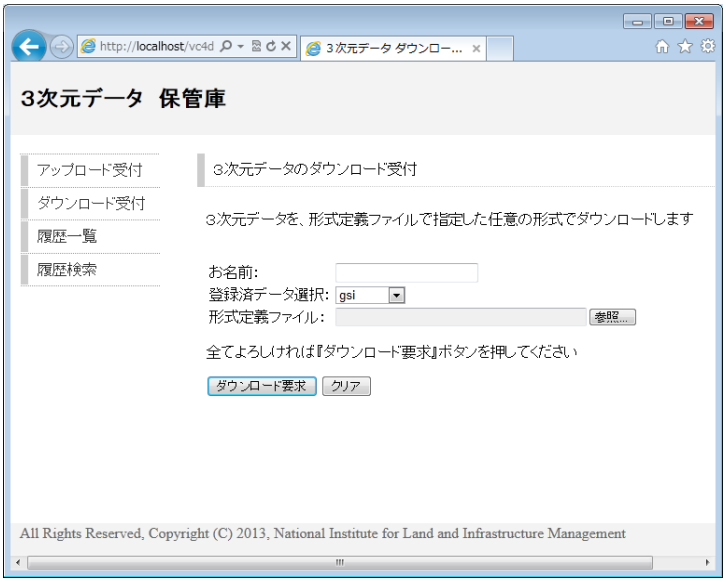


図 2-7-21 ダウンロード受付画面

受付用フォームに指定する内容を表 2-7-12 に示す。

フォームの各項目に文字列／ファイルを指定後、「ダウンロード要求」ボタンを押すことで、サーバ上に受け付けた内容が登録され、受付 ID が発行される。

発行された ID は、後述する履歴検索において用いるため、必要に応じてテキストファイル等に保存しておく。

ダウンロードを受け付けた後は、履歴詳細画面へ遷移する。

画面中の「ステータス」が“未処理”の場合、仮想コンバータでのファイル生成処理は行われていない。

表 2-7-12 VC-4D ダウンロード受付フォームの指定内容

項目名	内容
お名前	任意の文字列を入力する。
登録済データ選択	3次元データが登録されているデータベース名称を指定する。 仮想コンバータで生成する形状記述ファイル名は、ここで指定した値が用いられる。 （拡張子は形式定義ファイルで定義した文字列が付与される）
形式定義ファイル	3次元データのファイル生成に利用するメタ情報ファイルを指定する。

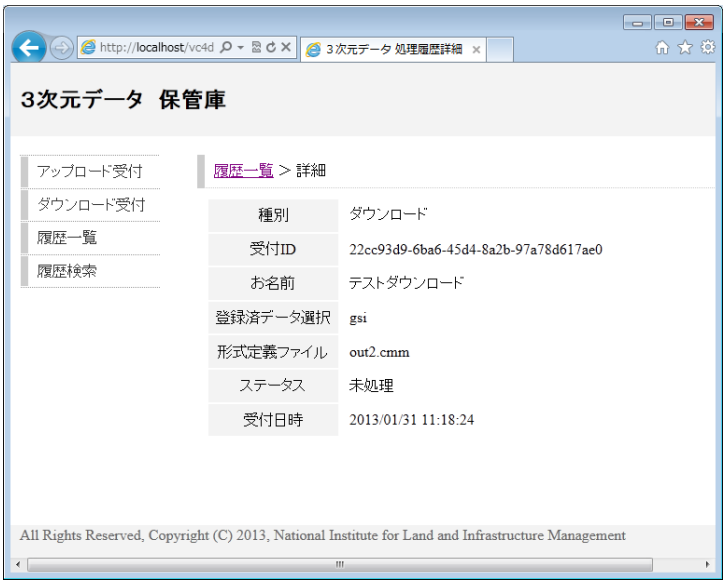


図 2-7-22 仮想コンバータ実行前画面

「ステータス」が“未処理”または“処理中”の場合、本画面は１０秒ごとに自動リロードされる。

「ステータス」が“完了”の場合、仮想コンバータでの処理は終了しており、処理結果が表示される。

処理が成功した場合は、「形状記述ファイル ダウンロード」に生成したファイル名が表示される。

ファイル名のリンクをクリックすることで、ファイルダウンロードが行える。

図 2-7-23 仮想コンバータ実行完了画面

4）履歴一覧画面

画面左側メニューの「履歴一覧」より、現在使用するブラウザよりアップロード／ダウンロード要求を行った履歴の一覧が表示される。

図 2-7-24 履歴一覧画面

履歴一覧中の「受付日時」をクリックすることで、各履歴の詳細画面へ遷移する。

図 2-7-25 履歴詳細画面

アップロード／ダウンロードの履歴は、Web ブラウザの Cookie 情報と紐付けされているため、他のブラウザまたは別環境（ＰＣ）では参照できない。

また同じブラウザであってもユーザが Cookie 情報を消去した場合は、同様に参照することはできない。（この様な場合は履歴検索より参照する必要がある）

5）履歴検索画面

画面左側メニューの「履歴検索」より、過去に登録したアップロード／ダウンロード要求を行った履歴を、受付 ID より検索する画面が表示される。

図 2-7-26 履歴検索画面

フォーム中の「受付 ID」に対して過去登録時に発行された ID 入力し、「検索」をクリックすることで該当する履歴の詳細画面へ遷移する。

図 2-7-27 履歴検索結果表示画面

（７）システムの移行

①現行環境での作業

システム移行時に、現行環境では表 2-7-13 に示した作業を行う。

表 2-7-13 VC-4D システム移行時の現行環境における作業内容

	名称	内容
1	データバックアップ	vc4d データベースおよび３次元データが展開された各データベースのエクスポート、本システムにアップロードされたファイルのバックアップを行う。
2	システムのアンインストール	本システムおよび本システムが利用する各アプリケーションのアンインストールを行う。

1）データバックアップ

1-1) データベースのバックアップ

データベースのバックアップには、SQL Server に付属する“bcp”コマンドを用いる。

“bcp”コマンドでは、各データベースのテーブル単位にファイルへエクスポートを行う。

コマンドプロンプトより、『bcp [データベース名].[スキーマ名].[テーブル名] OUT [出力ファイル名] –c –T』の形式で実行する。リスト 2-7-10 に例を示す。

リスト 2-7-10 vc4d データベースのテーブルデータをエクスポートするコマンド

```
bcp vc4d.dbo.Master OUT vc4d-Master.csv -c  -T
bcp vc4d.dbo.JobTask OUT vc4d-JobTask.csv -c  -T
```

3 次元データが展開された全てのデータベースに対して、各テーブル（COLOR, COORD, FACE, GRUPPE, LINK, NORMAL, TCOORD, VERTEX）のバックアップを実行する。

エクスポートするファイル名が重複しないよう、ファイル名は“[データベース名]-[テーブル名].csv”などとする。

リスト 2-7-11 に 3 次元データベース（=post）のテーブルデータをエクスポートする例を示す。

リスト 2-7-11 vc4d のテーブルデータをエクスポートするコマンド

```
bcp post.dbo.COLOR OUT post-COLOR.csv -c  -T
bcp post.dbo.COORD OUT post-COORD.csv -c  -T
bcp post.dbo.FACE OUT post-FACE.csv -c  -T
bcp post.dbo.GRUPPE OUT post-GRUPPE.csv -c  -T
bcp post.dbo.LINK OUT post-LINK.csv -c  -T
bcp post.dbo.NORMAL OUT post-NORMAL.csv -c  -T
bcp post.dbo.TCOORD OUT post-TCOORD.csv -c  -T
bcp post.dbo.VERTEX OUT post-VERTEX.csv -c  -T
```

1-2) アップロードファイルのバックアップ

本システムにアップロードされたファイルを圧縮してバックアップを行う。

共通設定プロパティファイルに定義した、“file.upload.dir”と“file.download.dir”以下のフォルダを、任意の圧縮／解凍ツールを用いて ZIP ファイルへ圧縮する。

2）システムのアンインストール

「(4)システムのアンインストール(2-84)」の内容に従い、現行環境より本システムおよび各アプリケーションのアンインストールを行う。

vc4d データベースおよび 3 次元データが展開された各データベースは、削除前に必ずデータエクスポートを行うこと。

②移行環境での作業

次に、移行環境で表 2-7-14 に示した作業を行う。

表 2-7-14 VC-4D システム移行時の移行環境における作業内容

	名称	内容
1	システムのインストール	本システムおよび本システムが利用する各アプリケーションのインストールを行う。
2	バックアップデータの復元	vc4d データベースおよび 3 次元データが展開された各データベースの作成／データインポート、本システムにアップロードされたファイルの展開を行う。

1）システムのインストール

「(3)システムのインストール(2-75)」の内容に従い、移行環境に本システムおよび各アプリケーションのインストールを行う。

2）バックアップデータの復元

2-1) データベースのリストア

始めに３次元データを登録するデータベースを作成する。

作成するデータベースの名称は、vc4d データベースの Master テーブルの内容より決定する（現行環境よりエクスポートしたファイルの内容を参照する)。

データベースの作成は、「0．システム用データベース／テーブルの作成」の内容を参考にして、SQL Server Manager Studio より行う。

作成後は､各データベースにテーブル（COLOR, COORD, FACE, GRUPPE, LINK, NORMAL, TCOORD, VERTEX）を作成する。リスト 2-7-12 に例を示す。

リスト 2-7-12　３次元データベース（=post）のテーブルを作成する SQL 文

```
USE [post]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[COLOR](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[R] [float] NULL,
	[G] [float] NULL,
	[B] [float] NULL,
	[A] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[COORD](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[X] [float] NULL,
	[Y] [float] NULL,
	[Z] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[FACE](
	[Gid] [int] NULL,
	[Fid] [int] NULL,
	[idcolor] [int] NULL,
	[idnormal] [int] NULL,
	[idmaterial] [int] NULL,
	[idtexture] [int] NULL,
	[SHP] [int] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[GRUPPE](
	[Gid] [int] NULL,
	[idmaterial] [int] NULL,
	[idtexture] [int] NULL,
	[Attribute] [varchar](max) NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[LINK](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[GPid] [int] NULL,
	[GCid] [int] NULL,
	[mat0] [float] NULL,
	[mat1] [float] NULL,
	[mat2] [float] NULL,
```

```
	[mat3] [float] NULL,
	[mat4] [float] NULL,
	[mat5] [float] NULL,
	[mat6] [float] NULL,
	[mat7] [float] NULL,
	[mat8] [float] NULL,
	[mat9] [float] NULL,
	[mat10] [float] NULL,
	[mat11] [float] NULL,
	[mat12] [float] NULL,
	[mat13] [float] NULL,
	[mat14] [float] NULL,
	[mat15] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[NORMAL](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[X] [float] NULL,
	[Y] [float] NULL,
	[Z] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[TCOORD](
	[id] [int] IDENTITY(1,1) NOT NULL,
	[X] [float] NULL,
	[Y] [float] NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[VERTEX](
	[Gid] [int] NULL,
	[Fid] [int] NULL,
	[Ix] [int] NULL,
	[Icoord] [int] NULL,
	[Inormal] [int] NULL,
	[Icolor] [int] NULL,
	[Itcoord] [int] NULL
) ON [PRIMARY]
GO
```

各テーブルのデータインポートには、SQL Server に付属する“bcp”コマンドを用いる。

コマンドプロンプトより、『bcp [データベース名].[スキーマ名].[テーブル名] IN [入力ファイル名] –c –T』の形式で実行する。

入力ファイルは、現行環境よりエクスポートした各データベースとテーブルに対応するファイルを指定する。リスト 2-7-13、14 にコマンドの例を示す。

リスト 2-7-13　3 次元データベースのテーブルデータをインポートするコマンド

（データの名称：post）

```
bcp post.dbo.COLOR IN post-COLOR.csv -c   -T
bcp post.dbo.COORD IN post-COORD.csv -c   -T
bcp post.dbo.FACE IN post-FACE.csv -c   -T
bcp post.dbo.GRUPPE IN post-GRUPPE.csv -c   -T
bcp post.dbo.LINK IN post-LINK.csv -c   -T
bcp post.dbo.NORMAL IN post-NORMAL.csv -c   -T
bcp post.dbo.TCOORD IN post-TCOORD.csv -c   -T
bcp post.dbo.VERTEX IN post-VERTEX.csv -c   -T
```

リスト 2-7-14 vc4d データベースのテーブルデータをインポートするコマンド

```
bcp vc4d.dbo.Master IN vc4d-Master.csv -c   -T
bcp vc4d.dbo.JobTask IN vc4d-JobTask.csv -c   -T
```

2-2) アップロードファイルの展開

現行環境で作成した ZIP ファイル（アップロードファイルのバックアップ）を、所定のフォルダ下で展開する。

尚、展開先は現行環境と同じフォルダとする。

３．システム開発者の手引き

３－１．はじめに

建物や町並を記録した三次元データに、このデータ形式を解読するためのメタファイルをセットにした保存データを利活用する方法は、上記で例示した携帯端末による現場表示や PC による表示にとどまらずに、「任意形式の三次元データに、解読方法を記述したメタファイルを添付した保存データ」を核とした様々な利活用方法の開発が可能である。

本章では、これまでに作成した４種類の実装形態の開発過程を解説することを通じて、将来新たな処理系を開発しようとするシステム開発者のための解説資料とする。

多様な利活用のシステムに共通する部分は、メタファイルに基づいて、データを入力・解釈するコンバータを生成するコンパイラである。このコンパイラ部は、C 言語で記述されており、PC の OS である Windows のための開発環境 VS2005 と、携帯端末の OS である Android のための開発環境 ndk において、共通であり、このコンパイラのソースコードには互換性がある。

一方、様々な利活用を実現するための実装部分は、使用する機器・OS に適した処理内容を実装する関数を、共にコンパイル・ビルドすることにより作成した。

実際の利活用に当たっては、全てのシステムに共通して、まずメタファイルをコンパイルして実行形式を生成する処理を行う。次にこの実行形式を実行して建物や町並の形状を記述したデータファイルを解読し、様々な利活用の目的に応じた処理を実行する。

従って、新たな利活用方法を開発するためには、コンパイラ部分のソースコードをそのまま利用し、ライブラリ関数の処理内容を利活用目的に応じで書き換えると共に、全体をパッケージ化するためのユーザーインターフェースやマシンコントロールの外皮部分を、ターゲットとするマシンやＯＳ等、目的に応じた開発環境で作成することとなる。

本章では、現在までに試作した４の利活用形態の開発手順を記録することにより、今後の新たな利活用システム開発のための解説資料とする。

表 3-1-1 実装形態別概要

名称	処理概要	ハード	OS	開発環境	外形	ライブラリ
①VC-1C	ファイル出力	PC	Windows	VS2005	コンソール	LIBC
②VC-2V	仮想現実表示	PC	Windows	VS2005	景観シミュレータのプラグイン	MFC
③VC-3M	AR 表示	スマホ、タブレット	Android	sdk+ndk	アプリ	SDK(Java)
④VC-4D	データ保管庫	サーバ	Windows +SQL	VS2005	WEB アプリ	C, Java

３－２．仮想コンバータのコンパイラ・インタプリタ基幹部分の開発

様々な利活用システムに共通となるコンパイラ部分は、林晴比古「コンパイラ・インタプリタ開発」[51]の添付 CD-ROM に収録・公開されたソースコードを出発点とした。この解説書は、各読者がこのソースコードに様々な機能を追加することを前提として提供されていた。そこでまず、出版元であるソフトバンク・クリエイティブ株式会社から、自由に拡張したものをアプリケーションとしてフリーで配布すること、出典を明記した上でソースコードを公開することの許可を得た上で作業に着手した（2012 年 3 月 14 日）。

このコンパイラはソースコードを解釈して、仮想スタックマシンのための機械語を配列上に生成する。次に、実行段階でインタープリタがこの機械語を解釈して処理を実行する。インタープリタも、上記のコンパイラ部分と同様に C 言語で記述されており、ビルド時点でそれぞれの CPU、OS のための実行形式を生成する。従って、原理的には異なる CPU、OS の上への移植を制約する条件がない。

実行形式の機械語は、固定長の配列の上に生成される。また、ポインタ型の変数を記号表アクセスに限定し、機械語がアクセスできるメモリも、固定長の配列の範囲に限定されている。従って、ファイルの読み書きを超えて、実行している仮想スタックマシンの OS のメモリ領域にアクセスして不正な処理を行うようなプログラムは作成不可能である。

また、動的なメモリ管理は行っていない。このことはコンバータを作成する上では一つの制約条件ではあるが、メタファイル（プログラム）が処理するデータファイルが特定されていることから、メタファイル作成段階で、当該データファイルを処理するために十分な長さの固定長の配列を用意することにより、動的なメモリ管理を行う必要はない。

この処理系は、機械語の種類は少なく、高度な最適化等を行っておらず、処理系が単純であるため、大容量のデータを高速処理するような実務には向かないとされているが、しかし長期保存データを解釈するためのメタファイルを記述するための処理系としては、処理速度よりもむしろ単純明快さが求められ、処理は遅くとも確実であることが求められることから、適していると考えた。

（１）数値型の増補と、それに伴う仮想マシンの機械語の増補

数値型としては、最初から提供されていた 32 ビット整数に、32 ビット浮動小数と、256 ビット四元数を加えて、四則演算もこれに合わせて拡充した。まず、建物等を記述する三次元データを処理するためには、32 ビット整数型の変数だけでは不十分であるため、32 ビット浮動小数型も追加した。これは、１ｍｍの精度で、団地や集落のスケールの空間を記述するためには十分な精度である。一方、GPS 座標等、地球の半径に相当するスケールの中で１ｍｍの精度の位置や形状を記述するためには不十分である。このことから、地球的尺度の位置を計算する目的で、256 ビット四元数型を用意した。

メタファイルで使用される変数や関数に関して定義される数値型、printf 関数や scanf 関数の書式文字列の中で使用される変換等の数値型は、ヘッダファイル cci_h の中で DtType として定義されている。これも原著に対して、増補を行った。

[原著]

NON_T 型無し（解析にのみ使用）

VOID_T 種別 Void 宣言「void」（戻り値のない関数の型宣言に使用）

INT_T 種別 Int 宣言「int」

[増補]

FLOAT_T 浮動小数 Float 宣言「float」

QUAT_T　四元数（クォータニオン） Quat 宣言「quat」

STR_T　文字列範囲

MULTI_T　可変型

NON_T は無で、例えば、関数呼び出しの解析において、引数が無い場合、引数の型としてされる。

[例] `x = t sin(  );`

VOID_T は空で、戻り値のない関数の型宣言に用いる。

[例] `void exit(void );`

scanf 関数の書式文字列の解析においては、「%s」,「%[」が検出された場合の書式文字列の型を示すために使用している。但し、関数の void 型の引数として void 型関数が明示的に参照されたような場合には、エラーとしている。

[例] `exit( exit() );`

STR_T は、数値型としては使用しない。

scanf 関数の書式文字列の解析において、%n が検出された場合の型分類に用いている。

MULTI_T は、数値型としては使用しない。

printf, logf 関数の引数として、記号表がポインタ型として参照された場合に解析結果として使用している。C 言語のポインタのような書式で変数の前に「＊」が付された引数が printf 関数等で参照された時には、書式文字列が要求する型の要素が処理のために使用される。具体的には、%s には変数名が、%d には格納アドレスに格納された値が参照される。

メタファイルの中で変数や関数型宣言に用いられる予約語と、トークン要素は、cci_tkn.c で定義された予約語一覧である、KeyWdTbl[]で、トークンを定義する文字列と、それぞれに対応したトークン要素（整数値による ID）の組として定義されており、この中の前半部分で型宣言の文字列と、トークン要素が定義されている。これも本処理系のために増補した。この増補分トークン要素は、cci.h におけるトークン要素の定義に増補した。

[原著]

void トークン要素 Void

int トークン要素 Int

[増補]

float トークン要素 Float

quat トークン要素 Quat

このような数値型の追加を行った上で、数値演算に関する機械語を増補した。オリジナルの機械語と、本処理系のために増補した機械語を、表 3-2-1 に示す。

インタープリタによるこれら機械語の実行処理は、cci_code.c に含まれる execute 関数の中で行われる。それぞれの機械語の処理内容はこの関数の中で直接定義されている。

機械語のプログラムは、命令を示す８ビットのコード、８ビットのフラグ及び処理対象を指示する３２ビットのデータが附属している。データは、処理に用いる値や、メモリ上のアドレスを記述している。処理対象とするアドレスは、大域変数や配列に対応するグローバルアドレスを指す場合と、局所変数を示すスタック上のアドレスを指す場合があり、コンパイラが変数・配列の大域／局所に応じて付与するフラグにより識別されている。グローバル変数の場合 memory 配列上の opData が直接指示する番地であり、ローカル変数の場合、その変数を用いる関数が実行段階で呼び出された時点で定まるベースアドレスからの opData 分のオフセットを加えた番地である。参考として、各コードが実行する処理の概要を付記した。

表 3-2-1 機械語一覧　　* sp の欄はコード実行後のスタックポインタの増減を示す。

コード	処理内容	sp*	参考
①原著の機械語			
DEL	スタックから削除	-1	–stp
STOP	OS 戻り	0/-1	if(stp>0) return POP(); else return 0;
JMP	ジャンプ	0	Pc=opData
JPT	真ジャンプ	-1	if(stack[stp--]) Pc = opData
LIB	組込関数（関数種別） （ライブラリ関数含む）	別表	library(opData) （表 3-2-2 参照）
LOD	メモリの値を PUSH	+1	PUSH(IntMem(adrs))
LDA	アドレスを PUSH	+1	PUSH(adrs)
LDI	値を PUSH	+1	PUSH(opData)
STO	スタックのトップを メモリに格納	-1	ASSIGN(adrs、stack[stp]; --stp;
ADBR	フレーム確保	0	baseReg += opData
NOP	何もしない	0	
ASS	スタックのトップを、 スタックのセカンドの アドレスに格納	-2	ASSIGN(stack[stp-1],stack[stp]); stp -= 2; 整数型の変数、配列に値を書き込む
ASSV	スタックのトップを、 スタックのセカンドに 格納し値をトップに残す	-1	ASSIGN(stack[stp-1],stack[stp]); stp--; 代入式(変数=式)の値が更に参照される場合。参照されない場合には ASS に変更。

VAL	スタックのトップのアドレスの値をスタックのトップに	0	stack[stp] = IntMem(stack[stp]); 整数型の変数、配列の値を取得するために使用
EQCMP	スタックトップと値を比較し結果をトップに(switch 用比較)	0	if(opData==stack[stp]) stack[stp]=1; else PUSH(0);
CALL	戻り番地を保存してジャンプ	+1	PUSH(Pc); Pc = opData;
RET	戻り番地の復活	-1	Pc = POP();
INC	トップが示す変数に１を足し結果をトップに	0	stack[stp]++; stack[stp]=IntMem(stack[stp]);
DEC	トップが示す変数から１を減じ結果をトップに	0	stack[stp]--; stack[stp]=IntMem(stack[stp]);
NOT	トップを否定	0	stack[stp] = ! stack[stp];
NEG	トップを符号反転	0	stack[stp] = - stack[stp];
DIV	整数除算	-1	stack[stp-1] =stack[stp-1]/stack[stp]; stp--;
MOD	整数剰余	-1	stack[stp-1] =stack[stp-1]%stack[stp]; stp--;
ADD	整数加算	-1	stack[stp-1] =stack[stp-1] + stack[stp]; stp--;
SUB	整数減算	-1	stack[stp-1] =stack[stp-1] - stack[stp]; stp--;
MUL	整数乗算	-1	stack[stp-1] =stack[stp-1] * stack[stp]; stp--;
LESS	整数未満	-1	stack[stp-1] =stack[stp-1] < stack[stp]; stp--;
LSEQ	整数以下	-1	stack[stp-1] =stack[stp-1] <= stack[stp]; stp--;
GRT	整数超	-1	stack[stp-1] =stack[stp-1] > stack[stp]; stp--;
GTEQ	整数以上	-1	stack[stp-1] =stack[stp-1]>=stack[stp]; stp--;
EQU	整数一致	-1	stack[stp-1] =stack[stp-1]==stack[stp]; stp--;

NTEQ	整数不等	-1	stack[stp-1] =stack[stp-1]!=stack[stp]; stp--;
AND	整数論理和	-1	stack[stp-1] =stack[stp-1]&&stack[stp]; stp--;
OR	整数論理積	-1	stack[stp-1] =stack[stp-1]||stack[stp]; stp--;
②本処理系のための機械語命令追加分			
F2I	トップが指す浮動小数変数を整数に変換	0	stack[stp]= (int)(*((float*)(&stack[stp])));
I2F	トップが指す整数変数を浮動小数に変換	0	*(float*)&stack[stp]= (float)stack[stp];
I1F	セカンドが指す整数変数を浮動小数に変換	0	*(float*)&stack[stp-1]= (float)stack[stp-1];
INCF	浮動小数に 1.0 を加算	0	FloatMem(stack[stp])+=1.0f; stack[stp]=IntMem(stack[stp]);
DECF	浮動小数に 1.0 を減算	0	FloatMem(stack[stp])-=1.0f; stack[stp]=IntMem(stack[stp]);
NEGF	浮動小数符号反転	0	UNI_OPF(-);
DIVF	浮動小数除算	-1	ZERO_CHKF(); FLOAT_OP(/); *(float*)&stack[stp-1] = *(float*)(&stack[stp-1])/ *(float*)(&stack[stp]),--stp;
ADDF	浮動小数加算	-1	FLOAT_OP(+);
SUBF	浮動小数減算	-1	FLOAT_OP(-);
MULF	浮動小数乗算	-1	FLOAT_OP(*);
LESSF	浮動小数未満	-1	FLOAT2I_OP(<);
LSEQF	浮動小数以下	-1	FLOAT2I_OP(<=);
GRTF	浮動小数超過	-1	FLOAT2I_OP(>);
GTEQF	浮動小数以上	-1	FLOAT2I_OP(>=);
EQUF	浮動小数一致	-1	FLOAT2I_OP(==);
NTEQF	浮動小数不一致	-1	FLOAT2I_OP(!=);
ANDF	浮動小数 AND	-1	FLOAT2I_OP(&&);
ORF	浮動小数 OR	-1	FLOAT2I_OP(||);
PI	円周率	+1	*(float*)&stack[++stp] = 3.14159208f;

SIN	正弦	0	*(float*)&stack[stp] = (float)sin(*(float*)(stack+stp));
COS	余弦	0	*(float*)&stack[stp] = (float)cos(*(float*)(stack+stp));
TAN	正接	0	*(float*)&stack[stp] = (float)tan(*(float*)(stack+stp));
ATAN	逆正接	0	*(float*)&stack[stp] = (float)atan(*(float*)(stack+stp));
SIND	正弦、度表示	0	#define PI180 (3.14159208f/180.0f) *(float*)&stack[stp] = (float)sin(*(float*)(stack+stp));
COSD	余弦、度表示	0	*(float*)&stack[stp] = (float)cos((*(float*)(stack+stp)) *PI180);
TAND	正接、度表示	0	*(float*)&stack[stp] = (float)tan((*(float*)(stack+stp)) *PI180);
ATAND	逆正接、度表示	0	*(float*)&stack[stp] = (float)atan((*(float*)(stack+stp)) *PI180);
SQRT	平方根	0	*(float*)&stack[stp] = (float)sqrt(*(float*)(stack+stp));
EXPQ	四元数指数	0	expq((double*)&stack[stp-7]);
LNQ	四元数対数	0	lnq((double*)&stack[stp-7]);
CONCAVE	凹ポリゴンフラグ変更		setconcave(opData);

ASSQ	四元数型の変数・配列への書込。	-9	stp-=7;
			(quat)(&memory[stack[stp-1]])=
			(quat)(&stack[stp]);
			stp-=2;

ASSQV	四元数型の変数・配列へ書込み、値をスタックに残す	-1	quat *q;
			q = (quat*)(&memory[stack[stp-8]]);
			*q = *(quat*)(&stack[stp-7]);
			(quat)(&stack[stp-8]) = *q;
			stp--;

LODQ	四元数変数読出	+8	*(quat*)(&stack[++stp]) = *(quat*)(memory+adrs); stp+=7;
STOQ	四元数変数書込(引数からローカル変数へ)	-8	stp-=7; *(quat*)(memory+adrs)= *(quat*)(&stack[stp]);

			stp--;
VALQ	四元数変数読出	+7	*(quat*)(&stack[stp])= *(quat*)(memory+stack[stp]); stp+=7;

ADDQ	四元数加算	-8	quat *q1,*q2;
			q1 = (quat*)(&stack[stp-15]);
			q2 = (quat*)(&stack[stp-7]);
			q1->t += q2->t;
			q1->x += q2->x;
			q1->y += q2->y;
			q1->z += q2->z;
			stp -= 8;

I2Q	整数を四元数に変換	0	何もしないこととなった

SUBQ	四元数減算	-8	quat *q1,*q2;
			q1 = (quat*)(&stack[stp-15]);
			q2 = (quat*)(&stack[stp-7]);
			q1->t -= q2->t;
			q1->x -= q2->x;
			q1->y -= q2->y;
			q1->z -= q2->z;
			stp -= 8;

MULQ	四元数乗算	-8	quat *p,*q,r;
			p = (quat*)(&stack[stp-15]);
			q = (quat*)(&stack[stp-7]);
			r.t = p->t*q->t - p->x*q->x - p->y*q->y - p->z*q->z;
			r.x = p->t*q->x + p->x*q->t + p->y*q->z - p->z*q->y;
			r.y = p->t*q->y + p->y*q->t + p->z*q->x - p->x*q->z;
			r.z = p->t*q->z + p->z*q->t + p->x*q->y - p->y*q->x;
			*p = r;
			stp -= 8;

DIVQ	四元数除算	-8	quat *p,*q,r;
			double l;
			p = (quat*)(&stack[stp-15]);
			q = (quat*)(&stack[stp-7]);
			r.t = p->t*q->t + p->x*q->x + p->y*q->y + p->z*q->z;
			r.x = -p->t*q->x + p->x*q->t - p->y*q->z + p->z*q->y;
			r.y = -p->t*q->y + p->y*q->t - p->z*q->x + p->x*q->z;
			r.z = -p->t*q->z + p->z*q->t - p->x*q->y + p->y*q->x;
			l = q->t*q->t + q->x*q->x + q->y*q->y + q->z*q->z;
			p->t = r.t/l;
			p->x = r.x/l;
			p->y = r.y/l;
			p->z = r.z/l;
			stp -= 8;

BCH	配列の添字の範囲検査	0	stack[stp]<0 なら実行エラー「配列の添字が負」 opData<=stack[stp]（配列宣言時の長さ）なら実行エラー「配列の添字が過大」
DELQ	スタックデータ削除	-8	quat 型の式の値が使用されない場合、削除する。DEL（整数）の処理を数値型に合わせて多様化する。
TVAL	記号表の変数格納域	0	stack[stp]を添字とする記号表に登録された変数の格納域を stack[stp]に置く
TNAME	記号表の変数名	0	stack[stp]を添字とする記号表に登録された変数の名称を stack[stp]に置く

（２）入出力に用いる組込関数の増補

上記文献で提供されているソースコードは、入力手段としてコンソールから数値を取得

する input 関数と、コンソールに文字列を出力する printf 関数しかもたない。そこで、この入出力機能に、コンバータの記述に適した入出力関数（ライブラリ関数、ないしライブラリ関数）を増補することにより、データファイルの読み取りや、処理結果の出力が簡単に記述できるような言語となる。このような組込関数は、機械語の LIB 命令に、データとして関数タイプを示す数値を付して定義されており、実行は上記の execute 関数の中で LIB 命令が検出された段階で、cci_code.c の中で定義された library 関数に関数タイプを引数で渡して行われる。それぞれのライブラリ関数に引数として渡す数値は、関数毎にあらかじめ決められた長さのスタックを用いて受け渡す。

library 関数から、関数タイプに従って分岐して実行されるそれぞれの組込関数の実体は、引数と戻り値を持たず、スタックから引数情報を受け取り、結果をスタックに積んで終了する関数群であり、処理系により処理内容は異なり、ビルドに利活用目的に応じたソースコードを含めることにより実装されている。

例えば、メタファイルにおいて法線ベクトルを定義する NORMAL(x,y,z);というコマンドが要求されると、コンパイラは x,y,z の値を評価した結果をスタックに積んだ上で、「LIB NORMAL_F」という機械語命令を生成する。実行段階でインタープリタの execute 関数は、「LIB NORMAL_F」という機械語命令の処理に当たって、NORMAL_F を引数として組込関数を呼び出す：library(NORMAL_F);

library 関数は、引数として与えられた関数タイプに従って分岐し、NORMAL_F が引数である場合には、NORMAL()関数を呼び出す。

VC-1C を実装している cci_ip.c においては、この NORMAL 関数は、スタックから３の法線ベクトル値 X,Y,Z を取り出し、これをソートした上で ID 番号を取得し、これを用いて、

printf(fp, “N%d=NORMAL(%f,%f,%f);¥n”,ID,X,Y,Z);

という１行を実行し、引数式を評価した値を用いて出力ファイルに１行出力すると共に、この ID 値を、スタックを通じて返す。

VC-2V、VC-3M を実装している cci_dml.c においては、同じ NORMAL 関数が、スタックから３の法線ベクトル値 X,Y,Z を取り出し、これを表示に使用するメモリ上の配列に格納した上で、その ID（配列の添字）を、スタックを通じて返す。

VC-4D を実装している cci_sql.c においては、同じ NORMAL 関数が、スタックから３の法線ベクトル値 X,Y,Z を取り出してＳＱＬ文を生成して実行し、データベース上の normal テーブルから同一の法線ベクトル値を登録したレコードがあるかどうか検索し、あればその ID を、また無ければ新規登録した上でその新たな ID を、スタックを通じて返す。

各実装形態に共通して生成されるスタックトップの ID 値は、メタファイルの記述における次の処理（例えば、変数への値の代入）に使用される。

上記の各処理系の処理内容は、ファイル出力、メモリ上の配列へのデータ登録、SQL データベースへの登録と、それぞれ異なっている。しかしいずれの NORMAL 関数もスタックから３の引数値を取得し、各様の処理を行った結果をスタックのトップで返している。

現在までに試作した４の利活用形態に共通するコンパイラ・インタプリタ処理系のために増補した組込関数を表 3-2-2 に示す。処理系毎に実行内容が異なるものをライブラリ関数と呼んで区別し、ここではテンプレート（引数構成、戻り値、処理内容）を示す。

関数タイプの欄は、処理系内部で各関数に共通の機械語 LIB を呼び出す際に、関数タイプを識別するために opData として添付される ID であり、cci.h の中で宣言されている。

概要の欄には組込関数の処理内容を簡単に説明した。

sp の欄は、組込関数の呼び出し前と後でのスタックポインタの増減を示した。例えば-4+8であれば引数として４取得し結果を８残す（結果的に sp は４増加する）。

メタファイル中で使用する関数と用例の欄には、この組込関数を起動するためのメタファイル中での記法を解説した。メタファイルの中で使用されるそれぞれの組込関数の名称(トークン)は、cci_tkn.c に宣言されている。組込関数の戻り値の数値型は、cci_pars.c の factor 関数において switch~case で識別し、グループ別にプログラムで直接指定している。組込関数の引数の数と数値型は、cci_pars.c の sys_fncCall 関数の中でその整合性をチェックしている。

メタファイルの中で組込関数に対して指定された引数は、それぞれを式として評価・計算する機械語に展開された上で、最終的にはスタックを通じ個別の組込関数の処理へと渡され、結果（戻り値）はスタックを介して返される。引数の型の内、変数（var）と示したものは、メタファイル作成者がメタファイルの中で定義する関数にはない、組込関数の引数にのみ使用される数値型であり、メタファイルの中で組込関数の呼び出しにおいては変数名または配列の１要素が直接記述される。これは C 言語の関数の引数において、「＆変数」の形式で参照されるものに相当する。

メタファイルの中で組込関数が使用された場合には、コンパイラの文法解析段階では、組込関数は、固有のトークンとして認識され、これに対応する関数タイプを opCode として随伴する機械語の LIB 命令を生成する。実行段階で機械語 LIB が実行されると、opCode を引数として library 関数が呼び出される。その中で、opCode 毎の処理が行われる。

様々な処理系に共通の簡単な処理は library 関数の中に直接記述されている。一方、図形定義などを記述する組込関数を特に「ライブラリ関数」と呼び、利活用処理系により異なる実装を定義することができる。この実際に処理を行う各関数では、スタックから引数情報を受け取り、処理結果が void でなければ、数値型に応じたサイズでスタックのトップに結果を積んで終了する。この呼び出しと実行の手順だけは関数テンプレートとして定めているが、長期保存されたデータとメタファイルを用いて、本稿で例示した cci_ip.c（ファイル出力）、 cci_dml.c（仮想現実表示）、及び cci_sql.c（データベース構築）とは異なるライブラリ関数の様々な実装形態を、将来における新たな利活用処理系のニーズや開発環境に応じて作成することにより実現することを想定した。プログラマのための参考とする。

個別のライブラリ関数が果たすべき役割とメタファイルにおける用法の詳細に関しては、資料４－２で解説する。

表 3-2-2 組込関数一覧　　* sp の欄はコード実行後のスタックポインタの増減を示す

関数タイプ	概要	sp*	メタファイル中で使用する関数と使用例
原著に含まれている組込関数			
EXIT	プログラムの強制終了	-1+0	exit(int); execute 関数の中で関数タイプが EXIT であった場合、library 関数を呼び出すことなく、直ちに処理を終了する。
INPUT_F	コンソール入力	+1	仮想コンバータでは使用しない
PRINTF1_F	引数のない出力	-1+0	printf(“文字列”);
PRINTF2_F	引数のある出力	-2+0	printf(“書式文字列”,式);
仮想コンバータで追加した組込関数（データファイルへのアクセスと、ログ出力）			
SCANF_F	ファイル入力	-1+1	int scanf(“文字列”); int scanf(“文字列”,var);
LEN_F	ファイルサイズ取得	-0+1	int LEN();
SIORI_F	栞位置を返す	-0+1	int SIORI();
SEEK_F	栞位置を設定する	-1+0	SEEK(int);
GETC_F	１文字取得	-0+1	int GETC();
GETINT_F	整数値の取得	-1+1	int GETINT(int);
GETS_F	行長さの取得	-0+1	int GETS();
LOGF1_F	引数のないログ出力	-1+0	logf(“文字列”);
LOGF2_F	引数のあるログ出力	-2+0	logf(“書式文字列”,式); (式の数値型は書式文字列に従う)
仮想コンバータで追加した組込関数（座標系に関する数値関数）			
_Q_F	四元数の構築	-8+8	quat _Q(float,float,float,float); (引数が int 型の場合、float 値に変換)
_QS_F	文字列による四元数の定義	-1+8	quat _QS(“文字列”); [例]q = _QS(“0.12, 0.23, 0.43, 0.76”);
_Qt_F	四元数 T 成分取得	-8+1	float _Qt(quat);
_Qx_F	四元数 X 成分取得	-8+1	float _Qx(quat);
_Qy_F	四元数 Y 成分取得	-8+1	float _Qy(quat);
_Qz_F	四元数 Z 成分取得	-8+1	float _Qz(quat);
IKE2DES_F	緯度経度→地球座標	-8+8	quat = IKE2DES(quat);
DES2IKE_F	地球座標→緯度経度	-8+8	quat = DES2IKE(quat);
IKE2NAT_F	緯度経度→国家座標	-8+8	quat = IKE2NAT(int, quat);
NAT2IKE_F	国家座標→緯度経度	-9+8	quat = NAT2IKE(int, quat);

IKE2WLD_F	緯度経度→世界座標	-9+8	quat = IKE2WLD(int, quat);
WLD2IKE_F	世界座標→緯度経度	-9+8	quat = WLD2IKE(int, quat);
仮想コンバータで追加したライブラリ関数（LSS-G 系コマンド、三次元形状の定義）			
TEXTURE_F	テクスチャ定義	-1+1	int TEXTURE(“文字列”);
MATERIAL_F	マテリアル定義	-1+1	int MATERIAL(“文字列”);
COORD_F	座標定義	-3+1	int COORD(float,float,float); (引数が int 型の場合、float 値に変換)
COORDQ_F	座標定義	-8+1	int COORD(quat);
VERTEX_F	頂点定義	-5+1	int VERTEX(var,var,var,var,var); 座標,法線,テクスチャ座標,カラー,座標 (定義しない var は省略可能)[註]
TCOORD_F	テクスチャ座標定義	-2+1	int TCOORD(float,float);
NORMAL_F	法線ベクトル定義	-3+1	int NORMAL(float,float,float); （引数は順に X 成分、Y 成分、Z 成分）
COLOR3_F	カラー定義	-3+1	int COLOR(float,float,float); （引数は順に R、G、B 値とし 0.0～１.0）
COLOR4_F	カラー定義(α値付)	-4+1	int COLOR(float ,float, float, float); （引数は順に R､G､B､α 値とし 0.0～１.0）
FACE_F	頂点列により面を定義	-X+1	int FACE(var,var,……); ライブラリ関数の呼び出しに際しては全ての引数をスタックにプッシュした上で総数をトップに置く。
GROUP0_F	グループ宣言	-0+1	int GROUP();
GROUP_F	属性付グループ宣言	-1+1	int GROUP(“属性文字列”);
GROUP_FACE_F	グループに面を追加	-X+0	GROUP_FACE(var,var,….);
GROUP_LINE_F	グループに線を追加	-X+0	GROUP_LINE(var,var,….);
FACE_NORMAL_F	面に法線を定義	-2+0	FACE_NORMAL(var,var);
FACE_COLOR_F	面に色彩を定義	-2+0	FACE_COLOR(var,var);
FACE_TEXTURE_F	面にテクスチャを定義	-2+0	FACE_TEXTURE(var,var);
FACE_MATERIAL_F	面にマテリアルを定義	-2+0	FACE_MATERIAL(var,var);
LINE_F	頂点列により線（折れ線）を定義	-X+1	int LINE(var,var,….);
LINE_NORMAL_F	線に法線を定義	-2+0	LINE_NORMAL(var,var);
LINE_COLOR_F	線に色彩を定義	-2+0	LINE_COLOR(var,var);
LINE_TEXTURE_F	線にテクスチャを定	-2+0	LINE_TEXTURE(var,var);

	義		
LINE_MATERIAL_F	線にマテリアルを定義	-2+0	LINE_MATERIAL(var,var);
GROUP_TEXTURE_F	グループにテクスチャを定義	-2+0	GROUP_TEXTURE(var,var);
GROUP_MATERIAL_F	グループにマテリアルを定義	-2+0	GROUP_MATERIAL(var,var);
LINK_F	親グループと子グループの間のリンクを定義	-2+1	int LINK(var,var);
LINK_XFORM_F	リンクの相対的位置関係を定義	-X+0	LINK_XFORM(var,LOAD/PRE/POST, IDENTITY); TRANSLATE,var,var,var); ROTATE_X/_Y/_Z,var); ROTATE_A,var,var,var,var); MATRIX,var,var,var,var, var,var,var,var, var,var,var,var, var,var,var,var);
FACE_VERTEX_F	定義済の面に頂点を追加	-2+0	FACE_VERTEX(var,var); (引数は面と頂点。LSS-G にない形式)
LINE_VERTEX_F	定義済の線に頂点を追加	-2+0	LINE_VERTEX(var,var); (引数は線と頂点。LSS-G にない形式)
GROUP_ATTR_F	グループにデータファイル中の文字列を属性として追加	-3+0	GROUP_ATTRIBUTE(var,int,int); (引数は、グループ id, 先頭アドレス、文字数。LSS-G にない形式)
仮想コンバータで追加したライブラリ関数（LSS-S 系コマンド、時刻、視点等の記録）			
TIME_F	時間を定義する		int TIME(float); （単位：日）

CAMERA_F	カメラアングルを定義する	-13+1	int CAMERA(float,float,float, //視点
			float,float,float, //注視点
			float,float,float, //上方ベクトル
			float, //視野角
			float, //アスペクト比
			float, float); //zNear, zFar
			[例]

			4.59636449813843, 7.70967817306519, 4.25015115737915, 4.48174047470093, 6.71801471710205, 4.19128608703613, -0.00861490704119205, -0.0582613088190556, 0.998264610767365, 35.8967247009277, 1.77777802944183, 0.100000023841858, 317.575653076172);
LIGHT_F	光源を定義する	-8+1	int LIGHT(int, float, float, float, //位置 int, float, float, float); //色彩
LIGHTGROUP_F	光源グループを定義する	-X+1	int LIGHTGROUP(var,var,…); (引数は光源を定義した変数とし最大８)
MODEL_F	モデルを定義する	-1+1	int MODEL(“メタファイル名?データファイル名”);
IMAGE_F	画像を定義する	-1+1	int IMAGE(“背景画像ファイル名”);
EFFECT_F	効果を定義する	-5+1	int EFFECT(IKE,"0.060198","9.449400","0.000000","/storage/emulated/0/VirtualConverter/meta/LSSG.cmm?/storage/emulated/0/VirtualConverter/data/hamakaze6t.geo")
EFFECTGROUP_F	効果グループを定義する	-X+1	int EFFECTGROUP(var,var,…); （引数は効果とする）
SCENE_F	シーンを定義する	-0+1	int SCENE();
仮想コンバータで追加したライブラリ関数（制御に関する特殊関数）			
(トークン処理)	コメント制御	-	void COMMENT(ON または OFF);
(トークン処理)	凹多面体処理制御	-	void CONCAVE(ON または OFF);
仮想コンバータで追加したライブラリ関数（出力のための形状要素取得）			
G_F	グループを順次選択	-0+1	int G();
Gid_F	選択グループの ID を取得	-0+1	int Gid();
Gattribute_F	選択グループの属性を取得	-0+1	int Gattribute();
Gmaterial_F	選択グループのマテリアルを取得	-0+1	int Gmaterial();
Gtexture_F	選択グループのテク	-0+1	int Gtexture();

	スチャを取得		
F_F	選択グループの面を順次選択	-0+1	int F();
Fshp_F	選択面のシェープを取得	-0+1	int Fshp();
Fnormal_F	選択面の法線を取得	-0+1	int Fnormal();
Fcolor_F	選択面の色を取得	-0+1	int Fcolor();
Fmaterial_F	選択面のマテリアルを取得	-0+1	int Fmaterial();
Ftexture_F	選択面のテクスチャを取得	-0+1	int Ftexture();
F3_F	面の三角形分割し順次取得	-0+1	int Ft();
V_F	選択面の頂点を順次選択	-0+1	int V();
Vcoord_F	選択頂点の座標を取得	-0+1	int Vcoord();
Vcolor_F	選択頂点の色彩を取得	-0+1	int Vcolor();
Vnormal_F	選択頂点の法線を取得	-0+1	int Vnormal();
Vtcoord_F	選択頂点の座標を取得	-0+1	int Vtcoord();
Vvirtual_F	選択頂点の仮想線フラグを取得	-0+1	int Vvirtual();
S_F	全ての頂点座標を順次選択	-0+1	int S();
Sx_F	選択座標の X 値を取得	-0+1	float Sx();
Sy_F	選択座標の Y 値を取得	-0+1	float Sy();
Sz_F	選択座標の Z 値を取得	-0+1	float Sz();
Sq_F	選択座標を四元数として取得	-0+1	quat Sq();

N_F	全ての法線を順次選択	-0+1	int N();
Nx_F	選択法線の X 値を取得	-0+1	float Nx();
Ny_F	選択法線の Y 値を取得	-0+1	float Ny();
Nz_F	選択法線の Z 値を取得	-0+1	float Nz();
Nq_F	選択法線を四元数として取得	-0+1	quat Nq();
C_F	全ての色を順次選択	-0+1	int C();
Cr_F	選択色の R 値を取得	-0+1	float Cr();
Cg_F	選択色の G 値を取得	-0+1	float Cg();
Cb_F	選択色の B 値を取得	-0+1	float Cb();
Ca_F	選択色のα値を取得	-0+1	float Ca();
Cq_F	選択所を四元数で取得	-0+1	quat Cq();
T_F	全てのテクスチャ座標を順次選択	-0+1	int T();
Tx_F	テクスチャの X 座標を取得	-0+1	float Tx();
Ty_F	テクスチャの Y 座標を取得	-0+1	float Ty();
Tq_F	テクスチャ座標を四元数で取得	-0+1	quat Tq();
L_F	全てのリンクを順次選択	-0+1	int L();
Lp_F	親グループを取得	-0+1	int Lp();
Lc_F	子グループを取得	-0+1	int Lc();
M_F	マトリクス読出	-0+1	float Lm();
Mi_F	単位マトリクスか？	-0+1	int Li();
Lt_F	リンクの並進成分	-0+1	quat Lt();
Lr_F	リンクの回転成分	-0+1	quat Lr();
Ls_F	リンクの拡大成分	-0+1	quat Ls();
D_F	全ての表示グループ	-0+1	int D();

	を順次選択		
Dl_F	選択した表示の階層を取得	-0+1	int Dlevel();
Df_F	選択した表示の実体が初出なら 1	-0+1	int Dfirst();

組込関数が数値型を有する場合には、演算結果がスタックのトップに残される。入出力処理(printf など)において組込関数の戻り値を参照する必要がない場合には、コンパイル段階で、statement として扱い、実行時に組込関数がスタックトップに残す戻り値を削除する機械語を追加して、スタックの一貫性を維持している。このような場合、組込関数の先に式が続くような場合、コンパイル・エラーとして処理している。

[例] printf(“first”) + 3;

数式(expression)も一般的には、別の変数に代入が可能である。例えば、a=b=c; は、式「b=c」の値（c に等しい）を a に代入する。この式全体の値は、更に代入する先がない場合には、to_leftVal 関数による機械語の変換が行われている。

関数の戻り値が参照されなければ、処理が意味を持たないような組込関数が文頭に記述された場合に関しては、statement 関数で、エラーとして処理する。

[例] sin(0.0); ・・・代入先や参照がない

（3）メタファイルの文法における標準 C 言語との違い

三次元形状を記述するデータファイルの解読方法を指示するメタファイルを記述するに当たって、以下のように関数定義の方法などを C 言語の標準的な書法から変更した。

①main 関数の省略

通常の場合プログラムは、main 関数を起点として実行する。main 関数から、他の関数を呼び出すことができ、その戻り値を用いて続く処理を実行することができる。他の関数はプログラムの中で定義することができるが、ライブラリ関数と同じ名称の関数を定義しようとするとエラーとしている。また、他の関数の中であっても、exit 命令が実行されると、main 関数に戻ることなく処理を終了する。

main 関数から呼び出す他の関数がない場合には、メタファイルにおいて main 関数の宣言を省略することができることとした。言い換えると、直ちにコマンド列が始まるようなプログラムは、main(){}の宣言が省略された、{} の内部だけの表現として解釈する。

これにより、LSS－G 形式のファイルを、この処理系の開発初期におけるテスト用メタファイル（既知の固定形状を記述）として利用することができるようになった。

②引数の数が可変のライブラリ関数

例えば、

F=FACE (V1, V2, V3);

は、3 の頂点 V1,V2,V3 を頂点とする三角形を定義するコマンドであるが、4 頂点以上の

場合には、引数は４以上となり、その上限はない。このように、引数の数が不定の関数もライブラリ関数として増補できるようにするため、処理系の修正を行った。

なお、引数の数が不定の関数は、一部のライブラリ関数のみであって、メタファイルの中で定義される関数をこのような関数として定義することは許容していない。

③引数を省略できるライブラリ関数

例えば頂点を定義する VERTEX 関数は、全ての引数を使用する場合、

V=VERTEX(S, N, T, C, Sv);

（但し、S は座標、N は法線ベクトル、C は色彩、T はテクスチャ座標,Sv は S が仮想線の起点の座標である場合の、終点の座標）

という書法となるが、法線ベクトル、色彩等を明示的に定義しない場合に省略することができ、

V=VERTEX(S…Sv); （N,C,T を省略）

V=VERTEX(S); （N,C,T,Sv を省略）

等と簡略に記述することができる。引数リストの二つの「,」の間が空白である場合には、エラーとせず、引数が省略されている表記として解釈するように変更した。

具体的には、コンパイル段階で引数リストを取得するオリジナルの args()関数に代わって、新たに作成した argsV()関数で、空白の場合も含めて、引数リストが終了するまで全ての引数の取得を行うように変更した（cci_pars.c）。

④LSS-G コマンドを補足するライブラリ関数

メタファイルの中で、例えば上記②の FACE コマンドのように、引数（頂点数）が不定の関数呼び出しを処理するためには、C 言語の処理系では引数リストを配列に格納して関数に渡すが、本処理系では、メタファイルの記述を容易化するために、一度定義した面の頂点リストの末尾に頂点を追加する FACE_VERTEX 関数を追加した。

FACE_VERTEX(F,V);

この時、F が 25 角形であれば、末尾に新たな頂点 V を追加した 26 角形となる。

以下、これまでに開発した 4 種類の処理系（①VC-1C②VC-2V③VC-3M④VC-4D）を例に用いて、開発プロセスを解説する。コンパイラとインタープリタから成る基幹部分と数値演算関数等の組込関数は、上記の４の実装例に共通である。一方、ライブラリ関数の実際の動作を定義するソースコードは、表 3-2-3 に示すように実装例毎に異なっている。

表 3-2-3 実装形態別の、ソースコード利用状況

ソースコード名（主な機能）	ステップ数		実装形態別使用状況			
	原作	現状	①	②	③	④
cci_code.c (コンパイラのコード生成、実行)	344	1,377	○	○	○	○
cci_face.c （面の生成処理、三角形分割等）	0	859	○	○	○	○
cci_file.c (データファイルへのアクセス)	0	581	○	○	○	○

cci_misc.c (エラー処理、文字コード処理)	75	377	○	○	○	○
cci_pars.c （メタファイルの構文解析)	682	2,553	○	○	○	○
cci_quat.c （四元数演算、測地系変換)	0	1,075	○	○	○	○
cci_tbl.c (記号表処理)	107	226	○	○	○	○
cci_tkn.c (メタファイル解析におけるトークン処理)	222	511	○	○	○	○
cci_io.c (書式付入力)	0	970	○	○	○	○
cci.c （main 関数を含む原本のエントリ部分)	21	109	○	×	×	×
cci_ip.c （ライブラリ関数実装：LSSG 形式ファイル出力)	0	631	○	×	×	×
cci_dummy.c（処理を行う必要がない空のライブラリ関数)	0	14	○	×	×	×
cci_dml.c (ライブラリ関数実装：メモリ上のデータ構築)	0	2,015	×	○	○	×
cci_sml.c (LSSS 形式のファイルの処理)	0	533	×	×	○	×
cci_sql.c (ライブラリ関数実装：SQL コマンドの発行)	0	1,782	×	×	×	○

（4）ライブラリ関数の追加

コンパイラのソースコードに以下の増補を行った。追加する新しいライブラリ関数をfunc とすると、その追加の手順は以下の通り。

a. 関数に対応する関数種別：Func_Fをenumに追加する(cci.h)

b. 関数に対応するトークン種別：QFuncをenum TknKindに追加する(cci.h)

c. 関数に対応するダンプ名称を，ssLibCode[]に追加する(cci.h)

d. 変換テーブルKeyWdTbl[]に、関数名リテラルとTknKindのペアを登録する(cci_tkn.c)

e. compile関数のswitch-case に、新たなTknKind QFuncを追加する(cci_pars.c)

f. statement関数に、新たなTknKind QFuncを加える(cci_pars.c)

関数の戻り値を使用せず、行頭に来る可能性がある場合→正常処理

関数の戻り値を必ず使用し、行頭に来る可能性がない場合→エラー処理

g. factor関数に、新たなTknKindを加える（戻り値のある場合、cci_pars.c)

関数の戻り値を使用する場合→正常処理

関数の戻り値がない場合→エラー処理

h. sys_fncCall関数に、新たなTknKindに対する処理を加える(cci_pars.c)

i. library関数に、case Func_F FUNC(); 等の処理を加える(cci_code.c)

j. ヘッダファイルに、この関数 FUNC()を追加登録する(cci_prot.h)

k.処理系に応じた FUNC()の実際の処理をプログラムする(処理系別のソース)

VC-1C では、cci_ip.c, VC-2V と VC-3M では、cci_dml.c, VC-4D では、cci_sql.c に、ライブラリ関数を実装した。

但し、これらに共通するライブラリ関数は、cci_face.c, cci_file.c, cci_quat.c および cci_vector.c に集約し共通で利用している。

なお､原著でコンパイラとインタープリタの処理系の入り口となっていた main 関数を含

む cci.c は、コマンドラインで起動する初期の VC-1C においては増補しつつ利用したが、以後の処理系においては、ダイアログベースの処理となったため、ビルドからは外している。

以上を理解するために、コンパイラの処理について関連する部分だけを概略解説する（詳しくは原著参照）。

コンパイル処理は、compile()関数(cci_pars.c)配下の関数群により実行される。この時、nextTkn()関数により、メタファイルの構文が解析される。

nextTkn 関数が文字列比較により検出するトークンは、KeywdTbl 配列に、トークンの文字列(keyName)とトークンの種別コード(keyCode)のペアとして、宣言されている。この内、文字列はこの配列の中で直接定義され、種別コードは、cci.h の中で enum 宣言されたトークン種別の一つが参照されている。文字列の内、「(」などの特別な文字は文字コードで直接宣言され、数値型(Void, Int, Float, Quat)、プログラム制御等の文字列(If, For…)、及びライブラリ関数名等は、150 から始まる整数値として定義されている。本処理系のソースコードの中ではこの宣言名を直接使用しており、整数値を直接参照することはない。新たなライブラリ関数を追加する場合には、cci.h の TknKind にトークン種別を加える（b.）と共に、KeywdTbl 配列に、トークンの文字列とこのトークン種別のペアを追加する（d.）。ライブラリ関数は、関数宣言の中か、式の中にしか現れないため、compile 関数の中で直接検出されることはない。通常の処理を行うのは、関数宣言の冒頭にある数値型とセミコロンであり、数値型(void, int 等)の場合には、関数宣言の処理(fncDecl 関数)に進む。

しかし、本処理系の場合には、main 関数を省略した形を含めているため、ここでライブラリ関数が検出された場合には、main 関数の内部として再処理している。そこで、compile 関数の中の switch-case のリストの中に、文頭に現れることのできるライブラリ関数のグループに加える（e.）。compile 関数の中で、文頭に直接ライブラリ関数が検出された場合には、main 関数が省略された形として処理している。

次に、通常の関数宣言形式で構成されたメタファイルの処理であるが、型宣言、関数名、引数リストの後、「{」から「}」までの間をプログラムの記述として処理する。

プログラムの記述には、変数宣言、プログラム制御のキーワード(if,for,do,swtich 等)、変数への代入、関数の実行等が含まれる。それぞれのキーワード毎の処理は、nextTkn()関数で検出されたトークンの種類のコードにより switch されている。

statement 関数において nextTkn 関数によりライブラリ関数の名称が検出された場合には、TknKind でそれぞれの関数の処理を行う（f.）。

ライブラリ関数が戻り値を使用しない命令として利用できる関数の場合には、トークン種別を引数として sys_fncCall 関数を呼び出して、機械語に翻訳し、正常に処理を終了する。

戻り値を必ず使用するライブラリ関数（例えば COORD 関数）がここで検出された場合（つまり文頭に検出された場合）には、エラーとして処理する。

ライブラリ関数が式の中や、関数の引数として呼び出された場合には、expression 関数配下の関数の中で処理される。expression は、8 の階層を有する term から構成され、最も

原始的な単位が factor である。これは数値、変数、関数、括弧で括られた式である。

term のレベルは次の通りである。レベルの高い二項演算が優先的に実行される。

リスト 3-2-1 式(term)の強さのレベル

```
レベル８ : factor
レベル７ : ＊, / , %
レベル６ : ＋,－
レベル５ : ＜,＜＝,＞,＞＝
レベル４ : ＝＝, ！＝
レベル３ : &&
レベル２ : ｜｜
レベル１ : ＝
```

factor には以下のものが含まれる。

リスト 3-2-2 因子(factor)

```
++変数, --変数, +変数, -変数（＊変数には配列も含む）
*変数, &変数
数値
関数
ライブラリ関数
（expression）　//括弧で括られた expression
```

factor 関数の中でライブラリ関数が要求された場合に、sys_fncCall(トークン種別)を呼び出す（g.)。

sys_fncCall 関数(cci_pars.c)の中では、トークン種別で switch-case を行い、それぞれのライブラリ関数毎に引数をスタックに積む処理を追加し(h.)、機械語 LIB に、定義された関数種別をオペランドとして付して処理を終了する。この関数種別は、cci.h の中で enum 宣言されているため、追加する関数をこの宣言に追加する（a.)。

実行処理は、execute 関数（引数はデータファイル名）の中で行われる。この関数は、機械語を順次読み出して実行するが、ライブラリ関数を起動する LIB 命令の実行は、オペランド（関数種別）を引数として、libary 関数を呼び出す。

library 関数(cci_code.c)においては、switch-case で引数の関数種別毎に処理を行う。この際に、利活用処理系により異なる処理が必要となる、図形定義の関数が要求された場合には､その処理の全てを処理系別のソースコードで定義されたそれぞれの関数を作成し(k.)、これにより処理する。更に、この関数のプロトタイプを、cci_prot.h に追加すると共に、ダンプリストを作成する際に使用する文字列を ssOpCode[]配列に追加する。この配列は、関数種別の定義と同じ順序・総数となっていなければならない（c.)。

例えば、メタファイルで座標値を定義する COORD 関数が用いられた場合には、LIB COORD_F という機械語が実行される。すると、library 関数において COORD__F が要求され、そこから、COORD（）関数が呼び出される。

この COORD()関数の実装は、利活用処理系により異なる。例えば、ファイルを出力する VC-1C のための cci_ip.c においては、以下のような処理を行っている。

リスト 3-2-3 VC-1C における COORD ライブラリ関数の実装

```
void COORD(){
        int i;
                float x,y,z;
                z = *((float*)&stack[stp--]);//元データをメモリと見て、送り先に書き込む
                y = *((float*)&stack[stp--]);
                x = *((float*)&stack[stp--]);
        for(i=0;i<NP;i++){
                if( x != X[i]) continue;
                if( y != Y[i]) continue;
                if( z != Z[i]) continue;
                break;
        }
        if(i == NP){
                X[i] = x, Y[i] = y, Z[i] = z;
                NP++;
                fprintf(stberr,"P%d=COORD(%f,%f,%f);¥n",i,x,y,z);
        }
        stack[++stp] = i;
}
```

VC-2V 及び VC-3M でメモリ空間上にデータを構築する cci_dml.c では、以下のように処理している。

リスト 3-2-4 VC-2V, VC-3M における COORD ライブラリ関数の実装

```
int fCOORD(float x, float y, float z){
        /*cci_dmlの場合、配列に蓄積するのみで、ファイル出力もデータベース登録もしない*/
        int i,beda;
//      static int kiri[MP], kanan[MP];
        static int HIT,FAIL,RANDOMFLAG;
        if(NP==0) HIT=FAIL=RANDOMFLAG=0;
        if(RANDOMFLAG){
                i = NP;
                goto nocheck;
        }
        for(i=0;;){
                if(NP==0) break;
                beda = banding(i,x,y,z);
                if(beda<0){
                        if(!kiri[i]){
                                kiri[i] = NP;
                                i = NP;
                                break;
                        }else{
                                i=kiri[i];
                                continue;
                        }
                }else if(0<beda){
                        if(!kanan[i]){
                                kanan[i] = NP;
                                i = NP;
```

```
                                        break;
                                }else{
                                        i = kanan[i];
                                        continue;
                                }
                        }
                        HIT++;
                        break;
                }
nocheck:
                if(i == NP){
                        if(NP%MP==0){
                                if(NP==0){
                                        //X = sX, Y = sY, Z = sZ;
                                        X = (float*)Malloc((NP+MP)*sizeof(float));
                                        Y = (float*)Malloc((NP+MP)*sizeof(float));
                                        Z = (float*)Malloc((NP+MP)*sizeof(float));
                                        kiri = (int*)Malloc((NP+MP)*sizeof(int));
                                        kanan = (int*)Malloc((NP+MP)*sizeof(int));
                                }else{/*注意：Reallocの場合には、解放されるブロックと新たに取得するブロック
を合わせた量の空きメモリが必要*/
                                        X = (float*)Realloc(X,(NP+MP)*sizeof(float));
                                        Y = (float*)Realloc(Y,(NP+MP)*sizeof(float));
                                        Z = (float*)Realloc(Z,(NP+MP)*sizeof(float));
                                        kiri = (int*)Realloc(kiri,(NP+MP)*sizeof(int));
                                        kanan = (int*)Realloc(kanan,(NP+MP)*sizeof(int));
                                }
                                if(!kanan){
                                        exe_err("オーバーフロー,fCOORD");
                                        return 0;
                                }
                        }
                        X[i] = x, Y[i] = y, Z[i] = z;
                        kiri[i]=kanan[i]=0;
                        NP++;
                        if(NP == 1000){//データのランダム性をチェック
                                if(HIT < 10) RANDOMFLAG = 1;
                        }
                }
                return i;
}


void COORD(){
                int i;
                float x,y,z;
                z = *((float*)&stack[stp--]);//元データをメモリと見て、送り先に書き込む
                y = *((float*)&stack[stp--]);
                x = *((float*)&stack[stp--]);
                if(MP <= NP){
                        exe_err("オーバーフロー,COORD");
                        stack[++stp] = -1;
                        return;
                }
                i = fCOORD(x,y,z);
                stack[++stp] = i;
}
```

SQL データベースを構築する VC-4D においては、次のように処理している。

リスト 3-2-5 VC-4D における COORD ライブラリ関数の実装

```
int coord(double v[3]){//一致するものがあればそのIDを、なければ追記し新しいIDを、返す（目標）
          int susun;
          Sel(hS);
          susun = TABLESIZE("coord WHERE X = %lf and Y = %lf and Z = %lf",v[0],v[1],v[2]);
          if(0 < susun){
                    Q("SELECT id FROM coord WHERE X = %lf and Y = %lf and Z = %lf",v[0],v[1],v[2]);
                    return Vint("id");
          }
          Q("INSERT coord VALUES(%lf,%lf,%lf)",v[0],v[1],v[2]);//
          Q("SELECT IDENT_CURRENT('coord') as hasil");
          return Vint("hasil");
}

void COORD(){
          int i;
//        float v[3];
          double v[3];
          v[2] = *((float*)&stack[stp--]);//元データをメモリと見て、送り先に書き込む
          v[1] = *((float*)&stack[stp--]);
          v[0] = *((float*)&stack[stp--]);
          i = coord(v);
          stack[++stp] = i;
}
```

なお、このCOORD関数は、メタファイルの書法においては、３の浮動小数を引数とすることも、１の四元数を引数とすることもできる。COORD関数の引数リストの解析は、

```
void sys_fncCall(TknKind kd); /* ライブラリ関数呼出 (cci_pars.c)*/
```

の中で実行しており、整数または浮動小数が３個であれば COORD 関数（上記の例）を、四元数が１個であれば、COORDQ 関数を呼び出し、それ以外の場合はエラーとしている。

上記のように、この COORD 関数を定義しているソースコードを、利活用処理系により３種作成して、ビルドにこのいずれかを含める方法を採った。

一方、本稿までに試作した４の利活用処理系に共通する処理は、以下の共通のライブラリ関数で記述し、全てのビルドに含めている（表 3-2-3）。

①cci_file.c： ファイル入力処理を行うライブラリ関数

②cci_face.c：頂点と面を定義する処理を行うライブラリ関数

③cci_quat.c：四元数の演算処理を行うライブラリ関数

④cci_sml.c：LSS-S 形式のファイルを入出力するためのライブラリ関数群

（５）機械語レベルで実装した数値計算の関数の書法

座標計算等に使用する超越関数等については、処理系に依存せず、高速処理する効果が高いため、機械語を追加する方法で実装した。メタファイル中の書法としてはライブラリ関数と同様のライブラリ関数として呼び出す。

表 3-2-4 仮想コンバータで追加した数値関数（超越関数）

機械語	円周率定数を取得	メタファイル中での書法

PI	円周率	float PI();
EXP	指数関数（浮動小数）	float exp(float);
EXPQ	指数関数（四元数）	quat exp(quat);
LN	対数関数（浮動小数）	float ln(float)
LNQ	対数関数（四元数）	quat ln(quat);
SIN	正弦（ラジアン）	float sin(float);
COS	余弦（ラジアン）	float cos(float);
TAN	正接（ラジアン）	float tan(float);
ATAN	逆正接（ラジアン）	float atan(float);
SIND	正弦（度）	float SIN(float);
COSD	余弦（度）	float COS(float);
TAND	正接（度）	float TAN(float);
ATAND	逆正接（度）	float ATAN(float);
SQRT	平方根	float sqrt(float);

（６）コンパイル時のメモリ

①記号表

コンパイル時には、メタファイルにおいて新たな変数や関数が宣言される度に、記号表に追加されていく。このテーブルの構造は、cci.h において次のように定義されている。

リスト 3-2-6 記号表を定義する構造体

```
typedef struct {                /* 記号表構成                    */
  char    *name;                /* 変数や関数の名前              */
  SymKind nmKind;               /* 種類                          */
  DtType  dtTyp;                /* 変数,関数の型                 */
  int     aryLen;               /* 配列長。0:単純変数            */
  char    level;                /* 定義レベル 0:大域 1:局所      */
  char    args;                 /* 関数の場合の引数個数          */
  int     adrs;                 /* 変数,関数の番地               */
} SymTbl;
```

記号表は、cci_tbl.c で宣言されており、サイズは 2000（固定）としている。これを超える数の変数名等を使用するメタファイルを扱う場合には、サイズを大きくする必要がある。

リスト 3-2-7 記号表の最大サイズの定義

```
#define TBL_MAX 2000
SymTbl table[TBL_MAX+1];      /* 記号表                */
```

このテーブルに登録される変数や関数の名前の文字列は、s_malloc 関数でシステムメモリから取得している。記号表に登録されるメモリブロックは、コンパイル終了に伴い全て解放される。

本処理系では、s_malloc 関数から、Malloc 関数を呼び出している。

この関数では、デバッグのために、「_DEBUG」が宣言されている場合には、メモリブロックをシステムから malloc 関数で取得する度に、memlist 配列にアドレスを登録している。メモリが解放された場合には、登録されたアドレスを検索して NULL に書き換えている。解放済のメモリブロックを再度解放しようとしたり、メモリブロックが存在しないアドレスを用いて解放しようとしたりすると、エラー処理する。また、システム終了時に解放されていないメモリブロック（アドレス）を検査し、ログファイルに出現順位を保存する。次にシステムを開始する際に、前回終了時に検出されたこのメモリーリークをロードしておき、同じ順位のメモリ要求に際してブレークを掛けることで、問題個所を特定している。このデバッグは、新たなライブラリ関数の追加などのシステムの改良に際して行う手順であり、メタファイルの作成時には必要ではない。

「_DEBUG」が宣言されていない場合には、単純に malloc 関数を実行するのみである。

②コード

コンパイルの結果として生成する機械語は、Inst 構造体の配列に格納されていく。構造体は、cci.h で定義されている。

リスト 3-2-8 機械語命令を格納する配列の構造

```
typedef struct {              /* 命令語      */
  unsigned char opcode;       /* 命令コード */
  unsigned char flag;         /* フラグ      */
  int           opdata;       /* 数値か番地 */
} Inst;
```

opcode(命令コード)は、表 3-1 に示した 1 バイトの機械語であり、本処理系では原著に対して大幅に増補している。

opdata は命令コードに添付する数値またはアドレスである。命令コードがオペランドを必要としない場合には、使用していない。この点は、マイクロプロセッサの機械語において多くの場合オペランドが機械語に続くメモリ上のアドレスに格納されているのと異なっている。

フラグは、opdata が変数のアドレスである場合に、大域アドレスか、局所アドレスかを識別している。 0 ならば大域アドレス、 1 ならば局所アドレスである。

機械語を格納する固定長配列は、cci_code.c でリスト 3-2-9 のように宣言されている。

リスト 3-2-9 機械語命令格納領域の配列宣言

```
Inst code[CODE_SIZ+1];        /* コード格納                    */
```

この固定長配列の長さは、cci.h でリスト 3-2-10 のように定義されている。

リスト 3-2-10 機械語命令格納領域の配列の長さの定義

```
#define CODE_SIZ 200000       /* コード格納領域サイズ 初版20000   */
```

機械語が長大になるのは、固定形状を記述する大きなメタファイルをコンパイルするよう

な場合であり、データ構造が単純であれば、データファイルが巨大であってもメタファイルは短く記述できるため、この配列を小さくすることができる。

③変数領域

実行時の変数や配列の値は、メモリ上のアドレス上に読み書きされる。このメモリは、cci_code.c において、char 型配列としてリスト 3-2-11 のように宣言されている。

リスト 3-2-11 定数、変数や配列を格納するメモリの配列宣言

```
char memory[MEM_MAX+1];      /* memory[0]～[MEM_MAX]がメモリ領域  */
```

このサイズは、cci.h において、リスト 3-2-12 のように定義されている。

リスト 3-2-12 定数、変数や配列を格納するメモリの配列の長さの定義

```
#define MEM_MAX 0xFFFC       /* メモリサイズ*/
```

memory 配列は、３の部分に分けて使用される。

a. 一番先頭の４バイトには、静的領域サイズ（整数）が記憶される。この長さは整数の長さ(sizeof(int):処理系に依存)である。

b. これに続く領域に、リテラル定数、グローバル変数、グローバル変数がコンパイル時に割り当てられていく。コンパイル開始時には、先頭アドレス４がポインタ globlalAdrs の初期値として設定され、コンパイル処理が進むに従い、この値が増加する。コンパイル終了時には、この末尾の値が先頭の４バイトに格納され、実行時の処理に使用される。

グローバル変数等の割り当ては、mallocG(int)関数により行われ、先頭から順に変数等にメモリが割り当てられる。

mallocS(char*)関数により、printf 関数、scanf 関数、logf 関数の書式文字列などの、メタファイル中で使用されたリテラル定数もこの領域に確保される。

globalAdrs が指し示すこの領域の末尾は、コンパイル処理中は次第に成長（アドレスが増加）し、コンパイルが終了すると確定する。その値は①に書き込まれる。

メタファイル中で使用される printf, scanf,logf 等の関数の書式文字列は、グローバル変数領域に、リテラル定数（固定的な文字列）として格納され、その先頭アドレスを用いて参照される。

c. 関数の中で定義され使用される局所変数と局所配列は、実行時にメモリ末尾から動的に割り当てられるため、コンパイル段階では仮のアドレスのみが決定されている。

（７）実行時のメモリ

①スタック

スタックは、整数の固定長配列として、cci_code.c においてリスト 3-2-13 のように宣言されている。

リスト 3-2-13 スタックの配列宣言

```
int stack[STACK_SIZ+1];      /* オペランドスタック                    */
```

この配列のサイズは、cci.h でリスト 3-2-14 のように定義されている。

リスト 3-2-14 スタック配列の長さの定義

```
#define STACK_SIZ 10000        /* スタックのサイズ                    */
```

実行時の数値計算はスタックを用いて行われる。単項演算の場合には、スタックのトップに対して行われた演算結果がスタックのトップに返され、スタックポインタ stp は変化しない。二項演算（例えば加算）の場合には、スタックのトップとセカンドの間で演算が行われ、結果がトップに残され、stp は 1 減少する。

関数（サブルーチン）の呼び出しに際して、戻り番地（呼び出し元のプログラムカウンタ）もスタックにプッシュされ関数のアドレスにプログラムカウンタがセットされてジャンプする。関数の処理が終了すると RET 命令により、スタックから元の番地が取り出され、プログラムカウンタにセットされ、元の文脈の中で処理が続行される。関数が戻り値を有する場合には、数値型に応じたサイズのデータとしてスタックのトップに結果が残される。

②局所変数、配列

実行時に個別の関数の中で使用される変数や引数に割り当てられる局所的なメモリには、上記の memory 配列で、コンパイル時に大域変数などに割り当てられた領域の末尾から後の残りの領域が使用される。局所変数および配列のアドレスは、関数呼び出しに際して動的に確保されるフレームの中に定義され、フレームの先頭アドレスを指す beseReg からのオフセットとしてアクセスされる。この baseReg は、memory 配列の末尾(MEM_MAX)で初期化され、関数が呼び出される度に baseReg が必要な値だけ引き下げられ、このその差分の領域に、ローカルメモリが割り当てられる。関数呼び出しの都度行われるフレーム確保の際に、メモリの先頭に記憶されているグローバル変数領域の末尾(コンパイル時に globalAdrs の終了時として memory の先頭 4 バイトに格納され、実行開始時に start_localMEM 変数に取得される)との比較を行い、領域侵犯を防いでいる。ローカル変数として長大な配列を宣言した場合や、再帰的な関数呼び出しで再帰回数が多くなったような場合に生じうる。

大きな三次元データを扱う場合には、処理系によっては大量のメモリが必要となる場合がある。しかしながら、基幹部分であるコンパイラ・インタプリタ系のメモリ必要量はメタファイルにおける変数の使用状況に依存するため、単純なデータ構造の場合には必ずしも大きなメモリを必要としない。例えば、VC-1C においては、大きなデータファイルを処理すると、大きな変換結果ファイルが作成される。VC-2V,VC-3M においては、表示処理系の大量メモリ容量の問題となる。VC-4D においては、データベースのテーブルサイズの問題となる。基幹部分のメモリ必要量はむしろデータ構造の複雑さ、バッファリングの必要性などに依存する。

（８）エラー処理

メタファイルのコンパイル時のエラーは、原著においては、err_ss 関数(cci_misc.c)により処理されている。この関数は二つの文字列を引数として受け取り、

行番号：文字列１（文字列 2）

の形でコンソールに出力を行う。派生して、第二引数を省略した err_s 関数、および書式付で整数値を表示する err_fi 関数が用意されている。

更に、エラーメッセージが出力される度に、変数 err_ct を用いて数え、総数が MAX_ERR で定義された値（cci_misc.c の冒頭部分で「#define MAX_ERR 10」と定義）を超えると強制終了している。

コンパイル・エラーが発生した場合には、compile 関数は 0 を返す。その場合には、実行段階に進む意味がないため、エラー処理に進むように各処理系でプログラムする。

コンパイルに成功し、１が返された場合には、実行段階に進む。実行段階でエラーが発生する度に、exe_err_ct に１が追加される。

実行時のエラーは、execute 関数の戻り値にも反映される。

インタープリタ処理系内部でエラーが発生した場合（スタックオーバーフローやゼロ除算等）は、処理が直ちに終了し、０を返す。

メタファイル中で、exit(int)が実行された場合には、引数が戻り値となる。

メタファイルの main 関数の中で、return int が実行された場合には、引数が戻り値となる。

メタファイルの main 関数が終了した場合には、0 を返す。

エラーメッセージは全て、プログラム中に日本語でリテラル定数として書き込まれている。処理系に依存しない共通のメッセージを cci_kanji.h で

```
#define KANJI008 "#コンパイル失敗¥n
```

の形式で定義し、これを利活用処理系で必要とする文字コード（Shift-JIS, utf-8, EUC 等）で保存して、エラーメッセージ出力箇所から参照している。

本処理系では、エラーメッセージの出力先を、利活用処理系によって指定できるようにした。多くの場合、処理に先立って予めログファイルを開いておき、そのファイルハンドルを渡して、各利活用処理系のエラー処理関数の出力先をファイルに行い、エラー個数が１以上の場合にこのファイルを表示するように処理している。

実行時のエラーは、原著においては、exe_err 関数(cci_code.c)により処理されている。この関数では、引数として渡されたメッセージ文字列を、プログラムカウンタおよびその時に実行していた機械語と共に表示する。

本処理系においては、上記のコンパイル・エラーと同様に、実行時エラーの出力先も実行前に準備したログファイルに出力するようにした。この exe_err 関数は、各種利活用処理系で実装するライブラリ関数からも呼び出すことができ、各処理系のトラブル（表示装置

の異常、データベースの不具合等）などを表示することもできる。但し、仮想現実等の表示エラーを、エラーを生じている表示装置を用いて行おうとすると、多重エラーが発生して収拾がつかなくなる恐れがあるので、注意を要する。

更に、メタファイルの中で、デバッグ用のメッセージや、メタファイル実行時に検出したデータファイルの問題点を指摘できるように、LOGF 関数を用意し、これの出力先を exe_err 関数の出力先と同じくログファイルに行うようにしている。

cci_prot.h で、outfile と stberr の二つの出力ファイルハンドルを定義している。

エラー出力先 stberr は、全ての利活用処理系において、compile 関数を呼び出してコンパイルを開始する前に開いておかなければならない。上位の処理系から引数等として渡されたファイル名が無効の場合には、必ず作成可能な固定名称等の一時的ファイルを開いて、stberr をファイルハンドルとする。

インタープリタで実行する段階では、二つのファイルが開いていれば、logf 関数の出力先を stberr とする。printf 関数の出力先は outfile ファイルハンドルとし、これが使用できない場合には stberr ファイルハンドラにエラーメッセージを出力する。

（９）ヘッダファイル

原著では、コンパイラーインタープリタ処理系で簡潔するソースコード群のためのヘッダファイルとして作成されていたが、様々な利活用処理系の中に組み込むことを想定して、以下のように再整理した。

① cci.h

コンパイラ処理系に必要なインクルード

コンパイラ処理系に共通の定数、構造体等の定義

② cci_prot.h

ライブラリ関数のプロトタイプ定義

ここでプロトタイプ定義された関数は、全ての利活用処理系において実装されていなければならず、関数型は一致していなければならない。当該処理系が必要としない関数は、ダミーとして設定する。

[例]実装例である cci_ip.c, cci_dml.c, cci_sql.c においてインクルードしている。

③cci_kanji.h

コンパイラ処理系の２バイト系メッセージ

（処理系が使用する文字コードに変換して保存する）

④cci_export.h

仮想コンバータを利活用処理系の API から利用する入口の関数（compile 関数、execute 関数等）のプロトタイプ宣言を含む。利活用アプリケーションからこの処理系を呼び出すソースコードでインクルードする。

[例]2Vwnd.cpp、mobile.c、VC-4DDlg.cpp、 VC-4D_exe.cpp

（１０）記号表ポインタ型変数

本処理系においては、標準の C 言語で利用できるような、システムのメモリ空間にアクセスできるようなポインタ型変数の使用を認めていない。しかし、内部に変数を有するデータファイルのためのメタファイルの記述において記号表を使用することは便利である。そこで、記号表へのアクセスに限定して、ポインタ型の記述ができるようにしている。

＊（式）の形の表現は、式を評価した結果の整数値がランタイムに生成する記号表の範囲（１～末尾）である場合に、記号表に登録された変数名と変数値にアクセスするために使用される。

①スキャン関数による記号表登録

rc = scanf(“%s”, word);

機械語は、LIB SCANF_F である。その処理内容は、int SCANF()関数(cci_file.c)に記述されている。引数の書式文字列のメモリ上の番地と、word のアドレスがスタックを通じて渡される。

この関数により、データファイルから文字列（例えば”HOUSE1”）が取得され、記号表の登録番号が word に格納され、整数型のデータ格納域が固定的に確保される。この文字列が既に登録されている場合には、その番号が返され、未登録の場合には新たな番号となる。

記号表にはスキャンされた文字列「HOUSE1」の名前を有する整数型変数が登録され、対応する 32 ビットのメモリが割り当てられる。この時、コンパイル時点で記号表に登録された”word”の名称はコンパイル終了時点で既にクリアされ、word はアドレスとしてのみ存在している。この word の格納域に、新たに登録された変数の ID 番号が格納される。この番号が仮に３であれば、*word により、この記号表の要素 table[3]つまり「HOUSE1」にアクセスすることができる。内部的には、ポインタ*word は、MULTI_T という特殊な型として処理されている。

②ID 値

printf(“%d”,word)では、word に格納された値「３」が出力される。これが、新たに記号表に登録された、ランタイムの変数の記号表における ID（添字）である。

③記号表へのアクセス

例えば上記のように新規に登録され word に格納された記号表のアドレス（配列の添字）が３である時に、 printf(“%s”, *3), printf(“%s”,*word)は、記号表に登録された「HOUSE1」を出力する。また、printf(“%d”,*3)は、ランタイムに登録された変数 HOUSE1 に対応する値を出力する。

*word = 123; で、HOUSE1 変数に１２３が代入される。よって、

*word = GROUP();

というコマンドを実行することにより、HOUSE1 変数に、GROUP の ID が登録される。

これを用いて、GROUP_FACE(*word, *face1); のように、記号表[face1]に登録されている面を、記号表[word]に登録されているグループオブジェクトに対して割り付ける。

④ポインタの内部処理と機械語

コンパイル終了時点では、メモリ上には、メタファイルが使用するグローバル変数の格納域と、リテラル文字列が格納されている。その末尾の後に、ランタイムに新たに記号表に登録された変数の格納域が追加されていく。

マシン語レベルでは、例えば VERTEX 関数は、頂点座標、カラーID 等を記入した変数を引数とする。この時、それぞれの引数のアドレスが、LDA スタックに積まれて、機械語

LIB VERTEX_F

が実行される。この引数としてメタファイル中の変数名が直接列挙されている場合には、引数は、LDA　アドレスの並びとしてスタックに積まれる。一方、引数として、*word の表現が行われた場合には、

LOD word　・・・word 変数に格納されている値をスタックトップに

TVAL　・・・記号表[値]のデータ格納アドレスをスタックトップに

以後、このアドレスの値を取得し、値を書き込むことができる。

記号表を書き換え、登録されたデータ格納アドレスを変更する機械語は存在しない

LOD word

TNAME　・・・記号表[値]の文字列格納アドレスをスタックトップに

スキャンした文字列を表示してデバッグ等に使用することができる。

文字列自体を書き換えるような機械語は存在しない。

この機械語は、例えば

printf(“%s”, *word);

のような形式で、記号表の文字列が書式文字列で要求した場合に、コンパイラにより生成される。

*word = 12345;

では、記号表[値]が指示するデータ格納域に格納された値が変更されるのみである。

printf(“%d”, *word);

のような形式でこの値を出力することができる。

ix = *word;

により、データ格納域の値を別のメタファイル変数 ix に代入することができる。

if(*word <10000000)｛・・・｝

のように条件判定に使用することができる。

func(*word);

のように、関数呼び出しの整数型引数として使用することができる。

⑤関連ソースコード

コンパイル段階で、expression 関数(cci_pars.c)の中で、ポインタ型の変数を処理し、MULTI_T という型を与える。

printf(“%s”, *(式)) ; の形式で評価された場合にのみ、記号表の文字列を参照する。この

場合には、機械語の TNAME が追加される。

その他の参照が行われた場合には、整数型として記号表の変数格納域の値を参照する。この場合、＊に対応して、機械語の TVAL が追加される（表 3-2-1、p.3-8)。

実行段階で、expression 関数の冒頭で、記号表がゼロクリアされ、スタティック変数に、コンパイル時に生成されたリテラル文字列やメタファイル中のグローバル変数の格納域の末尾がローカル変数 start_localMEM に格納されランタイムの間記憶される。

ここを先頭としてランタイムで生成される記号表の登録文字列や変数格納域が成長し、その末尾がグローバル変数 globalAdrs に格納される。ポインタのアクセス先がこの範囲の外にある場合には、実行時エラーとしている。

記号表のサイズは、グローバル変数 tableCnt に記憶され、ランタイムに 1 に初期化されている。ランタイムには、ランタイム変数の登録に従いこの値が増加する。ポインタ型での参照が行われた場合には、アドレスが 1 以上 tableCnt 以下であることをチェックし、この範囲外である場合には、実行時エラーとしている。

３－３．VC-1C の開発

（1）試作段階でのコンソール・アプリケーション

開発の初期の段階で、コンパイラ、インタープリタの動作を確認すると共に、個々のライブラリ関数を作成し、既知のサンプルデータを用いて動作確認を行うために使用した。

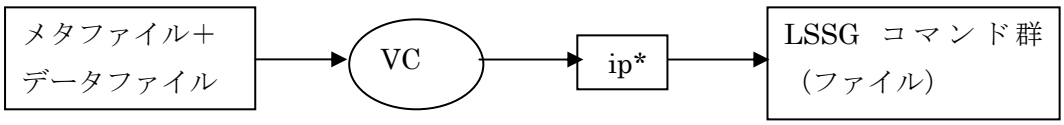


図 3-3-1　VC-1C による処理 (*ip は、cci_ip.c によるライブラリ関数の実装)

VC-1C の実行形式は、VC-1C.exe というコンソール・アプリである。コンソール（Windows では command.com または cmd.exe）からキーボードを用いてコマンドを入力する。

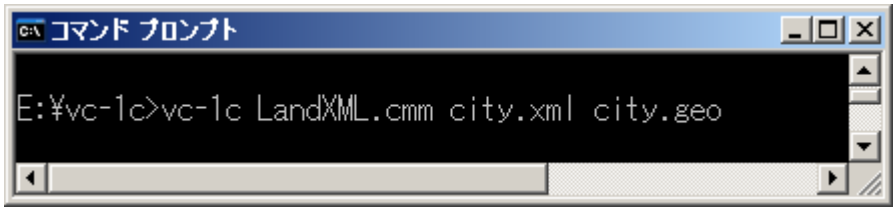


図 3-3-2　コンソールからの VC-1C の起動（変換実行）

(city.xml というデータファイルを、city.geo に変換する)

コマンドラインの引数は 3 である。

通常の場合、第一引数はメタファイル、第二引数はデータファイル、第三引数は、処理結果やエラーメッセージを出力するファイルである。main 関数の中では、第一引数に指定されたメタファイルをコンパイルし、実行形式を生成する。コンパイルに失敗するとここ

で終了する。コンパイルに成功すると、次の段階に進み、生成された機械語が仮想マシンの中で実行され、第二引数で指定されたデータファイルを解釈する。その結果は第三引数で指定されたファイルに格納される。

デバッグ時には、第二引数として、「--code」という文字列を入力し、コンパイルの結果生成する機械語をコードダンプして第三引数で指定したファイルに出力する。

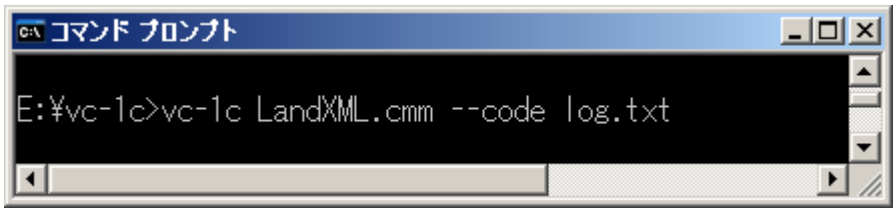


図 3-3-3　コンソールからの VC-1C の起動（デバッグ）

（LandXML.cmm というメタファイルをコンパイルし、ダンプリストを log.txt に出力する）

なお、cmd.exe は、MS-DOS において command.com がユーザーキーボード入力した指令を文字列として解釈して実行していた操作環境を、Windows 上の一つのサブ・ウィンドウとして継承したもので、その後 PowerShell.exe に発展している。

開発環境としては、VS2005 を使用し、OS は Windows Vista を主に使用し(2012 年まで)、後に Windows8.0 でデバッグを行った(2013 年以降)。なお、VS2005 の後継である開発環境 VS2008 は、操作性等において VS2005 から大きく変化しておらず、新たに追加された機能を開発に必要としなかったため、VS2005 を継続使用した。完成後の可搬性検証段階では、最新のものを含む様々な開発環境の上でテストすることは重要であるが、初期の開発着手から一定の安定動作を実現するまでの間は、たとえ古くとも、枯れて安定した開発環境を使用する方が能率的である。開発完了後に新しい開発環境に移行することは容易（ほぼ自動的）であるが、逆に新しい開発環境における成果を古い開発環境に戻してテストすることは、ビルドの再定義等の手入力が必要である。

LSS-G 形式のファイルは、データファイルにかかわりなく固定形状を記述した一種のメタファイルと見なすことができる。その内部においては変数や関数は存在せず、定数値を引数とするライブラリ関数だけから構成されている。LSS-G 形式のファイルをメタファイルとして読込んだ結果出力されるファイルは、景観シミュレータにおいて同じ表示を与えるものでなければならない。

ある形式の三次元データファイルを解読するメタファイルを作成し、解読結果を VC-1C におけるライブラリ関数の実装形態で処理することにより、他の実装形態（VC-２V 等）において「データファイル＋メタファイル」のセットと同様の作用を、メタファイル単独で行うことのできるような「固定形状を記述するメタファイル」を得ることができる。

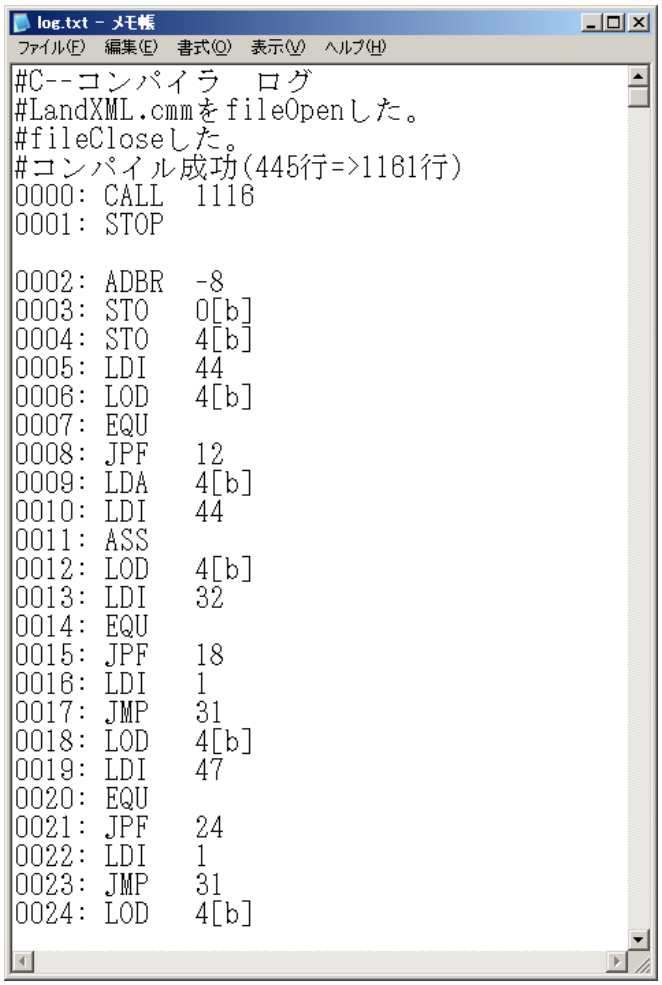


図 3-3-4 デバッグモードで作成した、機械語ダンプリストの冒頭部分

VC-1C におけるライブラリ関数の動作は、cci_ip.c の中で定義されている。LSSG コマンドをファイル出力する。これを景観シミュレータでファイルとして読み込み、正しく変換されていることを確認する。

例えば、P0=COORD(1.0, 2.0, 3.0);　というメタファイルの１行が実行されると、出力ファイルに、同じ内容の１行が出力される。正しく変換されていれば、このファイルを、景観シミュレータの LSS-G ファイルとして読み込むことができ、形状を確認することができる。

この実装形態は、仮想コンバータ開発の最も初期にコンパイラの開発とテストのために作成したものである。VS2005 のデバッガを用いて実行状況をトレースすることにより、バグがコンパイラ処理系によるものか、インタープリタによるものか、メタファイルによるものかを切り分けて解決する作業を行った。

（２）景観シミュレータの外部関数としての活用

現在は実用的な意味を持たせるために､パラメータ入力用のダイアログ vc-1c_D.exe を付して、景観シミュレータの外部関数の一つとして追加してある。

外部関数は、ユーザー定義による各種パラメトック部品の実行形式であり、景観シミュレータ本体から起動する。景観シミュレータの実行形式を変更してバージョンを多様化することなしに、選択的な機能を追加するための、初期から用意した仕組みである。

関数は「外部関数のリスト」(ext.tab)に追記することにより、景観シミュレータのメニュー[形状生成][オプション]から選択できるようになる。

外部関数を景観シミュレータから起動するためには二つの方法がある。

1) ＬＳＳ－Ｇ形式の外部ファイルの中での FILE 記述による起動

以下のような形式で外部関数を起動するコマンドを記述することにより、このファイルをロードする過程で子プロセスとして外部関数を実行する。

リスト 3-3-1 VC-1C の外部関数としての起動

```
（一般形）
グループ名称=FILE(外部関数名, パラメータ１, パラメータ 2, ・・・・);
（コンバータ STL2LSS.exe の場合）
グループ名称=FILE(STL2LSS,変換元ファイル名.stl);
（VC-1C.exe の場合）
グループ名称＝FILE(VC-1C, STL.cmm, 変換元ファイル名.stl);
```

外部関数には小数のパラメータから複雑な形状を生成するためのパラメトリック部品とならんで従来、各種のファイルコンバータ（入力元のファイルはそれぞれの固定的な形式）が登録されている。外部関数名と、パラメータの数およびデータ型を、外部関数と同じディレクトリに配置されたテキスト形式の登録テーブル ext.tab の中に１行を追加登録する。

リスト 3-3-2 外部関数 VC-1C の ext.tab への登録

```
（一般形）
FILE(外部関数名,パラメータ１の型,パラメータ２の型,・・・・);
（コンバータ STL2LSS.exe の場合）
FILE(STL2LSS, STRING);
（VC-1C の場合）
FILE(VC-1C, STRING, STRING);
```

この表に登録されていない外部関数が要求された場合には、エラーとして処理する。

2) パラメータ設定ダイアログを通じた起動

パラメータ設定用のダイアログの実行形式(Windows アプリ)を、外部関数実行形式(「外部関数名.exe」という名称のコンソール・アプリケーション)とは別に「外部関数名_D.exe」

という名称で用意した(図 3-3-8)。これは各種の外部関数と同様の扱いである。

起動に際しては、景観シミュレータのプルダウン・メニューから[形状生成][オプション]をまず選択する(図 3-3-5)。

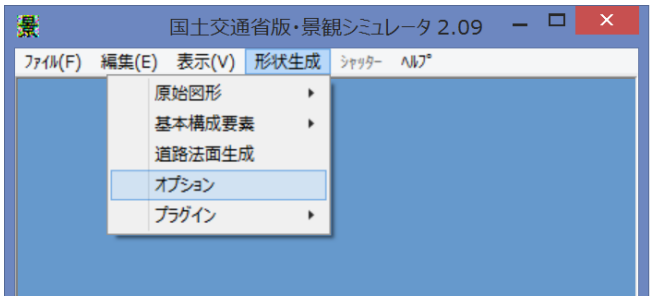


図 3-3-5 景観シミュレータの[形状生成][オプション]で外部関数の一覧を表示

この時、ext.tab に登録された関数の一覧が表示されるので、VC-1C を選択すると、VC-1C_D.exe が起動し、パラメータ設定画面が開く(図 3-3-8)。パラメータを入力した上で、このダイアログの[実行]ボタンの操作により、VC-1C.exe が起動する。

実行ボタン操作後、このダイアログに入力されたパラメータから、リスト 3-3-1 に示した形式の 1 行を持つ一時的なファイル VC-1C.geo がテンポラリディレクトリ(ksim/temp)に作成され、ダイアログが終了し、景観シミュレータに制御が戻る。景観シミュレータは引き続きローダを起動する。ローダはこの行を実行して、1)の場合と同じ形状生成の処理を行い、現在表示されているシーンの中に生成したオブジェクトを追加して終了する。

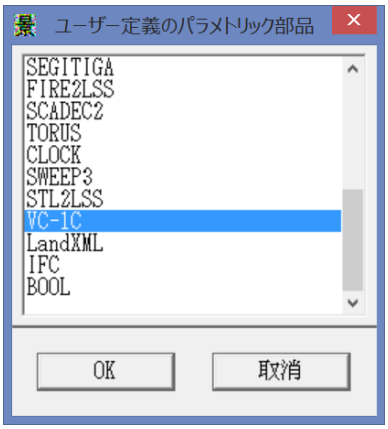


図 3-3-6 ユーザー定義のパラメトリック部品から、VC-1C を選択

3)生成したオブジェクトの選択によるパラメータの再設定

過去に外部関数を用いて生成した物体が景観シミュレータの画面に表示されている時に、これを画面から左クリックで選択し強調表示された状態で、更に右クリックによりポップアップするメニューから、「オプション」を選択すると(図 3-3-7)、パラメータ設定用のダイアログの実行形式が再度起動され、現在設定されているパラメータを表示する(図 3-3-8)。

これにより、修正したパラメータ(VC-1C の場合には、形式定義=メタファイル名または形状記述=データファイル名)を用いて解読→形状生成の処理を再実行することができる。

景観シミュレータから起動するコマンドの指定の中には、変換結果を出力するファイル名が明示的に示されていないが、これは内部的に外部関数名から、VC-1C.g という一時的なファイル名を自動生成した上で、実行形式 VC-1C.exe を起動する段階でパラメータに第3引数(出力ファイルの名称)として付け加えていることによる。

ダイアログ部でパラメータが決定され、実行ボタンが押された時点で生成した VC-1C.geo という、上記の FILE コマンドの１行だけから成るファイルは、変換が失敗しダイアログ部が再度開いた段階で、VC-1C_D.exe がファイル名入力欄に初期表示する文字列に使用する。また、既存の変換結果を選択して、パラメータを再設定する場合にも、このファイルが作成され、ダイアログ部が開いた時点での表示の初期化に使用される。

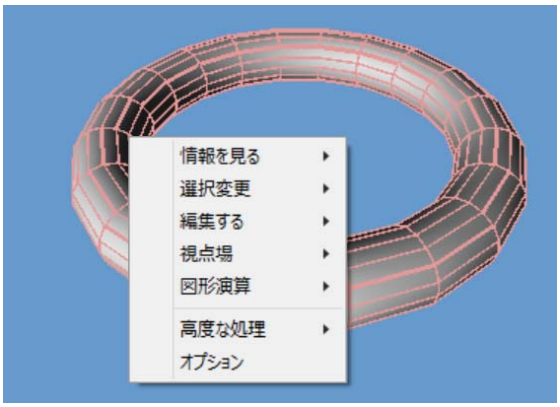


図 3-3-7 既存の変換結果を選択して、パラメータを再編集する場合の操作

ダイアログからパラメータを設定して生成した物体を含むデータをＬＳＳ－Ｇ形式で保存すると、1)で示した形式の１行としてこの物体の保存が行われる。

VC-１C は特殊なファイルコンバータであり、入力元のファイルは任意形式であり、メタファイルで解読方法定義した任意のファイル形式のデータを入力することができる。

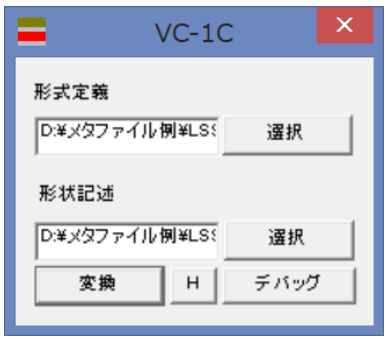


図 3-3-8 パラメータ設定ダイアログ(VC-1C_D.exe)

図 3-3-8 に示したダイアログ画面は従来の他の「ユーザー定義のパラメトリック部品」のダイアログ部と同様に、VC-1C_D.exe という独立した実行形式であり、VC-１C.exe という変換処理を行う実行形式に受け渡す引数を入力するだけの機能を有する。

右下のデバッグを選択することにより、起動元の景観シミュレータに制御を戻すことなく、このダイアログの中で変換を行い、ログファイルを表示し、メタファイルのデバッグを行うことができる(図 3-3-9)。

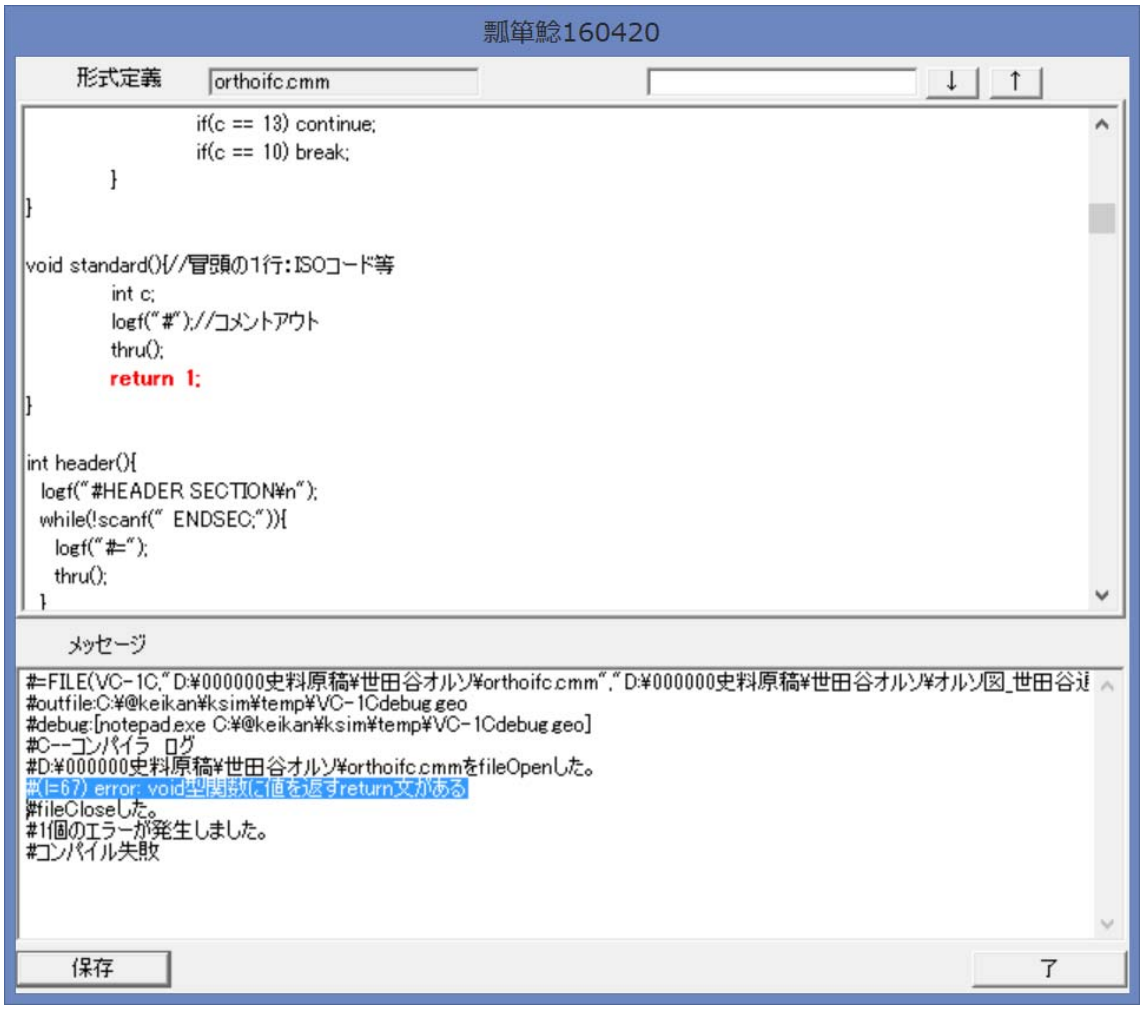


図 3-3-9 VC-1C_D.exe のデバッグ画面

（３）パラメータ設定ダイアログの単独起動

更に、景観シミュレータからの起動ではなく、Windows 上で、このダイアログ部 VC-1C_D.exe を単独のアプリケーションとして起動することも可能である。この場合、変換結果は、固定名称の VC-1CDebug.geo としてテンポラリディレクトリに残されるため、変換結果を検査することができる。また、デバッグ・ボタンを選択することにより、メタファイルを修正する編集画面を開くことができるように改良した。これにより、メタファイルのデバッグ等を行う簡単なツールとして使用することができる（図 3-3-9）。

表 3-3-1　VC-1C の用途と処理内容

用途	メタファイル	データファイル	出力
テスト・デバッグ(固定形状を記述したメタファイルとして実行)	LSS-G ファイル	ダミー	LSS-G ファイル（メタファイルと同じ内容）または、機械語リスト（--code オプション）
簡単なパラメトリック部品の試作	パラメータから形状を生成するプログラム	小数のパラメータを格納した小さなファイル	生成したパラメトリック部品
ファイルコンバータ	コンバータ処理	各種データファイル	LSS-G ファイル
汎用スクリプト言語としての用途	プログラム	パラメータ	printf を使用して各種ファイルを生成

表 3-3-2 VC-1C の起動方法

1. コマンドラインからの直接起動
2. パラメータ設定ダイアログ VC-1C_D.exe からの起動（変換モード、デバッグモード）
3. 景観シミュレータからの起動
3-1.　LSS-G ファイルに埋め込まれたファイルコマンドからの起動
3-2.　形状生成メニューからパラメータ設定画面 VC-1C_D.exe を開く起動
3-3.　生成されたオブジェクトを選択して、パラメータの再設定から VC-1C_D.exe を開く

VC-1C のビルド（VS2005）を表 3-2-3 に、メイクファイルをリスト 4-5-1 に示した。コンパイラ・インタプリタ系のソースコード（原著＋増補）だけによって構成されている。

景観シミュレータのための外部関数の終了コードは、一般に次のように定められている：

表 3-3-3 外部関数エラーコードの一般則

０：未定義のエラー
１：正常終了
２：引数の数が、当該外部関数が必要とする数と合わない場合
３：出力ファイルが開けない場合

外部関数を起動する景観シミュレータ(simexe)ではこの終了コードを見て、1 の場合には生成した形状を入力・表示して終了し、それ以外の場合にはダイアログから起動した場合にはダイアログに戻り、LSS-G ファイル入力処理中の場合には、別途エラー処理を行う。

原著における main 関数は、cci.c に含まれている。main 関数は、通常 4 の引数が与えら

れる｛VC-1C.exe のフルパス、メタファイル名、データファイル名、出力ファイル名｝。実装では、５以上の引数が与えられた場合に、出力ファイルを最後の引数とし、４番目の引数をデバッグなどに保留し、現在は無視している。

従って、VC-4C メタファイル --code データファイル ダミー引数 出力ファイル

のようなコマンドを与えている。

景観シミュレータの外部関数として実装するために、main 関数のリターンコードをリスト 3-3-3 のように定義した。

リスト 3-3-3 main 関数の戻り値の意味

０：エラー １：成功 ２：引数の数が合わない場合 ３：出力ファイルが開けない場合 ４：メタファイルが開けない場合 ５：データファイルが開けない場合 ６：ログファイルが開けない場合 ７：メタファイル・コンパイルエラー ８：実行時システムエラー その他：メタファイルの exit, return 命令により返されたリターンコード

３－４．VC-2V の開発

（１）DML ライブラリを用いたメモリ上のデータ構築

この実装では、ライブラリ関数は、cci_dml.c で記述したライブラリ関数により処理系のメモリ空間上に、OpenGL で表示を行うためのオブジェクトを生成する。このオブジェクトには、頂点座標、面、立体、カラー、法線、テクスチャ、属性等が含まれる。

入力動作は、コンソール・アプリとして一度 LSS-G 形式のファイルを生成してから読み込む VC-1C と同様であるが、処理結果が直接メモリ上に形成され表示に使用されるため、ファイル出力・ファイル入力の処理が不要となり、それだけ処理速度は向上する。

（２）景観シミュレータ用のプラグイン DLL としての実装

具体的な実装例として、景観シミュレータのためのプラグイン.dll の仕様をもつ VC-2V.dll を試作した。この Windows のための dll は、メタファイルとデータファイルの組を入力して表示を行う、という点に関しては、VC-1C と同様であるが、プラグイン DLL であることから現在メモリ空間内にあるオブジェクトにアクセスすることができる。したがって、メタファイルで定義した任意の保存形式で、現在表示されている建物などを指定した名前のファイルとして出力することができる。

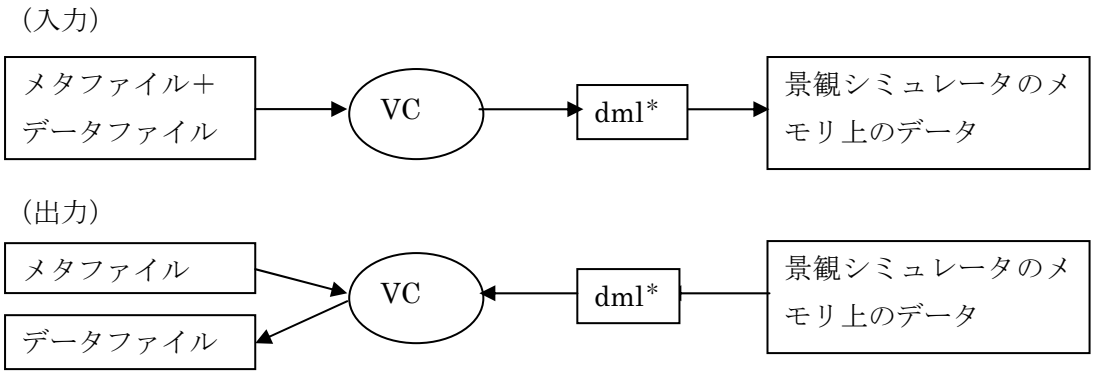


図 3-4-1 VC-2V による処理 (*dml は、cci_dml.c によるライブラリ関数の実装)

ユーザーは、景観シミュレータのプルダウン・メニュー[形状生成][プラグイン]で表示される一覧から、VC-2V を選択することにより、起動する（図 3-4-2)。

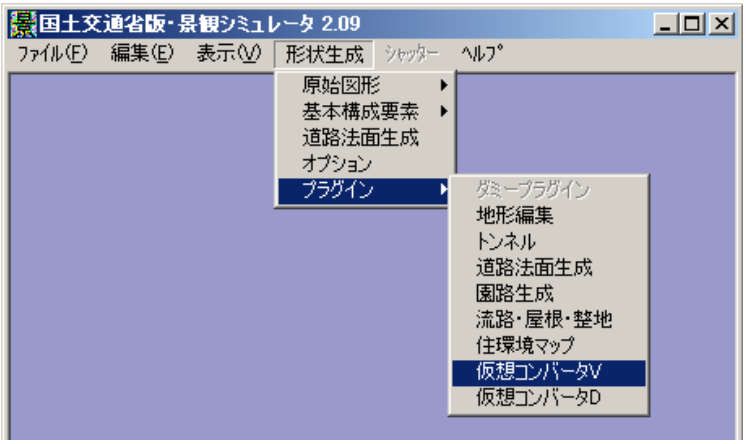


図 3-4-2 VC-2V の起動

メタファイルとデータファイルをファイル選択ダイアログまたは直接キーボード入力により選択し、入力か出力かをボタンで選択する(図 3-4-3)。

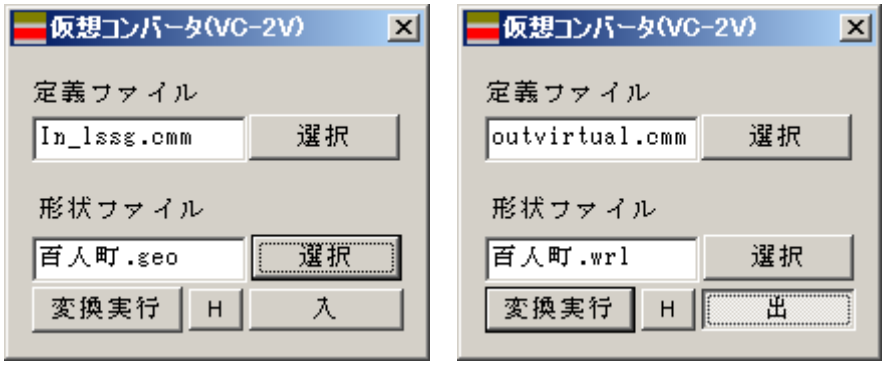


図 3-4-3 VC-2V 画面（左：形状ファイル入力　右：形状ファイル出力）

入力において定義ファイルと形状ファイルを指定して変換実行を行うと、コンパイラにより実行形式が生成され、これを用いたデータファイルが入力され、メイン画面に表示さ

れる(図 3-4-4)。

出力において定義ファイルと形状ファイルを指定して変換実行を行うと、コンパイラにより実行形式が生成され、これが実行されることにより、現在表示されているメモリ上の三次元データが、変換され指定された名称のデータファイルとして出力保存される。このデータファイルは、定義ファイルにより任意の形式とすることができる。

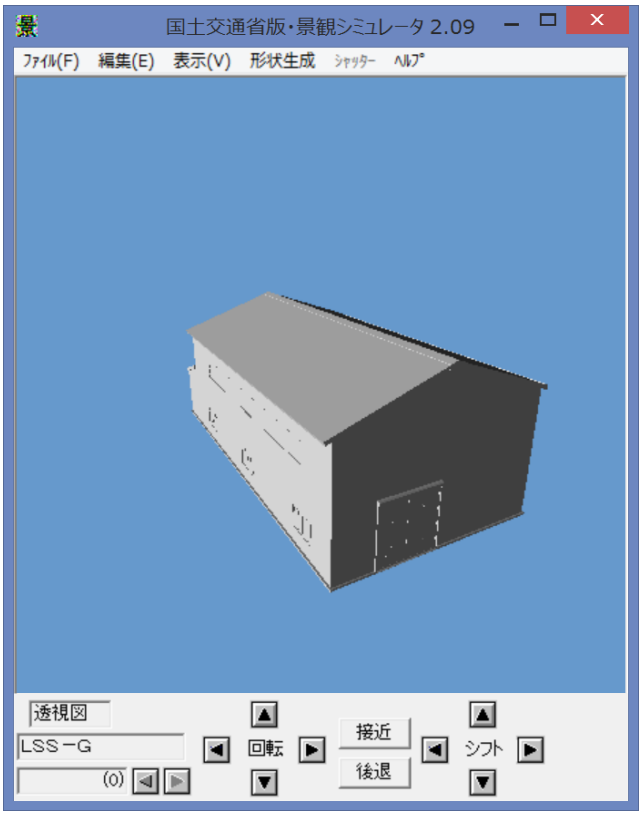


図 3-4-4 データファイル解読結果の表示

（３）出力系関数群の実装

この出力側の変換もまた、利活用形態の一つであり、後述の VC-4D においてまずライブラリ関数の作成と動作のテストを行った。ここで関数使用を定めてから、同じ定義ファイルを用いて出力処理を実行することができるように、VC-2V のライブラリ関数を作成した。なお、(4)VC-3M においても原理的には同じような出力処理を行うことができるが、携帯端末上に生成したデータファイルの実用的な意味が当面ないため、現在はまだファイル名を指定して保存するような操作環境を用意していない。

コンパイラ基幹部分が出力するエラー等に関するメッセージは、景観シミュレーション・システムの一時的ファイル格納用ディレクトリ（デフォルト ksim/temp）に CClog.geo という名称のファイルに保存され、変換処理が終了した時点でテキストエディタを起動して表示する。よって、VC-2V は、メタファイルをデバッグするための作業環境として使用することができる。

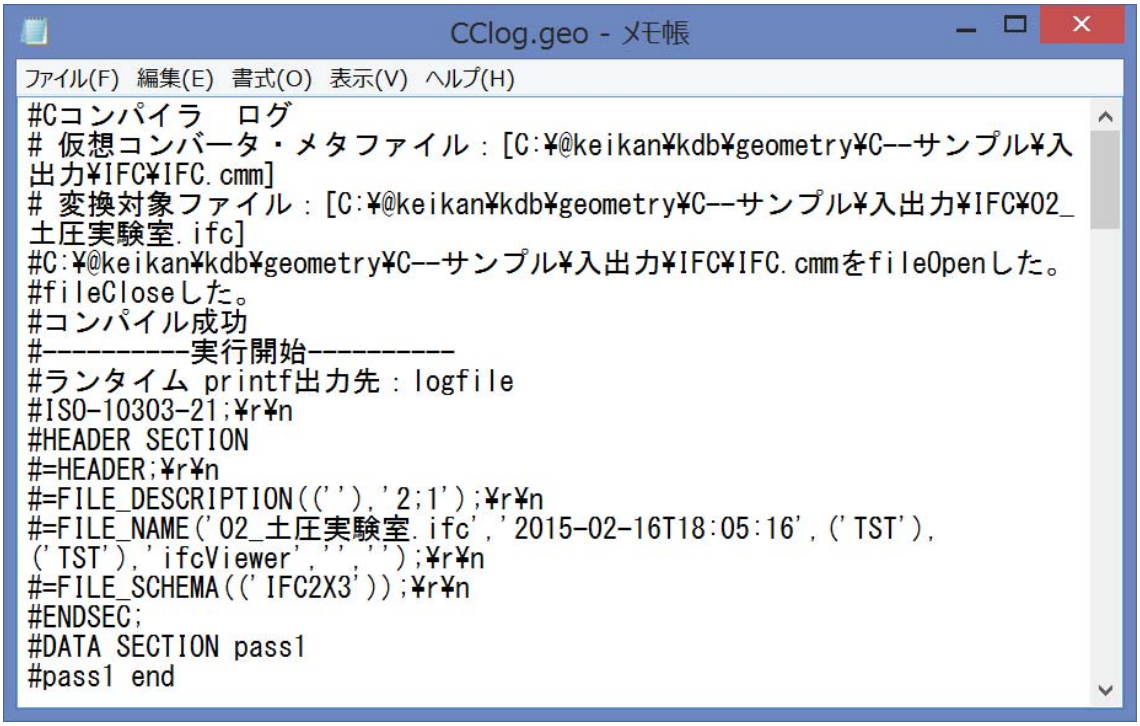


図 3-4-5 テキストエディタによるエラーメッセージ等の表示

データファイルが空欄のまま実行された場合には、コンパイルだけが実行され、生成したコードをテキストエディタで表示して終了する。この動作は、VC-1C において、第二引数として「--code」が入力された場合と同様である(図 3-4-6)。

この機能は、基幹部分のデバッグを行う場合に有用である(図 3-4-7)。

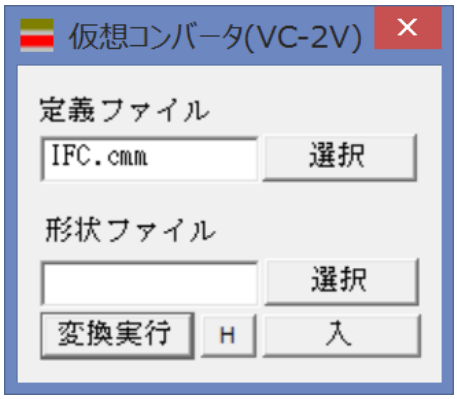


図 3-4-6 機械語表示の指定（形状ファイル入力欄を空欄のまま入力変換実行）

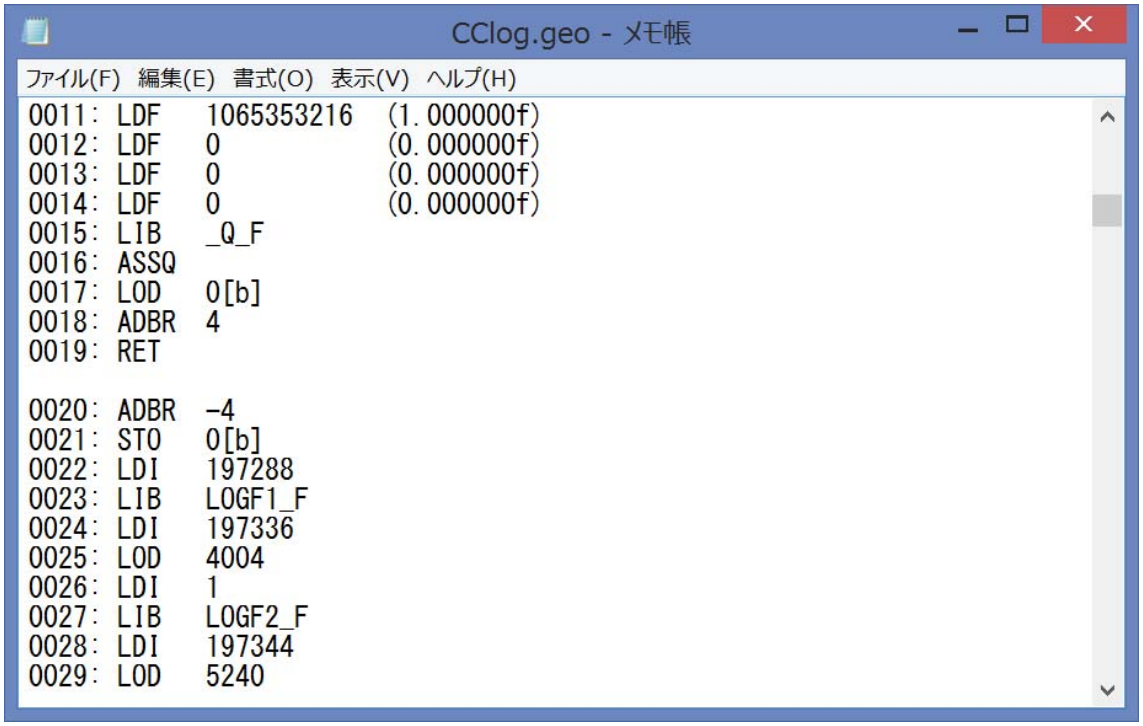


図 3-4-7 テキストエディタによる機械語の表示

実行ボタンが押されると、コンパイル・変換処理のスレッドが開始される。処理が正しく終了すると、入力の場合にはメモリ上にインタープリタがデータファイルを解釈して生成した建物等が画面上に表示される。

表示方法や、ロード後の建物等データの編集操作は、景観シミュレータにおける通常のデータと同様である。偏光液晶パネルを備えた立体視可能なディスプレイが接続されている場合には、立体視を行うこともできる。

VC-2V.dll は、VS2005 を開発環境とし、MFC を用いてプログラミングされている。コンパイラ基幹部分のソースコード群に加えて、DllMain 関数を有し、ウィンドウを構築する VC-2V.cpp、ユーザー操作等のコールバックを処理する 2VWnd.cpp を有する。また、これらに付随したヘッダファイル等を含む。

表 3-4-1 VC-2V のソースコード等

①VC-2V.cpp 景観シミュレータのプラグイン DLL にほぼ共通の基本処理 この DLL の開始と終了の処理
②2Vwnd.cpp 表示するダイアログにおけるユーザーインターフェース
③VC-2V.h ①のクラス等を定義したヘッダファイル
④2Vwnd.h②のクラス等を定義したヘッダファイル
⑤stdafx.cpp 開発環境が自動的に生成する共通ソース
⑥stdafx.h 開発環境が自動的に生成するヘッダファイル

⑦VC-2V.rc ダイアログのレイアウト等を定義したリソースファイル
⑧VC-2V.rc2 同上
⑨VC-2V.dll.ja.xml 日本語のダイアログ文字の定義ファイル
⑩VC-2V.dll.id.xml 外国語のダイアログ文字の定義ファイル（インドネシア語）
⑪VC-2V.ja.txt 日本語のヘルプファイル
⑫VC-2V.id.txt 外国語のヘルプファイル（インドネシア語）

cci_dml から呼び出される、景観シミュレータの DML ライブラリ関数（表示データ構築に関するもの）は、この DLL の呼び出し元であるアプリケーション本体 sim.exe の内部にビルドされた関数を呼び出すことにより実行している（ビルドに sim.lib を加えることにより実現）。よって VC-2V のビルドには、DML 関連のソースコードは含まれていない。

ポイントクラウド等の大きなデータファイルを扱う場合には、数分～数十分の処理時間がかかる場合があり、進捗状況が把握できないと、正常に処理が行われているかどうかの疑念が生じる場合がある。そこで、処理中断の機能、およびプログレス・インジケータの表示機能を追加した。

具体的にはインタープリタの処理系に Debug 関数と、TraverseDebug 関数(cci_code.c)を追加して、インタープリタの execute 関数(cci_code.c)による機械語の実行ステップのループの中で必ず Debug 関数を実行し、戻り値が非ゼロならば終了するようにした。この Debug 関数は、初期状態では何もしないが、TraverseDebug 関数により処理関数が登録されると、その関数を実行する。

VC-2V の操作画面で実行ボタンが押されると、まずデバッグ関数を引数として TraverseDebug 関数を呼び出して、VC-2C のためのデバッグ関数の登録を行う。デバッグ関数の中では、ダイアログ側から、変換データファイルへのアクセス位置をファイル全長で除した割合を、画面に表示する(図 3-4-8)。さらに、実行中にユーザーにより再度実行ボタンが押され、中断が要求された場合には、この寄り道処理の中で処理を中断するようにした。これにより、大規模なファイルのためのメタファイル作成の能率を上げることができる。

デバッグのための関数を登録する TraverseDebug 関数が実行されていない処理系においては、インタープリタの execute 関数は従来通りの動作を行うため、処理速度も従来とほとんど変わらない。

登録されたデバッグ関数を解除するためには、TraverseDebug(NULL)を実行する。

具体的には、ダイアログの実行ボタンが押されたタイミングで、コールバック関数である OnBnClickedOK 関数(2Vwind.cpp)の中で変換処理開始前に&dumy 関数を引数として TraverseDebug を呼び出す。この dumy 関数は、2Vwnd.cpp の中で static int 型として定義されており、データファイルへのアクセス位置をデータファイルの長さで除した値を、データファイル名欄に表示するような動作を行う。また、再度実行ボタンが押されているかをチェックし、もし押されていたら処理を中断して変換を直ちに終了する。

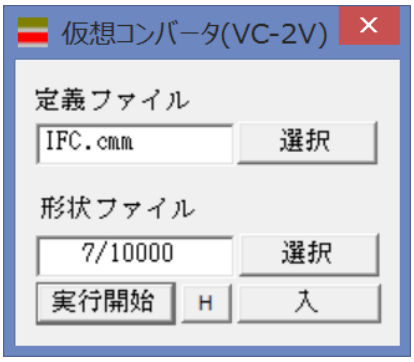


図 3-4-8 プログレス・インジケータ

メタファイルをコンパイルして生成した機械語のプログラムは、cci_code.c の execute 関数の中のループで順次フェッチ（取得）され実行されている。このループの中で Debug 関数が次の機械語の実行に先立って実行されている。

リスト 3-4-1 プログレス・インジケータの処理

```
//cci_code.cで、プログレス・インジケータとブレーク処理に関連する部分

int execute(char *fname) /* プログラム実行 */
{
------------------中略-----------------
  for (;;) {
    if(0 < exe_err_ct) break; //140622
    if(DebugFunction) if(Debug(Pc)) break;//140624
    if (Pc<0 || codeCt<Pc) {
                    fprintf(stberr, "# Pc=%d, codeCt=%d¥n", Pc, codeCt);/* exit(1); */     /* 念のため */
                    exe_err("プログラムカウンタが範囲外");
                    break;
    }
------------------中略-----------------
    機械語のフェッチと実行ループ
------------------中略-----------------
  }//end for
}//end execute
```

この Debug 関数は、DebugFunction が定義されていれば実行し、定義されていなければ（関数へのポインタが NULL)、何もせずに終了する。

リスト 3-4-2 デバッグ関数の動的登録

```
static int (*DebugFunction)() = NULL;//登録デバッグ関数へのポインタ。未登録ならNULL
void TraverseDebug(int (*func)(int pc,int pos, int len))
{//実行時のプログレス・インジケータ情報を受け、ブレークならTRUEを返す関数を登録する
   DebugFunction = func;
}

static int Debug(int pc){
          if(DebugFunction){
                    if(0<FCNT) return (*DebugFunction)(pc,SIORI(),FCNT);//戻り値がTRUEならブレーク
                    else return (*DebugFunction)(pc,0,0); //出力系の場合（とりあえず）
          }else return FALSE; //デバッグ関数登録が無ければブレークなし
}
```

DebugFunction は、プログラムの実行に先立って、TraverseDebug という関数により、Debug 関数を外から与えることができる。VC-2V における実装例として dumy という関数を用意した。この関数は、プログラムカウンタ、データファイルへの現在のアクセス位置、およびデータファイルの長さを取得し、中断が必要ならば１を、継続するなら０を返す。

中断は、実行中に再度実行ボタンが押されると、g_BREAK という大域変数を初期値０から１に変更することによって指示する。dumy 関数は、この値を返すことにより、インタープリタの execute 関数に中断／継続の指示を行う。

リスト 3-4-3 ブレーク処理

```
//2VWND.cppで、実行ボタンが押された時の処理

int g_PC, g_POS,g_BREAK;
#define 実行中g_bRunning
#define 停止中g_BREAK
HANDLE hThread = NULL;
DWORD ThreadId;

// 解読処理実行中にプログレス・インジケータが更新されるように独立したスレッドでメタファイルを実行する
void CreateAndStartThread(){
          g_iCount = 0;
          g_bRunning = true;
          hThread = CreateThread(NULL,0,MyThreadFunc,NULL,0,&ThreadId);
}

void C2vWnd::OnBnClickedOk(){
          // TODO: ここにコントロール通知ハンドラコードを追加します。
          char *path1,*path2;
          int hasil;
          char s[200];

          if(実行中){ //●実行中に再度実行ボタンが押されるとブレークを予約する
                    g_BREAK = 1;
                    GetDlgItem(IDOK)->SetWindowTextA(Kanji("RESET"));
                    return;
          }

          if(停止中){
                    g_BREAK = 0;
                    GetDlgItem(IDOK)->SetWindowTextA(Kanji("START"));
                    SetDlgItemText(IDC_EDIT_GEOFILE,g_geofile);
                    return;
          }

          GetDlgItemText(IDC_EDIT_GEOFILE,g_geofile,80);

          sprintf_s(s,200,"%s¥¥CClog.geo", e3GetEnvParam(E3_TEMP_PATH)->path);
          fopen_s(&stberr,s,"wt");
          fprintf(stberr,"#Cコンパイラ　ログ¥n");

          path1 = metafile.GetBuffer();//メタファイル
          path2 = geofile.GetBuffer();//データファイル

          TraverseDebug(NULL);
          TraverseDebug( &dumy ); //OK.VCにデバッグポートを登録
          SetIndicator(m_hWnd);//インジケートするウィンドウの指定
```

```
        if(b_IO){//ボタンが押されている：出力側
                //メモリ空間上にあるデータに、仮想コンバータがアクセスできるようにする。
                if(!access(path2,0)){//既にある（ないなら－１）
                        if(IDNO == MessageBox("上書き？",Kanji("OUT"),MB_YESNO)){
                                fprintf(stberr,"#既存データファイルへの上書き出力が非選択¥n");
                                fclose(stberr);
                                return;
                        }
                        if(access(path2,2)){//書き込み禁止
                                fprintf(stberr,"#出力ファイル(%s)が書き込み禁止¥n",path2);
                        }else{
                                fopen_s(&outfile,path2,"wt");
                        }
                }else{//まだない、新しいファイルを作成
                        fopen_s(&outfile,path2,"wt");
                }
                if(!outfile){
                        fprintf(stberr,"#出力ファイル(%s)が開かない¥n",path2);
                        MessageBox(path2,"#出力ファイルが開かないため、ログファイルに出力する");
                }else{
                        fprintf(stberr,"#出力ファイル(%s)を開いた¥n",path2);
                }
                //この時、メモリ空間に対して、LSSG系コマンドが実行されると厄介なことに
                //例えば、出力なのにメタファイルがLSSG準拠だとデータが追加されてしまう
                ReadOnly = 1; //cci_parsのグローバル変数
        }else{//入力
                if(access(path2,0)){
                        fprintf(stberr,"入力データファイル(%s)がない",path2);
                        MessageBox(path2,"入力データファイルがない");
                }
            ReadOnly = 0; //cci_parsのグローバル変数
        }
        fprintf(stberr,"# 仮想コンバータ・メタファイル：[%s]¥n", path1);
        fprintf(stberr,"# 変換対象ファイル：[%s]¥n", path2);
        if(path2[0]==0) dump_table(stberr); //140713 untableの中でテーブルをダンプ
        hasil = compile(path1);
        if(!hasil){
                fprintf(stberr,"#コンパイル失敗¥n");
                fclose(stberr);
                if(outfile) fclose(outfile);
                theApp.m_pMain->m_pView->m_drawFrm->RedrawWindow();
                PembantuX(s);
        }else if(path2[0]==0){
                code_dump();
//              dump_table(stberr);
                fclose(stberr);
                PembantuX(s);
        }else{//実行
                d3Group **root;
                int num,rc;
        //dcは、用が済んだらすぐ解放する必要があるため、メモリブロック型で取得
                CClientDC *dc = new CClientDC(theApp.m_pMain->m_pView->m_drawFrm);
                *theApp.m_pMain->m_pView->m_drawFrm->m_HDC = dc->m_hDC;
                wg3AssignDrawarea( (void*)&theApp.m_pMain->m_pView->m_drawFrm->m_HDC );
                rc = g3GetRootGroup(&num, &root);
                wg3AssignDrawarea(NULL);
                delete dc; //このDCを使用するスレッドを起動する前に解放しておく
                if (!rc){
          //            z3Message(5000,"ルートなし");
                        z3Message(5000,Kanji("KANJI_4"));
                        return;
                }
                rootgroup = root[0];
```

```
                    if(b_IO) CreateLGarray(rootgroup);
                    fprintf(stberr,"#コンパイル成功¥n");
                    SetRootGroup(rootgroup);

                    GetDlgItem(IDOK)->SetWindowTextA(Kanji("BREAK"));
                    if(b_IO) g_path2 = NULL;
                    else g_path2 = path2;
                    hwnd = m_hWnd;//マルチスレッド用
                    CreateAndStartThread();
          }
}
```

コンパイルと実行の開始を指示する［実行］ボタンが押された時点で、TraverseDebug 関数の引数に dumy 関数を指定して実行することにより、実行段階でのプログレス・インジケータ表示とブレーク処理を登録する。

リスト 3-4-4 プログレス・インジケータの表示関数

```
//2VWND.cppで登録するプログレス・インジケータ表示と強制ブレークのための関数

static int dumy(int pc, int pos, int len){
          char kata[2000];
          int rate,i,T,H;

          static int prate;//前回の進捗率
          g_PC = pc;//終了時デバッグ出力用
          g_POS = pos;//同

          if(len<1) return g_BREAK;//出力時
          T = len/10 + 1; //ファイルが小さい時、少なくとも１
          H = 1000;
          rate = 0;
          for(i=0;i<4;i++){
            if(!T) break;
            while(T < pos){//千の桁
                    rate += H;
                    pos -= T;
            }
            T = T/10;
            H = H/10;
          }
          if(rate != prate){
                    sprintf(kata,"%4d/10000",rate);
                    SetDlgItemText(laporHWND,IDC_EDIT_GEOFILE,kata);
                    UpdateWindow(laporHWND);
                    prate = rate;
          }
//        return 0; //ブレークしない
//        return 1; //ブレークする
          return g_BREAK;
}
```

最後に、ブレークポイントによる中断である場合には、メッセージを出力する。

リスト 3-4-5 ブレークによる中断の表示

```
//2VWND.cppで、コンパイル・実行を行っているスレッド

DWORD WINAPI MyThreadFunc(LPVOID lpParam){
          int hasil,rc;
```

```
            char s[200];
            fprintf(stberr,"#----------実行開始----------¥n");
            hasil = execute(g_path2); //出力の時はNULL
            fprintf(stberr,"#----------実行終了----------¥n");
            if(g_BREAK){
                        fprintf(stberr,"#ブレークによる強制終了¥n"); //●ブレークによる終了の表示
            }else{
                        fprintf(stberr,"#スレッドが終了¥n");
            }
            fprintf(stberr,"#終了コード：%d¥n", hasil);
            fprintf(stberr,"#終了時PC:%d, SIORI=%d¥n",g_PC, g_POS);
            if(!g_path2) DestroyLGarray();//出力時

    //スレッド起動側で、起動後にDCを解放するため、ここで新たに取得する
            CClientDC dc(theApp.m_pMain->m_pView->m_drawFrm);
            *theApp.m_pMain->m_pView->m_drawFrm->m_HDC = dc.m_hDC;
            rc = wg3AssignDrawarea( (void*)&theApp.m_pMain->m_pView->m_drawFrm->m_HDC );
            if(rc){
                        rc = g3LoadTextureAll();
                        rc = wg3AssignDrawarea(NULL);
            }else{
                        MessageBox(hwnd,"wa3SiignDrawarea失敗","thread",MB_OK);
            }
            SetRootGroup(NULL);//140605

            fclose(stberr);
            if(outfile) fclose(outfile);
            theApp.m_pMain->m_pView->m_drawFrm->RedrawWindow();
            //          PembantuX("c:¥¥@keikan¥¥ksim¥¥temp¥¥CClog.geo");
            sprintf_s(s,200,"%s¥¥CClog.geo", e3GetEnvParam(E3_TEMP_PATH)->path);
            PembantuX(s);

//          GetDlgItem(IDOK)->SetWindowTextA(Kanji("START"));
            if(!g_BREAK){//最後まで実行された場合
                        SetDlgItemText(hwnd,IDOK,Kanji("START"));
            }
//          g_BREAK = 0;
            g_bRunning = false;
            ExitThread(0);
            return 0;
}
```

３－５．ＶＣ－３Ｍの開発

（１）OS を Android とする場合の開発課題

ライブラリ関数は、処理系のメモリ空間上に、OpenGL で表示を行うためのオブジェクトを生成する点は、VC-2V と同様である。また、背面カメラから取得した画像の上に三次元モデルからレンダリングした CG を合成表示する処理は、景観シミュレーションにおける写真合成と同じ処理である。但し、背景画像と写真合成に必要なカメラ（視点座標やカメラアングル）に関する情報が時々刻々、リアルタイムで入力される点が異なる（いわば、「リアルタイム写真合成」とも呼ぶべき処理である）。

この処理系は、Android OS 上で使用する VC-3M.apk として実装し、サンプルデータを現場で、GPS センサで取得した視点位置から、磁気センサおよび加速度センサで取得したカメラアングルで表示するための、三次元データを用いた一種の AR アプリ(Augumented

Reality)である。具体的な配信に際しては、現場毎に、表示するメタファイル+データとアプリをセットとしたパッケージとして提供している。GPS 位置情報の精度を除き、何も手掛かりがないような茫漠とした場所であっても、記録された集落等を表示することができる。

モデルを記述するために用いた座標軸の原点を緯度・経度・標高で記述する。対象物が視界の中に無い場合には、左右あるいは後方のいずれにあるかを画面中央の文字表示でユーザーに指示する。対象物までの距離が大きい場合にも文字表示でユーザーに知らせる。

Windows 版の景観シミュレータからは独立して利用できる処理系であるが、OpenGL 表示、およびメモリ上のデータ構築に関して、景観シミュレータのライブラリ（ソースコード）を多く活用した。

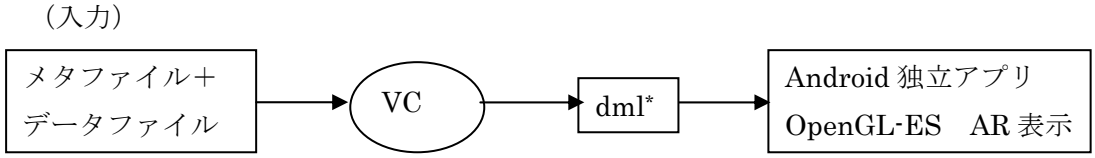


図 3-5-1 VC-3M による処理(*dml は、cci_dml.c によるライブラリ関数の実装)

小さいながら仮想コンバータのコンパイラ機能を搭載しているため、携帯端末だけを用いてメタファイルをプログラミングし、携帯端末上で利用できるテキストエディタを用いてデバッグすることもできる。

①開発環境

開発に際しては、Windows 上で動作する開発環境 Eclipse を使用した。更に、仮想コンバータのソースコード群（C 言語）をコンパイルするために、携帯端末で提供されている DALVIC 仮想マシンのための機械語を生成する ndk を使用し、これを用いて仮想コンバータの機能を実装した DLL（VirtualConverter.so）を作成した。表示速度を高めるために、OpenGL による描画部分も C 言語で記述し、この DLL の中に含めた。

②OpenGL

Android 上で利用した OpenGL ライブラリは OpenGL ES1.0 であり、景観シミュレータ等で使用している Windows 版と比較すると、利用できるコマンドが少なく、制約されている。具体的には、4 以上の頂点を有するポリゴンが出力できず、三角形のみの出力となる。このため、景観シミュレータで描画処理を行っている g3drl ライブラリを活用しつつ、不足する OpenGL 関数に関しては、関数プロトタイプを定義しているヘッダファイルの中に直接処理を書き込む方法で実装した。例えば、多角形の出力が要求された場合には、3 頂点からなる配列を用意しておきバッファリングを行い、多角形の全ての頂点が出力された段階で、分解後の三角形の描画を OpenGL ES ライブラリに対して要求している。

③センサ計測値による視点座標、カメラアングルの取得

一方、ユーザーインターフェース、各種センサの計測値の取得は、マーカーを用いたARアプリの開発実績があるエム・ソフト社に外注し、全体をVC-3M.apkというセットアップファイルに集約した。2012年2月に、東京都内、つくば市内および釜石市内でテストを行った。東京都内では、磁気センサの値が直流電車の影響などを受けることが判明したが、つくば市内のTX沿線や釜石ではこれは認められなかった。

2013年11月には「土木の日体験教室」で国総研旭本館裏の位置に配置した住宅をサンプルとして公開を行った。その後、テクスチャ表示機能、機種の違い（画面サイズ、縦横比など）による互換性への対応を行った上、2014年3月に、岬地区の復原データを用いて、奥尻島でのテストを行った（写真3-5-1）。

この再現表示のためのデータは、図2-3-13のデータを構成する地面、道路、複数の住宅を一つのデータファイルに集約すると共に、メタファイルを付したものである。

このデータファイルとメタファイルに、携帯端末を用いて表示するために必要なアプリケーションの組を合わせたセットアップ一式を、４種類の携帯端末（タブレット３種、スマートフォン１種）で確認した上で、上記サイトからダウンロードできるように試験的に公開開始した(2014.3)。セットアップ方法、操作方法等に関する解説資料を添えた。

但しアプリケーションはまだ、Android携帯端末の多様な機種、OSバージョンにおける互換性の改善の余地があるため、デバッグ認証の段階である。2014年8~9月に、３機種15台のタブレットを用いて性能テストを行い、精度の高かった５台を使用して10月８日に、奥尻島で小学生を対象とした体験教室を開催した(写真2-1,2-2)。

写真3-5-1 奥尻島岬地区での表示テスト（2014.3）

図 3-5-2 FSMFC 起動画面(左：現地調査、右：記録再生)

④Windows 上のモックアップ FSMFC の作成

この処理系の開発に先だって、デバッグ環境の良い PC 上の Windows－VS2005 で、プログラムの全体構成を検討するために、FSMFC.exe というモックアップを作成した。GPS センサ、磁気センサ、加速度センサが利用できないため、これ自体、実用性は無いが、VC-3M の状態遷移、画面構成等を検討するために有用であった(図 3-5-2)。

また、「見たい場所」として一覧を表示する名称、メタファイル、メタファイル等のリストである ModelIndex.txt の準備段階でのテスト等にも有用である。

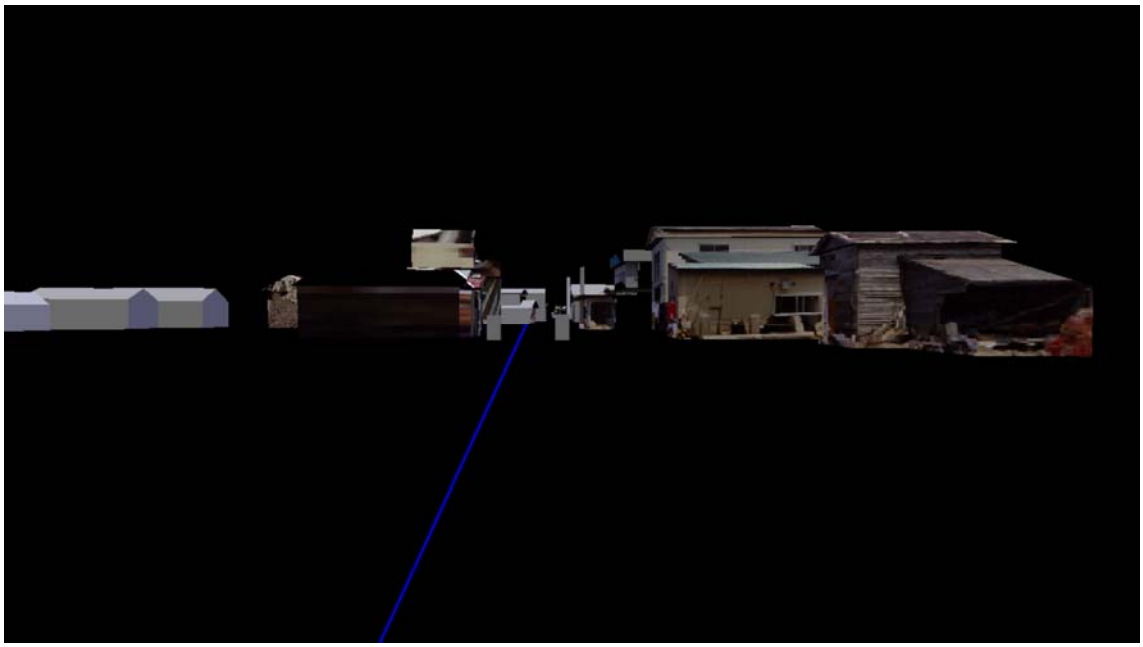


図 3-5-3 FSMFC による表示状態（背景画像はない）

仮想コンバータの処理結果を、景観シミュレータの dml ライブラリを用いてメモリ上に構築し、g3drl ライブラリを用いて描画を行う構成は、VC-2V とほぼ同様である。しかしながら、Android 上で三次元表示に使用できる OpenGL-ES1.0 は、PC 用の OpenGL と比較して、利用できる命令の種類やデータの精度が限られている。利用できない命令に関して

は、利用できる命令を組み合わせて疑似的に構築する必要があった。この疑似的に増補する関数は、OpenGL 関数のプロトタイプを定義しているヘッダファイルの中で直接記述してリンクエラーを回避する方法を採った。

（２）アプリケーションと基幹部分の開発の役割分担とインターフェース関数

Android 系のユーザーインターフェースに関して国総研に開発経験が無かったため、2012 年度の研究予算において、「仮想コンバータの可搬性検証のためのプログラム業務」を一般競争入札によりエム・ソフト社に外注した。同社が作成した部分は、Android SDK の開発環境上で Java 言語を用いて、FSMFC で検討した画面遷移を実現すること、およびユーザーの操作、各種センサの計測値、背面カメラで取得した画像を表示画面の背景として表示することである。プログラムの役割分担を明確化するために、以下のようなインターフェース関数を定義し、エム・ソフト社が作成するのは、インターフェース関数を呼び出すフレームワークとし、インターフェース関数の実行内容は国総研（小林）がプログラムした。このインターフェース関数は全て C 言語で記述し、ndk でビルドし、.so の拡張子を持つ Java アプリのためのダイナミックリンクライブラリとして、フレームワークから起動する方法とした。そのためのインターフェース関数は以下の通りである。

① MOpen(char *logfilename, char *scnfilename)

コンパイラと実行形式の処理結果を記録するログファイルと、ユーザーのシャッター操作を記録するシーンファイルの名称を引数として、システムを初期化する。シーンファイルが存在すれば、それをロードしてセッションの初期条件とする。「歩いた場所」の一覧から、シャッターを押した場所を選択できるようにする。シーンファイルを LSS-S 形式とし、シーンファイルの読み込みにも仮想コンバータを使用することとし、このために、以下のコマンドをライブラリ関数として追加した。

SCN, MODEL, LIGHT, LIGHTGROUP, CAMERA, TIME, EFFECT, EFFECTGROUP

② MInit(int W, int H, int w, int h, double fovy)

表示ディスプレイの横 W 縦 H 寸法、背面カメラが取得する画像の横ｗ縦ｈ寸法、カメラの画角 fovy を引数として、表示エリアを初期化する

③ MLoad(char* metafile, char*datafile, double Ido, double Keido, double Takasa)

メタファイル名、データファイル名、座標原点の緯度経度標高を引数として、データをロードし表示できるようにする。

④ MMove(int Hold, double Ido, double Keido, double Takasa, double R, double P, double Y, double Zoom)

ホールドボタンの状態（キャリブレーションに使用）、GPS で計測された現在の緯度、経度、標高、端末の姿勢を示す３値、ズーム

なお、当初、携帯端末の姿勢を示す R,P,Y にロール（回転）、ピッチ（仰角）、ヨー（方

位角）を用いて、API 関数から得られる値を渡していたが、最新版では、機種による互換性を高めるために、磁気センサの計測値と加速度センサの計測値から直接携帯端末の姿勢を示す四元数を直接計算し、その x,y,z 各成分をこの関数に渡している。

Zoom は背面カメラズームが設定された場合の値を渡すが、本稿時点では簡単のためズームは固定としている(ModelRenderer.java の中で 1.0 に固定)。

2014 年 10 月の「むかしめがね」バージョンでは、GPS がデータを取得するまでの間（Ido, Keido, Takasa が全て 0.0）は、携帯端末の姿勢データだけを用いて、町並全体を対応する向きから画面に対して適切な寸法でプレビューする機能を加えた。

⑤ MShutter(const char *image)

シャッターが押された段階で起動する。その時点の背景画像を API の側で JPEG 形式のファイルに保存し、引数として渡す。関数の中で、現在の視点、表示中のメタファイルとデータファイルの名称などを記録する。この記録は終了時点で、Mobile.ja.scn ファイルに保存される。

⑥ MSelect(const char *imagefilename)

「歩いた場所」を選択する画面において、ユーザーがサムネイル画像の一つを選択して表示を指示した段階で呼び出される。引数で渡された画像ファイルを背景画像として表示し、その上にシャッター時点で記録されたデータファイル、メタファイル、カメラ位置等をもとに、現場で記録した写真合成画面を再生表示する。なおこの記録再生画面は固定であり携帯端末を動かしても変化しない。

⑦ MDelete(const char *imagefilename)

「歩いた場所」を選択する画面で、ユーザーが選択した画像の削除を要求した場合に、その記録を削除する。

⑧ MTerm()

OpenGL の表示画面を終了し、デバイスコンテキストを解放する。

⑨ MClose()

シーンファイル（mobile.ja.scn）を保存し、メモリを解放する。

以上①～⑨の関数をインターフェースとして、役割分担してプログラム作成を進めた。

国総研では、①～⑨の機能を有するトップレベルのソースコード mobile.c を先頭に、これから以下のソースコードにより compile 及び execute 処理を実行すること、及び VC-2V と同様に、景観シミュレータのソースコードから DML ライブラリ及び G3DRL に関するものをビルドに含め、これらを ndk コンパイラを用いて、Android が実行されるマシンの機械語の DLL である、VirtualConverter.so を作成した。

ソースコードの一覧は、メイクファイルである、Android.mk ファイルに列挙されている。

リスト 3-5-1 Android.mk メイクファイル

```
LOCAL_MODULE    := VirtualConverter
LOCAL_SRC_FILES := ¥
```

```
mobile/cci.c¥
mobile/cci_code.c¥
mobile/cci_dml.c¥
mobile/cci_file.c¥
mobile/cci_pars.c¥
mobile/cci_misc.c¥
mobile/cci_tbl.c¥
mobile/cci_tkn.c¥
mobile/cci_face.c¥
mobile/D3dml.c¥
mobile/D3malloc.c¥
mobile/D3mat.c¥
mobile/S3sml.c¥
mobile/S3set.c¥
mobile/S3get.c¥
mobile/S3delete.c¥
mobile/G3drl.c¥
mobile/G3load.c¥
mobile/mobile.c¥
VirtualConverter.c¥
LOCAL_MODULE_FILENAME := libVirtualConverter
LOCAL_LDLIBS := -lGLESv1_CM -ldl -llog
include $(BUILD_SHARED_LIBRARY)
```

これらのビルドは、コンソール(cmd.exe)において、Android.mk ファイルが存在するディレクトリに移動し、

```
ndk-build
```

とタイプすることにより実行される。

なお、この共同作業により一通りの動作を行う処理系を完成した後に、地面の高さの取得、四元数を用いた姿勢制御、テクスチャ付きの画像の表示等を行った。本稿執筆時点のNDK ビルドの構成については、４－５に掲載する。

一方､Java 言語による開発環境 Eclipse 上の AndroidSDK を用いて､ユーザーの操作と、各種センサの値の変化に対応する処理を開発した。

起動から終了に至る遷移に基本的な違いはないが、画面デザイン上の制約から、トップに機能選択メニューを追加し、そこから「現地調査」（アプリ上では、「見たい場所」の選択画面）と、「記録再生」（アプリ上では、「歩いた場所」の選択画面）のどちらを開くかを選択するようにした。更に、歩いた場所については、機械的な文字列（日付時刻等を表す）から選択するのではなく、サムネイル画像を表示してそこから選択できるようにした。

SDK のためのビルドを構成するディレクトリを、リスト 3-5-2 に示す。

リスト 3-5-2 Android SDK のビルドを構成するディレクトリ

```
VC-3M
/.settings ビルドの環境設定等
/assets 空
/bin　VC-3M.apk (ビルドの結果生成するセットアップ)
/gen　BuildConfig.java, R.java
/jni C 言語のソースコード群(.c)
/libs DLL である libVirtualConverter.so
/obj C ソースをコンパイルした結果のオブジェクトファイル群(.o)
/res アイコン画像、操作画面のレイアウト.xml、文字列リソース
/srs この中に、java ソースコードが格納されている
```

java で記述されたソースコードをリスト 3-5-3 に示す。

リスト 3-5-3 java ソースコード一覧

Nativelib.java
VirtualConverterActivity.java
/adapter/GpsValue.java
/adapter/ImageItem.java
/adapter/InfoFileAccessor.java
/adapter/Listitem.java
/adapter/OrientationValue.java
/adapter/SelectImageAdapter/java
/adapter/SelectModelAdapter.java
/base/BaseRenderer.java
/realtime/ModelRenderer.java
/realtime/RealtimeActivity.java
/replay/ReplayActivity.java
/replay/SceneRenderer.java
/selectimage/SelectImageActivity.java
/selectmodel/SelectModelActivity.java
/util/DateUtility.java
/util/DialogUtility.java
/util/FileUtility.java
/util/VirtualConverterErrorDialog.java

前述の ndk でビルドを行い作成した Android のための dll である libVirtualConverter.so を含めたビルドを行い、実機の上にセットアップしてテストする作業を繰り返して完成した。

なお、この java 言語のソースコードに関しても、現在までに姿勢センサで取得した情報を表示系に四元数で渡すための/realtime/quat.java を追加した。

2013 年 1 月に、基本的な動作を確認する所まで到達し、東京都内、およびつくば市内で表示の位置ずれ等の計測を行った。更に、２月には、釜石市内で、表示確認を行った（写真 3-5-2)。

写真 3-5-2：釜石におけるテスト風景(2012.2.17)

表示精度に影響するのは、GPS センサが計測する位置精度、磁気センサが計測する地磁気センサへの局所的な環境条件や携帯端末の帯磁の影響等である。センサの計測値を確認するために、GPS Status という無料アプリ、および受注社が補助的に Android SDK だけで作成し、数値でセンサ計測値を画面に表示する SensorEx.apk という、計測値を直接画面に表示する簡単なアプリを用いて、環境条件の計測を行った。例えば、上野公園内で定点計測すると、直流で送電している地下鉄が通過するだけで、地磁気が大きく変動すること

が判明した。この補助的なアプリは、機能の追加が容易であり、画面の縦横切り替えなどの機種による互換性の向上のためのテストにおいても有効であった。

この改良に際して、携帯端末の姿勢を、SDK の関数が提供するロール・ピッチ・ヨーに加えて、磁気センサと重力センサの生の計測値から VC-3M アプリ側で計算するように修正したが、計測値の履歴をファイルに記録する方法が特に有効であった。

Android 開発環境においても、PC 上の携帯端末エミュレータが利用可能であった。しかし、FSMFC と同様に、屋外で使用する 3 種類のセンサは利用できないため、上記のモックアップとは大差なく、データ転送と起動にも時間がかかるため、本処理系の開発に際しては、あまり有用性は感じられなかった。

機種の互換性の上で問題となったのは以下の点である。

・画面の縦横寸法の取得、現在の縦横の判定、センサの座標軸の判定

・背面カメラから取得した画像と、グラフィックス画面の上下関係

また、センサの物理的な性能を補う上で効果があったのは以下のような点である。

・手振れは、加速度センサの計測値の揺らぎを生じるため、過去の計測値の履歴を反映させたデータを表示用に用いるようにした。具体的には、3 軸の計測値それぞれの値を記憶するスタティック変数を用意しておき、記憶＊0.9＋新値＊0.1 を新たな記憶とし、これを表示に反映させている。このパラメータ 0.1 は、携帯端末の加速度センサの感度や変化検出頻度に依存し、小さすぎると姿勢変化への反応が緩慢となり、大きすぎると手振れに対して過敏となるため、動作テストの結果得られた経験値である。

Eclipse 開発環境においては、ソースコード(java)の修正⇒保管を行うことにより、自動的にリビルドが行われる、とされている。しかしながら、C 言語部分（コンパイラ・インタプリタ処理系、あるいは OpenGL 描画系）の修正を行い、ndk コンパイラによるリビルドを実行した場合には、更新されるのは、DLL である libVirtualConverter.so のみであり、その場合にはアプリケーション全体(.apk ファイル)のリビルドは行われない。そこで、リソースのストリングスにヴァージョンを記入することとし、DLL を更新した場合にはこの文字列を修正することにより、リビルドをトリガーした。

コンパイラ・インタプリタ系の C 言語ソースコード（3－2 で解説した基幹部分）には、コンパイルエラーメッセージなどが直接日本語の文字列で埋め込まれている。ビルドに際して、ソースコードを Shift-JIS 形式で保存すると、Android 系の出力ではエラーメッセージやログファイルが正しく表示されない場合がある。コード変換処理を組み込むことも検討したが、現段階ではコンパイラ・インタプリタ系のソースコードを utf-8 形式で保存すると共に、他の Windows 系利活用処理系においてもこのソースコードを共用することとした。

VS2005 開発環境にあっては、ビルドの設定により、たとえソースコードが utf-8 形式であっても、プログラム中に埋め込まれた UNICODE 形式の日本語のリテラル文字列を、fprintf 等により Shift-JIS 形式で出力する。

一方、メタファイルの中にリテラル文字列として埋め込まれた日本語の文字列の処理は、

メタファイルを保存する形式(Shift-JIS / utf-8)に依存する。PC、タブレット、サーバの間で互換性を維持するために当面、メタファイルは utf-8 形式で保存すると共に、Windows 系 PC の処理系において、Shift-JIS 形式に変換してログファイルに出力するような処理を行っている。

国総研における仮想コンバータのプログラミングと、外注先の業務の役割分担リスト 3-5-4 のように定めた。

リスト 3-5-4 VC-3M プログラム作成における役割分担を定めた開発手順

1　　業務目的

本業務は、国土技術政策総合研究所が開発した、三次元住宅情報の永久保存のために使用する仮想コンバータに関して、その可搬性を検証するために、異なる実行環境に移植するものである。

2　　業務構成

(1)　プログラム開発環境及び実行環境の構築
(2)　表示機能、操作ダイアログの開発
(3)　仮想コンバータの移植
(4)　サンプルデータを用いた動作テスト
(5)　セットアップ一式の作成
(6)　報告書の作成

3　　業務内容

(1)　プログラム開発環境及び実行環境の構築

実行環境は携帯端末であって、以下の性能を有するものとする。

ａ．三次元表示を行うに十分な解像度とカラー数のあるディスプレイを有する
ｂ．三次元モデルを表示するために必要な視点、注視点等に関するデータを、ＧＰＳ装置、地磁気センサー、加速度センサー等により自ら取得できる
ｃ．三次元モデルと合成表示する背景画像を、現場においてリアルタイムで取得するＣＣＤカメラを内蔵している
ｄ．三次元モデルを適切な視点・注視点等の条件設定に従って、立体的に表示するために OpenGL ES ライブラリが利用できる
ｅ．WEB 経由でサーバからファイルをダウンロードできる

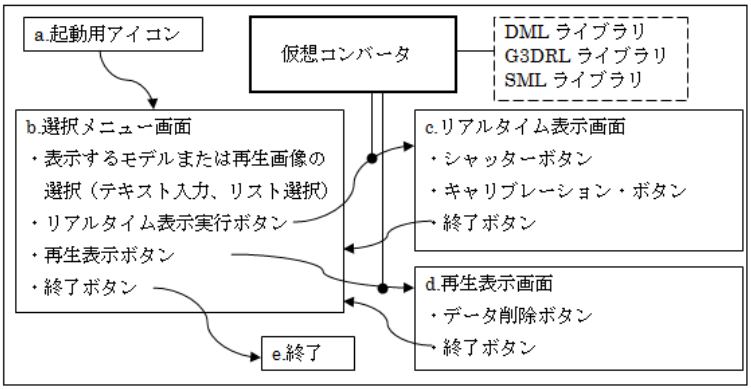
プログラムの作成に使用する開発環境は、無償で公開されているコンパイラ、リンカ、携帯端末エミュレータ等を用いて構成する。この開発環境は、甲においても容易に取得し利用できるものでなければならない。この開発環境は、ユーザーが動作を指示するためのインターフェースおよび端末が有する各種センサーのデータを取得するためのインターフェースにおいては JAVA 言語を、また仮想コンバータの移植のためには、C 言語を使用し、これらを合わせて一体として動作する実行形式を生成する機能を必要とする。

開発環境には、甲が実行形式を無償で自由に配布するライセンスを制限するような有償ライブラリ等を含めてはならない。

なお、動作テストに用いる携帯端末機器は、乙において用意することとし、(4)の動作テストが終了した後、納品することを要しない。

(2)　表示機能、操作ダイアログの開発

システムの全体は、ダイアグラム１に示すような状態遷移を有する。



a. 起動手段

まず、起動用アイコン(a)のタップ操作により選択メニュー画面(b)を表示する。

b. 選択メニュー画面

選択メニュー画面においては、ユーザーが｛①表示すべき住宅等の指定を行う　②リアルタイム表示(c)を指示する　③再生表示(d)を指示する　④終了する(e)｝のいずれかの機能選択を行う。

リアルタイム表示(c)、再生表示(d)が指示された場合、仮想コンバータ(3)に表示すべき三次元データ（ストリーム）と、そのデータ形式を定義するメタファイルを渡し変換を行う。WEB 上のデータとメタファイルが URL で指定された場合には、ダウンロード処理を行った上で、取得したローカル・ファイルを同上の処理対象とする。変換したデータを表示部(c.または d.)に渡し、制御を移す。

c. リアルタイム表示画面

リアルタイム表示画面においては、仮想コンバータの処理結果であるメモリ上の三次元モデルに関して、端末が取得する位置情報を用いて視点・注視点からの透視図を作成し、端末の CCD カメラが取得する背景画像の上に合成表示する。携帯端末が移動し、あるいは回転することにより視点・注視点の変化が検出された場合には、新たな背景画像及び視点・注視点を反映させた画像に更新表示を行う。視点の変更などは、後述の G3DRL ライブラリ関数が機能を提供している。

モデルが、携帯端末の表示画面の視野範囲の外となって表示されない場合には、モデルの中心位置が表示画面のどちら側にあるかを示す矢印等のアイコンを画面に表示し、ユーザーがモデルの位置を確認する作業を支援する。

シャッターボタンが操作された場合には、背景画像を携帯内部のファイルとして保存すると共に、その時点における視点・注視点、時刻情報、表示していたモデル、保存した背景画像のファイル名等を、ＳＭＬライブラリのシーン情報として保存する。この保存は、電源 OFF や OS の再起動により消失しない方法で行う。

キャリブレーション･ボタンは､背景画像の中の目印となる物件との比較対照においてモデルの表示位置が、GPS 位置情報の誤差等の理由により、正しい位置から継続的に一定値だけずれているような場合の補正のために使用する。このような場合に、キャリブレーション・ボタンが押されている間は、GPS 等のセンサーが検出した新たな位置情報をモデルの表示に適用することを停止し、モデルを画面に対して固定的に表示する。ユーザーはこの間に携帯端末の角度を変化させ、あるいは立ち位置を移動することにより、背景画像上でモデルが正しい位置に表示される状態を見出す。正しく表示された状態でキャリブレーション・ボタンを離すと、その時点での視点・注視点と、ホールドされていた視点・注視点との差分を補正データとして取得し、これを以後の視点移動に伴う表示更新に適用する。この差分から補正データを作成し、以後の表示に反映させるロジックは、建築技術的要素を含むため、甲が作成する。乙は、処理に必要な、空の関数、もしくは簡単な処理を行う関数と、必要なスタティック変数等を定義するだけで良い。

終了ボタンが押された場合には、表示を終了し、選択メニュー画面に戻る。

d. 再生表示画面

再生表示画面においては、リアルタイム表示画面(c)におけるシャッター操作により保存されているデータの内、選択画面(a)で指定されたものを、背景とモデルの合成表示として、シャッター操作時の表示画面と同一の静止画像として表示する。

データ削除ボタンが押された場合には、この背景画像ファイルと対応するシーンデータを削除する。

終了ボタンが押された場合には、表示を終了し、選択メニュー画面に戻る。

(3)　仮想コンバータの移植

移植する仮想コンバータのソースコードは C 言語で記述されており、その全体は別添１の通りである。この内、cci_dml.c で定義された組み込み関数群を変換結果出力手段として使用する。更に、別添２に示す DML ライブラリを構成するソースコード（C 言語）をコンパイル・リンクすることにより、変換結果をメモリ上のデータ構造として構築する。次に、G3DRL ライブラリを構成するソースコード（C 言語）をコンパイル・リンクすることにより、OpenGL でモデルを画面に出力することができる。

但し、携帯端末においては、OpenGL ライブラリの機能が一部制約された OpenGL ES を使用することが一般的である。このために必要な対応は甲が行うこととし、乙が G3DRL ライブラリを修正する必要はない。G3DRL ライブラリの多くの機能は、本件の移植には必要でないため、ES にない OpenGL 関数の参照に関してはダミー関数を作成しリンクすることで対応し、処理が必要な場合には本件専用の OpenGL 関数を作成する。可能な限り

G3DRL の変更は行わない。乙が作成する範囲は、OpenGL ES が描画するモデルのパースを、背景画像と合成表示（オーバーレイ）する機能である。

次に、表示画面においてシャッターボタンの操作が行われた場合の処理を行うために、SML ライブラリを構成するソースコード（C 言語）をコンパイル・リンクする。シャッターボタンのコールバックの中で、必要な関数を使用する。

移植に使用する仮想コンバータ、DML ライブラリ、G3DRL ライブラリ、および SML ライブラリのソースコード、および OpenGL ES に関連して追加で必要となる関数等のソースコードは、CD-R の形で契約の後、作業に先立って甲が乙に貸与する。

(4) サンプルデータを用いた動作テスト

動作テストに使用する建物等のサンプルデータは、甲が２種類提供する。それぞれのデータが対象とする建物の所在地は、茨城県つくば市内、及び乙の所在地に近い地区であって、GPS 電波等が良好に受信できるような場所に設定する。サンプルデータは、仮想コンバータが処理することのできる形で、地球座標（ＧＰＳ座標）に関連づけるために十分な情報を含むものとする。

動作確認においては、このサンプルデータの所在地の近傍において、携帯端末によるモデルの表示を行い、正しい位置からの表示のずれ等を計測する。

この動作確認は、GPS の誤差に影響する大気の状態が異なる環境下で少なくとも３回実行する。

動作テストに使用するサンプルデータは、データ形式を記述したメタファイルと共に、CD-R 等の形で契約の後に貸与する。また、動作テストのために、同じファイルを甲が管理するサーバから WEB 経由でダウンロードできるように甲が準備する。

(5) セットアップ一式の作成

作成したプログラムの実行形式を、携帯端末に新たに導入するためのセットアップ手段を作成する。このセットアップ手段の操作方法は必ずしも簡潔である必要はない。むしろ、実行形式がデバッグされ、あるいは改良された場合に、セットアップの再作成が容易であることが望ましい。その手段の実現方法・形態は随意とする。

以上

別添１：仮想コンバータのソースコードのファイル構成（Ver.1.0）

cci.c (43)	コンバータの動作制御（組み込み用には不要）
cci_tkn.c (341)	メタファイルのトークン解析
cci_pars.c (1652)	メタファイルのパーサ
cci_tbl.c (121)	変数テーブル管理
cci_file.c (90)	データファイル入力関数
cci_ip.c (545)	ファイル出力関数（組み込み用には不要）
cci_misc.c (119)	エラー処理等
cci_dml.c (840)	変換結果を DML ライブラリに出力する（本業務で使用）
cci_code.c (810)	仮想マシン上でのコンバータ実行形式の生成と実行
cci_quat.c(391)	クォータニオン演算ライブラリ、地理空間座標変換関数

(　)内はステップ数

別添１は新規に開発した成果である。本業務の成果などを反映して、一般性・移植性を高めるために更に改良する予定である。

別添２：リンクすべきライブラリとソースコード

①DML ライブラリ　メモリ上の三次元データの管理
　d3dml.c(1442)　　各種表示オブジェクトの生成・削除
　d3malloc.c(983)　メモリ管理
　d3mat.c(385)　　座標に関する同次行列演算
②G3DRL ライブラリ　三次元データの表示出力

g3drl.c(5866)	各種オブジェクトの OpenGL ライブラリへの送出・描画
③SML ライブラリ	シーンに関するデータ管理
s3sml.c(681)	初期化・終了
s3set.c(185)	データの作成・設定
s3get.c(140)	データの取得
s3delete.c(259)	データの削除

( )内はステップ数

別添２については、国土技術政策総合研究所　研究報告第 42 号に解説している。同報告付録 CD-ROM に採録している景観シミュレーション・システムのソースコードの一部に含まれるものである。なお、同研究報告及び付録 CD-ROM の内容は、下記の WEB サイトからダウンロードすることができる。

http://www.nilim.go.jp/lab/bcg/siryou/rpn/rpn0042.htm　（本文のみ）

http://sim.nilim.go.jp/MCS/REPORT42/REPORT42.asp　（本文＋付録 CD-ROM）

http://sim.nilim.go.jp/ksim/program/SIM209/SRCNT/LIBRARY/

（３）結果の現場検証

動作を確認するために、東京、筑波および釜石市で動作テストを行った。直流電車（地下鉄含む）の近傍で、磁気が安定しないことがわかっていたため、東京（上野付近）でのテストの結果は芳しくなかった。これに対して、つくば市は、付近を通過する常磐線やつくばエキスプレスが、八郷の地磁気観測所への影響を避けるために交流給電を行っているため、TX の直下でも磁場は安定していた。更に、釜石市においては、ローカル鉄道は電化されておらずかつ運休状態にあり、磁場を発生させるようなアクティビティも低下していたため、姿勢センサの計測値は最も安定していた。

センサ計測値は、シャッターボタンを操作することにより記録されており、報告書にも掲載しているが、この時点では携帯端末の精度が低く、本稿では割愛する。

３－６．ＶＣ－４Ｄの開発

VC-2V および VC-3M においては、仮想コンバータがメタファイルとデータファイルから取得するデータは全てメモリ上に蓄積される。従って、メモリ容量を超えた大きなデータを扱うことはできない。また、電源を喪失した場合には、メモリ上のデータは失われる。

一方、MSSQL, Oracle などのデータベースでは、テーブルの形で登録されたデータは、ハードディスク上のファイルとして記録されており、処理途中での電源障害等に対して強靭な仕組みが用意されている。

メモリ上の三次元データは、基本的には、複数の数値を一定の規則で束ねた構造体の、可変長の配列として組み立てられているため、データベース上のテーブルに変換することが可能である。これにより、電源の問題と、データ規模の問題を解決することができる。個別のデータベースが実体化されたファイル（MSSQL の場合には、mdf という拡張子を有する）のサイズを計測してみると、圧縮アルゴリズムが用いられている。

市販の CAD ソフトや、GIS ソフトの多くは、操作しているユーザーへの応答性の観点から、基本的にはメモリ上に編集中のデータを展開して表示している。しかし、処理速度が求められない長期保存やデータ形式の変換の用途のためには、リクエストを受けてからバッチ処理でデータの分解と、別形式への再構築をバックグランドで実行し、処理が終了してからクライアントに結果を返すような堅牢で確実な処理系ないしサービスが存在すれば有用であると着想し、試作を行った。

WEB ベースで受付と処理結果の納品を行うようなサーバ上のアプリケーションで、以下のような機能を有するものとする。

・アップロード受付画面で、任意形式のメタファイルとデータファイルを受け付ける（アップロード)。
⇒サーバ側では、このペアを仮想コンバータで処理し、データファイルを完全に要素に分解した上で、データベース上のテーブルを構築する。

・ダウンロード受付画面で、データベースの名称と、必要とする（別の）形式のメタファイルを登録する。
⇒サーバ側では、データベース上のテーブルから、メタファイルの指示に基づいて、新たなデータファイルを構築して、クライアントに送付する。

このような処理を実現するためには、メタファイルをコンパイルした実行形式が起動するライブラリ関数が SQL コマンドを生成し、データベース上に頂点座標、面、立体、法線、テクスチャ、属性等のテーブルを構築する。また、構築されたテーブルを参照し、異なるデータ形式を定義したメタファイルにより、異なるデータを出力することができる。

Apache を介して、WEB サーバにおいて動作するアプリから起動し、WEB を介してアクセスしたユーザーに対して、サービスを提供する。

VC-2V、VC-3M と比較すると、読み込みの速度に時間がかかることから、ユーザーからのアップロード要求、ダウンロード要求を受け付けた時点で進行状況を表示するページを立て、ユーザーの進捗状況を伝えるような処理とし 2013 年 3 月に初版が稼働した。

この SQL データベースは、建物や町並などの物件毎のファイルを作成するシステムであるため、メモリ容量による制約を受けることはなく、また処理中の電源障害等に対して強靭である。この処理系は dml ライブラリや g3drl 等、景観シミュレーション関連ソフトからは完全に独立して動作する。このことにより、仮想コンバータの処理系の可搬性を、より高いレベルで検証したことになるであろう。

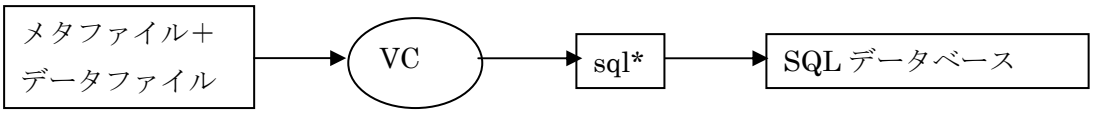
現在は、政府統合サーバへの移行（仮想化）に伴い、OS(Windows2003→2012Server)と SQL データベースの変更（MSSQL→PostgreSQL）の要求に対応して、更に研究開発を続けている。

経時的な使用方法は社会的ニーズに依存するが、基本形は、過去に保存されたデータファイルとメタファイルのペアを、利活用時点で本処理系にアップロードし、直ちに別形式のデータファイルでダウンロードする、データ形式変換サービスである。このようなサービスにより、例えば過去に記録された町並のデータを用いて、避難シミュレーションや延焼シミュレーションを実行可能なデータ形式への変換を行うようなニーズが存在するであろう。

更に、データベース（その実体はサーバー上のファイル）の継続性が制度的に保証されるならば、データの保存方法の一形態として、アップロードからダウンロードまでの一定

期間の保管を行うサービスとしてのニーズも存在する可能性がある。

（入力）



（出力）

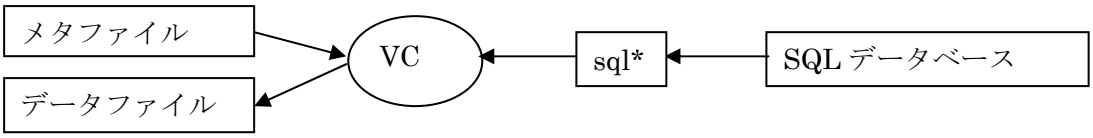


図 3-6-1 VC-4D による処理(*sql は、cci_sql.c によるライブラリ関数の実装)

最終的に作成した WEB アプリは、二つの操作画面を持つ。一つは、データファイルとメタファイルを選択してアップロードし、SQL サーバ上にデータを展開する操作画面である。今一つは、SQL サーバから物件と任意形式のデータ形式のためのメタファイルを指定して、要求した形式のデータのダウンロードを行うための操作画面である。

いずれも、ローカルな端末において選択・指定したファイルがサーバに届けられた段階で、制御が戻る。以後、サーバ側ではバッチ処理としてデータの展開や再構成を行い、終了した段階でユーザーによる結果のダウンロードが可能になる機構となっている。

この処理系を実現するための通過点として、以下のような試作プログラムを段階的に作成した。

（１）VC-4D(dll)（2012.10.11）

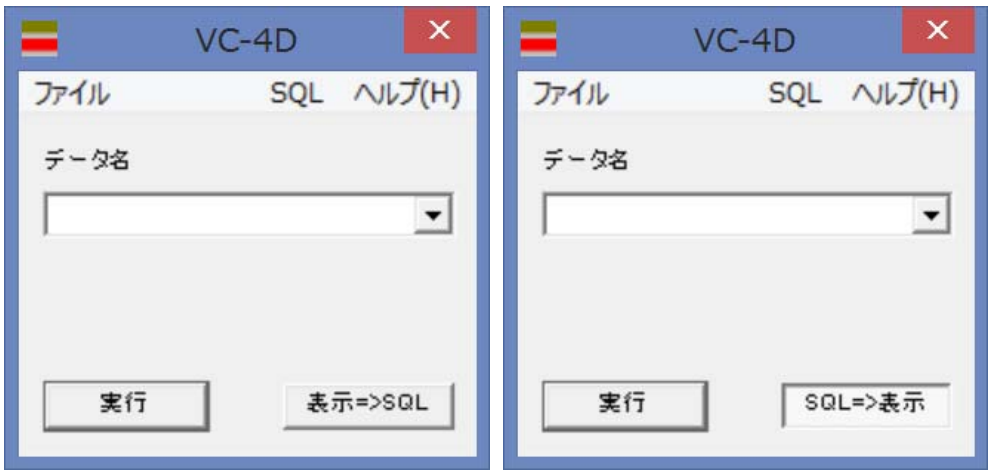


図 3-6-2 VC-4D の画面（SQL データベース入出力のプログラムに使用

表示→SQL では、主画面に現在表示されている建物・町並を、任意名称の一つのデータベースとして記録するとともに、名称を「VC4D」というデータベースの master テーブルに追加登録する（表示→SQL）。

SQL→表示では、登録された名称一覧から選択したデータベースに記録されている建物・町並を取り出して、現在表示されている建物・町並に追加し、画面表示する。

メニューの SQL の下に、以下のメニューを用意した

－ログ表示・・・データベースへの接続、コマンド発行、結果などの履歴を記録
－ログ削除・・・ログをクリア
－データベース設定管理：以下のサブメニューをもつ
　－サーバ設定・・・サーバ、ユーザ、ディレクトリ、ログファイル名を設定する
　－マスタ作成・・・データベースを一覧するテーブルを作成する
　－個別データベース削除・・・選択したデータベースを削除する
　－全データベース削除・・・全てのデータベースと一覧テーブルを削除する
　－データベース集計・・・選択したデータベースの各テーブルのサイズを表示する

アプリの表題は、VC-4D となっているが、仮想コンバータのコンパイラ・インタプリタ処理系は使用せずに、PC 上で SQL の基本的な入出力に関する動作確認を行う関数群を作成するために作成した。メモリ上に展開され表示されているオブジェクトを、SQL テーブルに保管し、あるいは SQL テーブルからメモリ上に取得し表示する。後の段階では、仮想コンバータを組み込んだ処理系が作成した SQL テーブルの内容を表示確認するためのツールとして使用した。

データベース関連の設定を行うダイアログが、メニューの[SQL][データベース設定管理][サーバ設定]から開く、図 3-6-3 に示した SQL サーバ設定画面である。

図 3-6-3 SQL サーバ設定

ソースコードに含まれる SQLDB.cpp の中に、データベースへの接続、SQL コマンドの発行、エラー処理、処理速度の計測実験などを行う関数群を作成した。

データベースとしては、MSSQL または SQLEXPRESS を使用した。

sql サーバにコマンドラインでアクセスするためには、コンソールから、

>sqlcmd –S (local)¥sqlexpress –E

>sqlcmd –S .¥sqlexpress –E

と入力する。接続に成功すれば

1>

という状態で SQL コマンドを受け付ける状態となる。

ここで「exit」と入力すれば、終了する。

>sqlcmd –S .¥sqlexpress –U sa

というオプションでログインすると、パスワードを尋ねてくる。

Windows の C++プログラムからデータベースにアクセスする方法は複数存在する(４－５)。本処理系では、ADO (ActiveX Data Objects) を使用した。接続方法をリスト 3-6-1 に示す。

リスト 3-6-1 VC-4D.exe におけるデータベースの接続

```
１．基本的な変数
_ConnectionPtr m_con;
_RecordsetPtr m_rs;
_variant_t t;

２．開始
CoInitialize(NULL);
m_con.CreateInstance(__uuidof(Connection));
m_con->Open(str,"ユーザ名","パスワード",adConnectionUnspecified);

３．各種処理：SQL コマンド str を以下のように実行する
m_rs = m_con->Execute(str,NULL,adCmdUnknown);
GetFieldValue(m_rs,t,フィールド名);
(t に、データ型に応じた値が格納されているので処理系で利用する)
  データ型は、t.vt の数値で知ることができる。

４．終了処理
m_con->Close();
m_con.Release();
m_con = NULL;
CoUninitialize();
```

この手順は、Windows 上の MSSQL を使用するための特殊な方法である。そこで処理を一般化するために、SQL コマンドを能率的に発行するための関数

pRs Q(const char *format, …);

を用意した。printf(書式文字列,引数リスト)と同様の書法で SQL コマンドを文字列として発行している。戻り値はレコードセット形式であり、これを用いて個々のデータを参照することができる。

以上のような、Windows＋MSSQL に固有の処理を sqllib.cpp に集約し、SQL 文の生成を主な内容とするライブラリ関数の実行を cci_sql.c に、また main 関数を含むソースコードを vc-4d_exe.cpp に分離する方法で、仮想コンバータを構成するソースコードの可搬性に配慮した。

三次元図形を記述する新たなデータベースを作成する際には、createdb 関数を呼び出す。

この関数は以下のような処理を行う。

・必ず存在する master データベースを指定して SQL サーバを開く。

・dbname 変数で指定した名称の新たなデータベースを作成する。

・vc4d データベースの master テーブルに、この新たなデータベース名を追記する。

・dbname 変数で指定した名称の新たなデータベースに、COORD, COLOR, NORMAL, TCOORD, VERTEX, FACE, GRUPPE, LINK のテーブル群を作成する(リスト 3-22)。

・SQL サーバを閉じる。

リスト 3-6-2 データベース作成処理

```
int createdb(char *dbname){//vc4d_exe.cppから参照
          if(SQL_ErrCt) return 0;
          if(SQL_D) path = SQL_D;
          else path = "c:¥¥@keikan¥¥kdb¥¥snapshot";

          DB("master");
          Q("CREATE DATABASE %s ON(NAME=%s_dat, FILENAME='%s¥¥%s.mdf')"
           " LOG ON(NAME=%s_log,FILENAME='%s¥¥%s.ldf')",dbname,dbname,path,dbname,dbname,path,dbname);
          Q("USE vc4d");
          Q("INSERT master VALUES('%s')",dbname);
          Q("USE %s",dbname);
          Q("CREATE TABLE COORD (id int IDENTITY(1,1), X float, Y float, Z float)");
          Q("CREATE TABLE COLOR (id int IDENTITY(1,1), R float, G float, B float, A float)");
          Q("CREATE TABLE NORMAL (id int IDENTITY(1,1), X float, Y float, Z float)");
          Q("CREATE TABLE TCOORD (id int IDENTITY(1,1), X float, Y float)");
          Q("CREATE TABLE VERTEX (Gid int, Fid int, Ix int, Icoord int, Inormal int, Icolor int, Itcoord int,
            Ivirtual int)");
          Q("CREATE TABLE FACE (Gid int, Fid int, idcolor int, idnormal int, idmaterial int, idtexture int,
            SHP int)");
          Q("CREATE TABLE GRUPPE (Gid int, idmaterial int, idtexture int, Attribute varchar(max))");
          Q("CREATE TABLE LINK (id int IDENTITY(1,1), GPid int, GCid int,"
                              "mat0 float, mat1 float, mat2 float, mat3 float,"
                              "mat4 float, mat5 float, mat6 float, mat7 float,"
                              "mat8 float, mat9 float, mat10 float, mat11 float,"
                              "mat12 float, mat13 float, mat14 float, mat15 float)");
          DB(NULL);
          return 1;
}
```

COORDテーブルは、座標値のidと 三次元座標値(浮動小数)を格納する。

COLORテーブルは、カラーのidとRGBAの４数値を格納する。

R(赤)G(緑)B(青)A(不透明度)の各値は、0.0～1.0の間の浮動小数である。

NORMALテーブルは、法線ベクトルのidとXYZの３成分値(浮動小数)を格納する。

TCOORDテーブルは、テクスチャ座標のidとUVの２値(浮動小数)を格納する。

VERTEXテーブルは頂点を登録し、帰属するグループのid、面のid、頂点番号Ix に続き、頂点の属性を座標値Icoord、 法線ベクトルInormal、カラーIcolor、テクスチャ座標Itcoord、仮想線フラグIvirtualの順に各テーブルの参照idで示す。

FACEテーブルは、面を登録し、帰属するグループのid、面の通し番号に続き、面の属性をカラーidcolor、法線ベクトルidnormal、材料idmaterial、テクスチャidtexture、形状（面、折れ線、点群）を示すコードSHPで示す。

面の通し番号は、１から始まり、帰属するグループの中で連続している。

従って、一つの面が複数のグループに帰属することはない。

GRUPPEテーブルは、意味あるまとまり（形状の単位）を記述する。

このGidが、面や頂点の帰属先を示すために参照される。

Gidは開始番号が不定であり、連番で登録されている必要はない。

＊なお、本来であればライブラリ関数名に対応した「GROUP」をテーブル名とするのが明快であるが、SQLサーバにおける予約語として使用不可であったため、方言的名称とした。

LINKテーブルは、複数の意味あるまとまりの親子関係を記述する。

関係のid、親グループのid、子グループのid、相対的な位置関係を記述する同次行列**の16成分(浮動小数)を登録する。

**同次変換行列、同次座標変換行列、斉次行列、homogeneous transformation matrix

この親－子のペアは、例えば団地と住宅、建物と部材等の関係である。

上記のように、VC-4Dにおいては、例えば頂点を格納するVERTEXテーブルを面毎に作成するのではなく、一つの記録に対応するデータベースに属する全ての頂点を一つのテーブルに格納するとともに、個々の頂点にはそれが帰属するグループと面のIDを登録している。ある面に属する頂点を取り出すためには、SQLコマンドのSELECT文で立体と面のidを絞り込む。

リスト3-6-3 データベース削除処理

```
void deletedb(char *dbname){
        DB("master");
        Q("DROP DATABASE %s",dbname);
        Q("USE vc4d DELETE FROM master WHERE name='%s'", dbname);
        DB(NULL);
}
```

データベースを削除する場合には、deletedb 関数を呼び出す。この関数は、dbname で指定した名称のデータベースを削除すると共に、vc4d データベースの master テーブルからこのデータベース名を削除する。

仮想コンバータの機械語の実行時にライブラリ関数が呼び出されると、cci_sql.cで定義した処理が実行される。例えばCOORD関数が実行されるとVC-4Dにおいては、リスト3-24に示したようなcoord関数を起動する。この実装では、データベースのcoordテーブルから、引数のXYZ座標値が一致するレコードを探し、存在すればそのidを返す。存在しなければ、新たなレコードを追加しそのidを返す。

リスト 3-6-4 VC-4D における COORD 関数による頂点座標の登録処理

```
int coord(double v[3]){//一致するものがあればそのIDを、なければ追記し新しいIDを、返す（目標）
        int susun;
        Sel(hS);
        susun = TABLESIZE("coord WHERE X = %lf and Y = %lf and Z = %lf",v[0],v[1],v[2]);
        if(0 < susun){
                Q("SELECT id FROM coord WHERE X = %lf and Y = %lf and Z = %lf",v[0],v[1],v[2]);
                return Vint("id");
        }
        Q("INSERT coord VALUES(%lf,%lf,%lf)",v[0],v[1],v[2]);//
        Q("SELECT IDENT_CURRENT('coord') as hasil");
        return Vint("hasil");
}
```

（２）VC-4D(win)（2012.10.24）

上記(1)のように景観シミュレータに依存せずに、Windows 上で単独でダイアログを表示し動作するスタンドアロンの実行形式である。初期の開発段階では、この実行形式の中で直接データベースの作成や読み出しを実行していたが、最終段階では、③で解説するサーバー用のコンソール・アプリを、この処理系から別プロセスとして起動する方法とし、③の開発におけるテスト・デバッグのためのフレームとして使用した。

図 3-6-4 VC-4Dwin の初期画面

この試作プログラムは、既存の形状記述ファイル（データファイル）と形式定義ファイル（メタファイル）をそれぞれファイル・ダイアログで選択し、任意の新しいデータベース名を記入してアップロードボタンを押すと、データベース名の SQL データベースを構築する Windows アプリケーションであり、単独で使用することができる。

既存のデータベース名をドロップボックスで選択し、形式定義ファイル（メタファイル）を選択ボタンから開くファイル・ダイアログで選択し、形状記述ファイルに任意の新しいファイル名を記入してダウンロードボタンを押すと、メタファイルで指定した形式で、形状記述ファイル欄に指定した名称のファイルを生成する。

左上のアイコンをクリックすると、バージョン情報のダイアログが開く。このダイアログの中で、図 3-20 とほぼ同様に、使用する SQL サーバ名、ユーザー名、パスワード、ディレクトリ、実行形式、ログファイル名を指定する。

パスワードは、暗号化されたバイナリファイルに記録する。このファイルは、サーバー上にセットアップする際に使用することができる。

ディレクトリは、データベースを新たに作成する際にファイル(.mdf, .ldf)を作成するディレクトリである。

実行形式は、データベースの入出力のために起動する別プロセスの実行形式④である。

（３）VC-4D.exe(2012.12.26)

サーバー上で動作するコンソールアプリであり、引数としてデータファイル、メタファ

イル、SQL 名称を受け取り、SQL データベースを構築する。

また、引数として指定されたメタファイルと SQL 名称から、指定した名称のデータファイルを再構築する。入力か出力かは引数として渡されるフラグで識別する。

サーバー側の WEB アプリケーションの処理ロジックは java でプログラムされており、ユーザーから必要な情報が取得された後に、実行形式 vc-4d.exe が起動されて、メタファイルのコンパイルを行い、SQL データベースの入出力を行う。

main 関数をリスト 3-6-5 に示す。

リスト 3-6-5 VC-4D.exe の main 関数

```
int main(int argc, _TCHAR* argv[], char*env[])
{
	char *metafile;
	char *datafile;
	char *sqlname;
	char *logfile;
	char *p;
	int IO;
	int hasil;

	showenv(env);
	if(argc<5) return 2;
	metafile = q(argv[1]);
	datafile = q(argv[2]);
	sqlname = q(argv[3]);
	IO = atoi(q(argv[4]));
	if(IO<0) return 2;
	if(2<IO) return 2;
	/*ログファイル名称の指定、無ければデフォルト名を使用*/
	if(5<argc) logfile = q(argv[5]);
	else logfile = q("Slog.geo");
	p = strrchr(logfile,'¥¥');
	if(p) p++;
	else p = logfile;
	if(!strcmp(p,"result.txt"))
		strcpy(p,"tluser.txt");/*ログ名result.txtが要求*/
	/*以下の戻り値は、それぞれの処理の中で決定(0-)*/
	if(IO==1) hasil = upload(metafile,datafile,sqlname,logfile);
	else hasil = download(metafile,datafile,sqlname,logfile);
	summary(logfile,hasil);
	return hasil;
}
```

アップロードとダウンロードは同じ実行形式が行っている。入出力は、第 4 引数で区別し、upload 関数(リスト 3-6-6) か download 関数(リスト 3-6-7)に制御を渡している。

リスト 3-6-6 VC-4D.exe の upload 関数

```
int upload(const char*m_MetaFile, const char*m_DataFile, const char*dbname, const char *logfile){
	int i,hasil;
	char debugcommand[240];

	ReadOnly = 0;
	SQL_ErrCt = 0;
//	fopen_s(&stberr,"Slog.geo","wt");
	fopen_s(&stberr,logfile,"wt");
```

```
            if(!stberr) return 2;//ログファイルが開かない
            fprintf(stberr,"#upload logfile:%s¥n",logfile);
            fprintf(stberr,"#metafile:%s¥n",m_MetaFile);
            fprintf(stberr,"#datafile:%s¥n",m_DataFile);
            fprintf(stberr,"#database:%s¥n",dbname);
            sprintf_s(debugcommand,240,"notepad.exe %s",logfile);
            errorlog = stberr;

            i3UserInfo *ui;
            char Usedb[80];

            ui = Makelist();
            for(i=0;i<ui->count;i++){
                        if(!strcmp(ui->keywords[i],dbname)) break;
            }
            if(i < ui->count){
                        fprintf(stberr,"#既存のDB(%s)に上書きupload¥n",dbname);
                        deletedb((char*)dbname);
            }
            if(!createdb((char*)dbname)){
                        fprintf(stberr,"#DB(%s)作成失敗。終了する",dbname);
                        goto end;
            }
            DB((char*)dbname);
            sprintf_s(Usedb,80,"USE %s",dbname);
            usedb = Usedb;

            InitProgressFILE();//インジケータ初期化(0を初期値とする)

            fprintf(stberr,"#C--コンパイラ　ログ¥n");
            SetRootGroup(0);//cci_sql の面、頂点リストに係るメモリブロック、カウンタ等初期化
            HAJIME();//cci 初期化

            if (compile((char*)m_MetaFile)) {//argv[1]はメタファイル
              fprintf(stberr,"#コンパイル成功(%d行=>%d行)¥n",srcLineno,codeCt);
              if (strncmp(m_DataFile,"--code",6)==0){//入力ファイル名の冒頭６文字
                          code_dump();
              }else{
                          fprintf(stberr,"#---------------実行開始--------------¥n");
                        hasil = execute((char*)m_DataFile);
                        untable();
                        fprintf(stberr,"#---------------実行終了--------------¥n");
                        fprintf(stberr,"exe_ret_code=%d¥n",hasil);
                        reset();//メモリ解放
              }
            }else{
              fprintf(stberr,"#コンパイル失敗¥n");
            }
end:
            fprintf(stberr,"SQL_ErrCt=%d¥n",SQL_ErrCt);
            fprintf(stberr,"err_ct=%d¥n",err_ct);
            fprintf(stberr,"exe_err_ct=%d¥n",exe_err_ct);
            DB(NULL);
            fclose(stberr);
            reset();//cci_sql メモリブロック等の解放
            Freelist(ui);
//リターンコード生成
            if(0 < SQL_ErrCt) return 3;//SQLエラー
            if(0 < err_ct) return 4;//コンパイル・エラー
            if(0 < exe_err_ct) return 5;//ランタイム・エラー
            switch(hasil){
                        case 0:
                        case 1:
```

```
                                return hasil;
                        default:
                                return 6;
        }
}
```

リスト3-6-7 VC-4D.exeのdownload関数

```
int download(const char*m_MetaFile, const char*m_DataFile, const char*dbname, const char *logfile){
        int hasil;
        char debugcommand[240];
        char Usedb[80];

        ReadOnly = 1;
        fopen_s(&stberr,logfile,"wt");
        if(!stberr) return 2;//ログファイルが開かない
        fprintf(stberr,"#download logfile:%s¥n",logfile);
        sprintf_s(debugcommand,240,"notepad.exe %s",logfile);
        errorlog = stberr;
        if(!strncmp(m_DataFile,"--code",6)) goto dump;
        fopen_s(&outfile,m_DataFile,"wt");
        if(!outfile){
                fprintf(stberr,"#outfile(%s)が開けない¥n",m_DataFile);
                hasil = 2;
                goto end;
        }

        SQL_ErrCt = 0;
        DBname = NULL;
        DB((char*)dbname);
        sprintf_s(Usedb,80,"USE %s",dbname);
        usedb = Usedb;
        if(SQL_ErrCt){
                hasil = 3;
                goto end;
        }
        InitProgressSQL();//インジケータ初期化(面の総数を初期値とする)

        fprintf(stberr,"#C--コンパイラ　ログ¥n");
        SetRootGroup(0);//cci_sql の面、頂点リストに係るメモリブロック、カウンタ等初期化
        HAJIME();//cci 初期化
dump:
        if (compile((char*)m_MetaFile)) {//argv[1]はメタファイル
          fprintf(stberr,"#コンパイル成功(%d行=>%d行)¥n",srcLineno,codeCt);
          if (strncmp(m_DataFile,"--code",6)==0){//入力ファイル名の冒頭６文字
                  code_dump();
          }else{
                fprintf(stberr,"#---------------実行開始--------------¥n");
                fprintf(stberr,"#DB(%s)からoutfile[%s] に出力する¥n",dbname,m_DataFile);
                hasil = execute(NULL);//入力ファイルはなし
                untable();
                fprintf(stberr,"#---------------実行終了--------------¥n");
                fprintf(stberr,"exe_ret_code=%d¥n",hasil);
                reset();//メモリ解放
          }
        }else{
          fprintf(stberr,"#コンパイル失敗¥n");
        }
end:
        fprintf(stberr,"SQL_ErrCt=%d¥n",SQL_ErrCt);
        fprintf(stberr,"err_ct=%d¥n",err_ct);
        fprintf(stberr,"exe_err_ct=%d¥n",exe_err_ct);
        DB(NULL);
```

```
        if(outfile) fclose(outfile);
        if(stberr) fclose(stberr);
        reset();//cci_sql メモリブロック等の解放

//リターンコード
        if(0 < SQL_ErrCt) return 3;//SQLエラー
        if(0 < err_ct) return 4;//コンパイル・エラー
        if(0 < exe_err_ct) return 5;//ランタイム・エラー
        switch(hasil){
                case 0:
                case 1:
                        return hasil;
                default:
                        return 6;/*メタファイルによる異常終了報告：コードは、ログファイル参照*/
        }
}
```

download 処理においては、出力形式を定義するメタファイルに従って、データファイルとして名称が指定されたファイルに書き込みを行う。このとき、保存データファイルは既に解読されて SQL サーバのテーブルとして展開され蓄積されている。メタファイルは、このテーブル群にアクセスしてデータを取り出し、これを用いて別形式のファイルを出力する。

この処理を行うために、テーブル群にアクセスするための出力系ライブラリ関数群を増補した。これらは、保存ファイルに添付するメタファイルにおいて利用されることはない。従って、利活用段階でこれらの出力系ライブラリ関数を更に増補し、あるいは目的に応じて動作を変更しても、保存ファイルの解読処理に影響を及ぼすことはない。保存時点から年数が経過した後の、利活用時点における処理系に属する機能である。これらの内部処理に関しては３－７で、また別形式での出力用に作成する第二のメタファイルにおける参照方法に関しては４－２(3)で解説する。

（４）WEB アプリケーション「三次元データ保管庫」

以上の段階で開発した、メタファイルの記述に従いデータファイルを処理し、SQL サーバへの入出力を用いる実行形式（エンジン部分）を、ユーザーからのリクエストに基づき起動し、得られた結果を返送する WEB アプリケーションを「三次元データ保管庫」としてとりまとめた（２－７）。この処理系においては、Java で記述され稼働しているサーバ側のアプリから、必要に応じて VC-4D.exe が起動され、引数として渡されたデータファイル名、メタファイル名、およびログファイル名等の情報に基づき処理を行う。終了後にサーバーアプリに制御が戻される。このような前記 VC-3M と同様の国総研と受注者マックスネットコンサルティング株式会社との間での業務の切り分けにより、インターフェース境界をめぐる支障を避けつつ、同時平行的にプログラム、デバッグの作業を進めることができた。

３－７．出力系のライブラリ関数の開発と出力系メタファイルの作成

VC-4D の特徴は、一度 SQL サーバー上に構築された三次元データを系統的に取り出し、ユーザーがリクエストした形式に再編成してデータファイルに出力する工程のための機能である。このために、データベース上のオブジェクトにアクセスするためのライブラリ関

数群を用意した。ダウンロードのリクエストと共に添付した第二のメタファイルは、これらのライブラリ関数を呼び出し、取得した数値などを printf 等の組込関数に渡して、求められたデータファイルの出力を行う。

これらの関数は、①VC-1C においては、全て何もしない関数として、cci_ip の中でダミー定義している。②VC-2V においては、同じ仕様で dml メモリ空間から形状や属性を取り出す処理を記述した。

これらの出力系ライブラリ関数を用いたファイル出力の形式定義は、利活用局面において使用するものであるため、データファイルの長期保存に際して添付するメタファイルの中で必ずしも使用しなくともよい。

int G(); 全てのグループを順にアクセスし、ID を返す。終了時はゼロを返す。
int Gattribute(); アクセスされているグループの文字列を登録し、メモリアドレスを返す。

リスト 3-7-1 全てのグループへの順次アクセス

```
int gid,fid;
while(gid=G()){//全てのグループに順次アクセスする
  //ここに、選択されているグループに関する処理を記述する
  d = Gattribute();//グループに設定されている文字列をメモリ上に取得しアドレスを返す
  //ここに、属性を用いた出力処理を記述する
  while(fid=F()){//選択されたグループに属する全ての面に順次アクセスする
    //ここに、選択されている面に関する処理を記述する
    while(vid=V()){//選択された面に属する全ての頂点に順次アクセスする
    }
  }
}
```

仮想現実を記述する VRML 形式や、建築物や機械部品の CAD データを保存する DXF 形式において、予め定義された部品やシンボルが繰り返し位置や向きを変えて配置されているシーングラフ型のデータ構造の場合には、リスト 3-7-2 のようにアクセスする。

リスト 3-7-2 全ての表示グループへの順次アクセス

```
int did;
while(did=D()){
  //ここに、選択されている表示グループに関する処理を記述する
}
```

例えば、２０本の同じネジを使った家具が、6 個配置されているとすると、ネジは全体で 120 本存在する。この時、ネジの形状を定義したグループは一つで足りる。このような空間構成に対して、D()関数を使いアクセスすることにより、120 回配置されている同一の

ネジのそれぞれの位置と向きを取得することができる。

この時、親子関係を示すリンクテーブルには、２６の位置関係が定義されている。そのうち６は、６の家具の位置に関する情報であり、２０は、一つの家具の中に用いられているネジの位置に関する情報である。このような位置情報にアクセスするためには、L()関数を使用する。

また、この時、グループは二つだけ存在し、一つは家具の形状を示すグループ、もう一つはネジの形状を示すグループである。上記の G()関数はこの二つのグループを取得する。

D（）関数を用いることにより、表示されている全てのオブジェクトを再構成することができ、STL 形式のような模型加工のデータを生成することができる。G（）関数と L（）関数を用いることにより、冗長性を避けた最小限の形状定義によるデータを再構成することができる。

データアクセスに関する関数は、記録保存段階では必要ではなく、利活用段階において必要となる機能である。しかしながら、記録保存段階で、記録保存方法が有効であることを検証（ベリファイ）する手段としては有用である。

ここでは、出力系のライブラリ関数の仕様と内部処理について解説し、メタファイルからの呼び出し等の用法については、４－２（３）で解説する。

（１）出力系のライブラリ関数

出力系のライブラリ関数はまず VC-4D において、構築済のデータベースのテーブル要素を読み出す機能として実装した。内部処理においては、SQL 文を発行し得られたレコードセットに順次アクセスするような構成となっている。

具体的には、それぞれの階層のオブジェクト（立体、面、頂点等）が複数存在する場合に、そのオブジェクト群（データベースにおけるレコードセットの概念に相当する）の中のオブジェクトを先頭から順次選択し、残りのオブジェクト数を戻り値として返す整数型のオブジェクト関数を基本形とした。更に、現在選択されているオブジェクト（例えば立体）にカーソルが置かれた状態で、そのオブジェクトの構成要素（例えば面）を、別の関数を用いて同じような手順で順次アクセスする。

さらに、現在選択されているオブジェクト（例えば頂点）の様々なパラメータ（例えば座標値や色彩）を数値として取り出すライブラリ関数を用意した。

これらのライブラリ関数が、SQL データベースに保存データの解読結果を蓄積する VC-4D の上で正常に動作し、ファイル出力処理を記述するために有用であることを確認した上で、出力系のライブラリ関数が同じようにメモリ上に保存データの解読結果を蓄積し３D 表示に使用する VC-2V においても同様の手順でデータを読み出すように cci_dml においても実装した。その際には、オブジェクトへのアクセス順序戻り値の定義に関して一貫性を保つための検証と調整を行い、互換性を確保した。これにより、同じメタファイルを使用して、二つの経路で保存データを対象としたファイル変換を行うことができる。

保存データ＋メタファイル→VC-4D による解読（DB）→別形式の出力

保存データ＋メタファイル→VC-2V による解読（表示）→別形式の出力

これにより、同じ保存データとメタファイルから出発し、この別形式の出力に際して、同じ第二の出力用メタファイルが二種類の異なる処理系である VC-4D 上と VC-2V 上で同じファイルを出力するような結果が得られた。以下、各関数に関して解説する。

① G()関数

全ての立体に順次アクセスするために使用する。

初回：立体の総数 nG をカウントし、最初の立体にカーソルを当てる。 次回：次の立体にカーソルを移動する。 終了：次の立体が無い場合、初期化する。 戻り値：選択された立体を含む残りの立体数（初回は総数)。

これにより、出力系メタファイルにおいて、リスト 3-7-1 に示したようなループ処理が簡単に書ける。

派生的な関数として、選択された（カーソルの当たっている）立体の ID 値を取り出す Gid （）関数、属性を取り出す Gattribute()関数を用意した。

② F()関数

選択された立体に属する全ての面に順次アクセスする

初回：面の総数ｎF をカウントし、最初の面にカーソルを当てる。 次回：次の面にカーソルを移動する。 終了：次の面が無い場合、初期化する。 戻り値：面の ID（1～N)またはゼロ（終了）

これにより、出力系メタファイルにおいて

```
while(F()){
   処理
}
```

というループが簡単に書ける。

選択されている面の属性を取得する関数として、Fshp(), Ftexture(), Fmaterial(), Fcolor(), Fnormal()を用意した

③ V()関数

カーソル面に属する全ての頂点に順次アクセスする

初回：頂点の総数ｎF をカウントし、最初の面にカーソルを当てる。 次回：次の頂点にカーソルを移動する。 終了：次の頂点が無い場合、初期化する。 戻り値：面の ID（1～N)またはゼロ（終了）

これにより、出力系メタファイルにおいて

```
while(V()){
  処理
}
```

というループが簡単に書ける。

派生的な関数として、Vcoord,Vcolor,Vnormal,Vtexture を用意した

更にその下の派生関数として、Sx,Sy,Sz,Sq,Cr,Cg,Cb,Ca,Cq,Nx,Ny,Nz,Nq,Tx,Ty,Tq を用意した。

④ F3()関数

カーソル面を三角形分割し順次アクセスする。

三角形分割された後の面の総数を nF3 に保存する。

三角形分割が行われた場合(0<nF3)、V_()コマンドは、分割三角形の 3 頂点に順次アクセスする。

⑤ S_()コマンド

全ての頂点座標に順次アクセスする。オブジェクト毎に頂点座標や面を出力するのではなく、予め全ての頂点座標を定義しておいて、それに基づいて面や立体を構成するようなファイル形式での出力を行う処理のために用意した。

⑥ N_()コマンド

全ての法線ベクトルに順次アクセスする。

⑦ T_()コマンド

全てのテクスチャ座標に順次アクセスする

⑧ C_()コマンド

全てのカラーに順次アクセスする

⑤～⑧のライブラリ関数は、立体－面－頂点の構造や選択状況に関わりなく、各テーブルに登録された全てのレコードに順次アクセスする。これらの処理は、SQL データベース上に展開されたデータに関しては、一つへのテーブルへの順次アクセスとして実装が容易であるが、cci_dml において同一機能を実現するためには工夫が必要となった。具体的には(時間はかかるが) 初回アクセスで、全空間の例えば⑧の場合、カラーを検索しリスト化し、先頭にカーソルを置く。二度目からはリストの次の要素にアクセスし、終端に到達した段階で、リストを解放する。

このリストとしては、例えば⑧の場合、データファイルの読み込み段階でライブラリ関数 COLOR(・・・);が実行される度に成長するリストを再利用することができる。しかし、VC-2V 処理系においては、読み込み処理が終了した段階で構築された配列は一度解放されているため、C()関数が呼び出される出力処理段階では、初回アクセスで、メモリ上の全ての面や頂点に定義されたカラーからこの配列を再構築している。その際にソーティングを行っている。

（２） 凹ポリゴン

面が凹ポリゴンである場合、面の種別は通常のポリゴン(D3_SHP_FACE=0)ではなく、凹ポリゴン(D3_SHP_CONCAVE=2)の属性が SHP メンバとして付与される。

面の SHP メンバは､IP の FACE コマンドの中で明示的に面毎に指示されることはなく、CONCAVE(ON); CONCAVE(OFF); のスイッチにより、FACE コマンドが発行された時点でのコンテキストとして設定される。

DML,DRL においては、F->shp の属性を見て、図形演算や描画の方法を決定している。

従って、入力に際しては、FACE コマンドを実行する前に、その面の凹凸を判定して、必要な CONCAVE コマンドを直前に発行する。

LSSG 出力に際しては、FACE コマンドを発行する前に凹ポリゴンであれば、その面の凹凸を判定して、CONCAVE コマンドを直前に発行する。

SQL においては、FACE テーブルの第七列として、SHP 属性を保存している。

因みに、凸面(0)と凹面(2)以外に、線(1)と点群(3)の属性があり、これらは同じく面のテーブルに保存される。また、（４）で解説する穴あきポリゴンの場合には、必ず凹(2)となる。

（３） VERTEX コマンドとポイントの扱い

メタファイルにおけるライブラリ関数である VERTEX 関数は、LSSG 形式における VERTEX コマンドとの互換性を保つために､以下の特殊な引数の与え方を可能としている。

・フルスペック：５個の引数

VERTEX(V,N,T,C,Va);

V：頂点名称

N：法線名称

T: テクスチャ座標名称

C：カラーを定義名称

Va：仮想線を定義する頂点名称

ここで言う名称とは、仮想コンバータにおいてはメタファイル中の変数名である。

実行時の VERTEX 関数の内部処理においては､この変数に代入された ID が使用される。

データファイルに使用された変数名を変数として使用したい場合には、

scanf 関数等で、データファイルの変数を記号表に登録し、そのアドレスをメタファイルの変数に取得する。VERTEX 関数の引数には、「＊メタファイル変数」の表現により、

データファイルの変数に代入されている値を使用する。

なお、景観シミュレータの歴史的経緯により、内部処理における D3_VA_XXX の定義や d3Vertex 構造体では、以下のような順が用いられており、外部ファイルの VERTEX の引数順とは異なっている点には公開ソースコードのライブラリ関数を応用する際に注意を要する。外部ファイルの順序を変えると、これまで作成した大量のファイルが読めなくなる。また､内部構造体の順序を変えると､プラグイン.dll 等の処理に不具合が生じる恐れがある。

リスト 3-7-3 景観シミュレータの dml ライブラリにおける VERTEX 構造体定義

```
 (d3dml.h)
#define D3_VA_NORMAL        0x0001
#define D3_VA_COLOR         0x0002 /*何故か、IP,struct と異なり、texture の前にある*/
#define D3_VA_TEXTURE       0x0004
#define D3_VA_VIRTUAL       0x0008 /*990115 DR.H.K. sifat pulau lobang*/

struct _d3Vertex {
        unsigned long va;
        double v[3];
        float n[3];
        float t[2];
        float c[4]; /*この順序は外部関数と同様である。*/
};
```

（省略形について）

頂点座標は必須であるが、それ以外の要素は省略可能である。

その場合、省略された項目は空欄とすることができる。

[例]VERTEX(P1,,,P2);

更に、ある項目以降が全て空欄である場合には、省略することができる。

[例]VERTEX(P);　VERTEX(P,,T);

内部処理においては、空欄が入力された項目は-1 として処理されている。

（４）穴あきポリゴンと仮想線

景観シミュレータ(1996-)においては、穴あき図形は仮想線により表現した。

この方法の長所は、OpenGL を用いた三次元画像表示において、通常の凹ポリゴンと同じ方法で穴のあいた面がそのまま高速描画できる点にある。

つまり、三角形の中に三角形の穴が抜けた図形は、２辺が重なる８角形と等価である。

但し、頂点や辺を集計する場合には、幾何学的に意味のない便宜的に設定された仮想線は除外する必要がある。またワイヤーフレームの表示を行う場合には、仮想線は表示しない方が明快である。このような場合に明示的な違いが生じる。

穴あきポリゴンの表現方法は二次元システムにも共通する問題である。GIS 系のデータ形式においては、外側を親、中の穴を子とする二層のデータ構造を有する場合が多い。漢字等の形状を記録するフォントファイルも二層のデータ構造となっている。その場合には、仮想線による表現に変換するか、三角形分割しなければ OpenGL 等のグラフィックライブラリで表示できない。

この変換ないし分割の方法は一意的には定まらない。また、担当エリアの入力オペレータの操作を反映して、複数の独立した領域が、それぞれ独立した図形となっていたり連結した図形となっていたり、回る向きが反転していたり一貫しない場合も見受けられる。このような幾何的に不正なデータであっても、地図などの二次元の線画として表示する場合には同じ表示結果が一応得られる。

これに対して逆に、仮想線による表現を二層構造（外周＋穴）に変換する処理は、一意的に行うことができる。

△P0-P1-P2 と、穴△P3-P4-P5 の間を、仮想線 P0-P3 が繋いでいる場合、

P0 の第五引数に P3、P3 の第五引数に P0 を設定する。

面は、頂点列 P0-P1-P2-P0-P3-P4-P5-P3 として定義する。

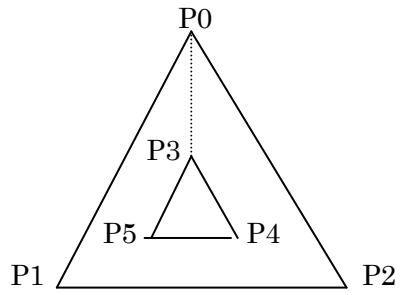


図 3-7-1 穴あきポリゴンの表現

面の描画においては、通常の凹ポリゴンとして P0-P1-P2-P0-P3-P4-P5 を描く
ワイヤーフレーム描画においては、P0-P3 と、P3-P0 のラインの描画を省略する。
dml 空間においては、面には長さ８の d3Vertex 構造体の配列が割り付けられ、次の頂点との間が仮想線である場合、配列 d3Vertex の va メンバにビットフラグが建っている。

リスト 3-7-4 インタープリタによる仮想線をもつ面の出力処理

```
#define D3_VA_NORMAL        0x0001
#define D3_VA_COLOR         0x0002
#define D3_VA_TEXTURE       0x0004
#define D3_VA_VIRTUAL       0x0008
```

LSS-G 形式の外部ファイルとの入出力を行うインタープリタ上の ip 空間においては、DML 表示用８頂点の内、重複している２頂点の出力を１頂点名称に統一した上で６頂点のみを出力し、頂点の属性として仮想線の先の頂点を属性として表示する。

リスト 3-7-5 仮想線のある面の出力結果

```
P00 = VERTEX(V00, , , C00, V03);
P01 = VERTEX(V01, , , C01);
P02 = VERTEX(V02, , , C02);
P03 = VERTEX(V03, , , C04, V00);
P04 = VERTEX(V04, , , C05);
P05 = VERTEX(V05, , , C06);
F00 = FACE(P00, P01, P02, P00, P03, P04, P05, P03);
```

言い換えると、６の頂点を有する穴あき面の頂点列

（外形）P0-P1-P2（穴）P3-P4-P5 を、

P0(P3)-P1-P2-P0(P3)-P3(P0)-P4-P5-P3(P0)

と表現する。(　)は、ワイヤーフレーム表示や線・頂点の集計において省略する同一頂点から発する仮想線の先の頂点を示している。

面を構築する FACE 関数では、P0(P3)-P1 は通常のワイヤーとして扱い、P0(P3)-P3(P0) は、ワイヤーを省略するように面を構成する。

面を構成する頂点を順次取得しファイル出力する、景観シミュレータの i3_outputFace 関数(i3ipoutput.c)では、８の表示用頂点と、６の座標値を出力するために、座標値の比較

を行っていた（リスト 4-5-15）。本処理系では、座標値を登録する段階でソーティングが行われているため、V 関数で８の表示用頂点が順次取得されるが、それらが参照する頂点座標の数は６である。

リスト 3-7-6 景観シミュレータのインタープリタにおける出力部分

（頂点座標関連以外の、カラーや法線等の処理を省略する）

```
int i3_outputFace(FILE *fp,d3Face *f,int type)
{
  int i,j,*nomor,nC,nN,*nomC,*nomN;
  d3Vertex *vl,*vlasli;

  nomor = d3Malloc(f->vnum * sizeof(int) );
 if(OPTIM & I3_OPT_N) nomN = d3Malloc(f->vnum * sizeof(int) );
 if(OPTIM & I3_OPT_C) nomC = d3Malloc(f->vnum * sizeof(int) );
  vlasli = d3GetFaceVertex(f);

  if( (7 <= f->vnum) ){//仮想頂点の抽出
        if(!vlasli){
              d3Free(nomor);
              return(FALSE);
        }
        vl = d3Malloc(f->vnum * sizeof(d3Vertex));  //作業用コピー
        if(!vl){
          fprintf(fp,"#メモリ不足のため、頂点照合省略¥n");
          vl = vlasli;
        }else{
          int mask;
          memcpy(vl,vlasli,f->vnum * sizeof(d3Vertex));
          mask = 15 - D3_VA_VIRTUAL;
          for(i=0;i<f->vnum;i++){ //仮想頂点以外の属性を除く
                if(15<vl[i].va) z3Message(3413);
                vl[i].va &= mask;
          }
        }
  }else{//頂点が 7 未満で仮想なき場合
          vl = vlasli;
  }
  //COORD,NORMAL,COLOR,TEXTURE の出力
  for(i=0;i<f->vnum;i++){
        if(vl != vlasli){//仮想橋の処理
              for(j=0;j<i;j++) if(!memcmp(vl+j,vl+i,sizeof(d3Vertex))) break;
              if(j<i) continue; /*dsb*/
        }
    if(bu){//頂点座標のソートを行う (#define bu 1)
              nomor[i]=buCOORD(fp,vl[i].v);
        }else{
              fprintf(fp,"¥tV%.2d = COORD(%0.15g, %0.15g, %0.15g);¥n",
                    i,vl[i].v[0],vl[i].v[1],vl[i].v[2]);
        }
    if(f->va_v & vl[i].va & D3_VA_NORMAL){/*処理略*/}
    if(f->va_v & vl[i].va & D3_VA_COLOR){/*処理略*/}
    if(vl[i].va & D3_VA_TEXTURE){/*処理略*/}
  }

  //VERTEX の出力
  for(i=0;i<f->vnum;i++){
        if(vl != vlasli){
              for(j=0;j<i;j++) if(!memcmp(vl+j,vl+i,sizeof(d3Vertex))) break;
              if(j<i) continue; /*出力済の位置にある重複頂点は省略する*/
        }
        fprintf(fp,"¥tP%.2d = VERTEX(V%.2d, ",nomor[i],nomor[i]);
    if(f->va_v & vl[i].va & D3_VA_NORMAL){
```

```
				if(OPTIM & I3_OPT_N){
					fprintf(fp,"N%.2d",nomN[i]);
				}else{
					fprintf(fp,"N%.2d",i);
				}
	}
	fprintf(fp,", ");
	if(vl[i].va & D3_VA_TEXTURE){
	   fprintf(fp,"T%.2d",i);
	}
	fprintf(fp,", ");
//	if(vl[i].va & D3_VA_COLOR){
	if(f->va_v & vl[i].va & D3_VA_COLOR){
				if(OPTIM & I3_OPT_C){
					fprintf(fp,"C%.2d",nomC[i]);
				}else{
					fprintf(fp,"C%.2d",i);
				}
	}
	//第 5 引数として
			if(vl != vlasli){//100129 同一頂点は既にスキップしているので、ここに来るとは限らない
				int k;
				for(k=0;k<f->vnum;k++)		if(vlasli[k].va		&		D3_VA_VIRTUAL)
if(!memcmp(vl+i,vl+k,sizeof(d3Vertex))) break;
				if(k<f->vnum){//同一位置の頂点で仮想あり
					if(k+1 < f->vnum) k++; else k=0;//仮想の先
					for(j=0;j<k;j++) if(!memcmp(vl+j,vl+k,sizeof(d3Vertex))) break;//仮想の先の前出
					fprintf(fp,", V%.2d",nomor[j]);
				}
			}
	fprintf(fp,");¥n");
  }
  //FACE の出力
  if(type == I3_EXT_FACE){
	fprintf(fp,"¥tF%.2d = FACE(",FaceNumber);
  }else{   /* LINE */
	fprintf(fp,"¥tL%.2d = LINE(",LineNumber);
  }
  for(i=0;i<f->vnum;i++){
	//長い行の折り返し処理
	if(i != 0){
	   if(i % 6 == 0){
	      fprintf(fp,"¥n¥t¥t");
	   }
	   fprintf(fp,", ");
	}
/*	fprintf(fp,"P%.2d",i); */
			if(vl != vlasli){//出力済の頂点をスキップする
				for(j=0;j<i;j++) if(!memcmp(vl+j,vl+i,sizeof(d3Vertex))) break;
				fprintf(fp,"P%.2d",nomor[j]);/*dsb.*/
			}else{
				fprintf(fp,"P%.2d",nomor[i]);
			}
  }
  fprintf(fp,");¥n");

//面の属性（法線、カラー）の出力
  if(f->va_f & D3_VA_NORMAL){
			if(type == I3_EXT_LINE){
				if(OPTIM & I3_OPT_N){
					nN = buNORM(fp,f->n);
					fprintf(fp,"¥tLINE_NORMAL(L%.2d, N%.2d);¥n",LineNumber,nN);
				}else{
					fprintf(fp,"¥tN%.2d					=
NORMAL(%g, %g, %g);¥n",f->vnum,f->n[0],f->n[1],f->n[2]);
					fprintf(fp,"¥tLINE_NORMAL(L%.2d, N%.2d);¥n",LineNumber,f->vnum);
				}
```

```
                }else{//FACE
#if 1
                        if( OPTIM & I3_OPT_N ){
                                d3Face F;
                                d3FindNormalf(f,F.n);
                                //if( *f->n == *F.n )
                                if(!memcmp( f->n, F.n, 3*sizeof(float) ) )
                                        goto NORMALEND;
                        }
//#else
                        if( OPTIM & I3_OPT_N ){
                                nN = buNORM(fp,f->n);
                                fprintf(fp,"¥tFACE_NORMAL(F%.2d, N%.2d);¥n",FaceNumber,nN);
                        }else{
                                fprintf(fp,"¥tN%.2d                                                                     =
NORMAL(%g, %g, %g);¥n",f->vnum,f->n[0],f->n[1],f->n[2]);
                                fprintf(fp,"¥tFACE_NORMAL(F%.2d, N%.2d);¥n",FaceNumber,f->vnum);
                        }
#endif
                }
  }
NORMALEND:
  /*040625*/
            if(f->va_f & D3_VA_COLOR){
              if( OPTIM & I3_OPT_C ){
#if 0
                        fprintf(fp, "#FACE_COLOR とりあえず省略¥n");
#else
                        nC = buCOLOR(fp,f->c);
                        if(type == I3_EXT_FACE){
                                fprintf(fp,"¥tFACE_COLOR(F%.2d, C%.2d);¥n",FaceNumber,nC);
                        }else{
                                fprintf(fp,"¥tLINE_COLOR(L%.2d, C%.2d);¥n",LineNumber,nC);
                        }
#endif
                }else{
                                fprintf(fp,"¥tC%.2d                                                                     =
COLOR(%g, %g, %g, %g);¥n",f->vnum,f->c[0],f->c[1],f->c[2],f->c[3]);
                                if(type == I3_EXT_FACE){
                                  fprintf(fp,"¥tFACE_COLOR(F%.2d, C%.2d);¥n",FaceNumber,f->vnum);
                                }else{
                                  fprintf(fp,"¥tLINE_COLOR(L%.2d, C%.2d);¥n",LineNumber,f->vnum);
                                }
                }
            }

    if(f->material){
                        if( OPTIM & I3_OPT_M ) CountMaterialOpt++;
                        else
                                fprintf(fp,"¥tM%.2d                                                                     =
MATERIAL(%s);¥n",f->material,d3GetMaterialName(f->material));
                        if(type == I3_EXT_FACE){/*040608 DR.H.K.*/
                                fprintf(fp,"¥tFACE_MATERIAL(F%.2d, M%.2d);¥n",FaceNumber,f->material);
                        }else{
                                fprintf(fp,"¥tLINE_MATERIAL(L%.2d, M%.2d);¥n",LineNumber,f->material);
                        }
    }
    if(f->texture){
                        if( OPTIM & I3_OPT_T ) CountTextureOpt++;//テクスチャ宣言は冒頭で一括済み
                        else{
                    fprintf(fp,"¥tI%.2d = TEXTURE(¥"%s¥");¥n",f->texture,d3GetTextureName(f->texture));
                        }
                        if(type == I3_EXT_FACE){
                                fprintf(fp,"¥tFACE_TEXTURE(F%.2d, I%.2d);¥n",FaceNumber,f->texture);
                        }else{
                                fprintf(fp,"¥tLINE_TEXTURE(L%.2d, I%.2d);¥n",LineNumber,f->texture);
                        }
    }
```

```
    if(type == I3_EXT_FACE){
      FaceNumber++;
    }else{
      LineNumber++;
    }
    if(vl != vlasli) d3Free(vl);
    d3Free(nomor);
    if(OPTIM & I3_OPT_N) d3Free(nomN);
    if(OPTIM & I3_OPT_C) d3Free(nomC);
    return(TRUE);
}
```

一方、本処理系において、データファイルとメタファイルから取得され SQL データ部宇に蓄積されたたデータから LSSG 形式に出力する処理を定義するメタファイルにおいては、面の出力部分は以下のように記述される。

リスト 3-7-7 LSSG 形式を出力するメタファイルにおける仮想線の扱い

```
（一つのオブジェクトに属する面を出力するループの部分）
        while(j=F()){
            for(nv=k=V();k;k=V()){
                printf("P%d=COORD(",Vcoord());
                printf("%f,",Sx());
                printf("%f,",Sy());
                printf("%f);¥n",Sz());
            }

            for(nv=k=V();k;k=V()){
                iN = Vnormal();
                iC = Vcolor();
                iT = Vtcoord();
                iV = Vvirtual();
                printf("V%d=VERTEX(",nv-k+1);
                printf("P%d",Vcoord());
                if(iN+iC+iT+iV == 0){
                    printf(");¥n");
                    continue;
                }else printf(",");

                if(iN){
                    printf("N%d",iN);
                }
                if(iT+iC+iV== 0){
                    printf(");¥n");
                    continue;
                }else printf(",");

                if(iT) printf("T%d",iT);
                if(iC+iV == 0){
                    printf(");¥n");
                    continue;
                }else{
                    printf(",");
                }
                if(iC) printf("C%d",iC);
                if(iV == 0){
                    printf(");¥n");
                    continue;
                }else{
                    printf(",P%d);¥n",iV);
                }
            }
        }
```

リスト 3-7-8 景観シミュレータから保存した LSSG 形式のファイル

```
# (ip ver.2.09) 国土交通省版・景観シミュレータ 2.09
    GRP2 = GROUP();
    CONCAVE(ON);
    V00 = COORD(1,1.4,0);
    C00 = COLOR(0, 0, 1, 1);
    V01 = COORD(0,0,0);
    C01 = COLOR(1, 0, 0, 1);
    V02 = COORD(2,0,0);
    C02 = COLOR(0, 1, 0, 1);
    V03 = COORD(1.106,0.679,0);
    C04 = COLOR(0.2045, 0.3105, 0.485, 1);
    V04 = COORD(1.21,0.33,0);
    C05 = COLOR(0.277143, 0.487143, 0.235714, 1);
    V05 = COORD(0.733,0.306,0);
    C06 = COLOR(0.524214, 0.257214, 0.218571, 1);
    P00 = VERTEX(V00, , , C00, V03);
    P01 = VERTEX(V01, , , C01);
    P02 = VERTEX(V02, , , C02);
    P03 = VERTEX(V03, , , C04, V00);
    P04 = VERTEX(V04, , , C05);
    P05 = VERTEX(V05, , , C06);
    F00 = FACE(P00, P01, P02, P00, P03, P04
        , P05, P03);
    N08 = NORMAL(0, 0, 1);
    FACE_NORMAL(F00, N08);
    GROUP_FACE(GRP2, F00);
```

リスト 3-7-9 LSSG 形式を定義したメタファイルによる出力結果

```
#入力ファイルは指定されていないことを確認
CONCAVE(ON);
#outvirtual.cmm(140531)
N1=NORMAL(0.000000,0.000000,1.000000);
C1=COLOR(0.000000,0.000000,1.000000,1.000000);
C2=COLOR(1.000000,0.000000,0.000000,1.000000);
C3=COLOR(0.000000,1.000000,0.000000,1.000000);
C4=COLOR(0.204500,0.310500,0.485000,1.000000);
C5=COLOR(0.277143,0.487143,0.235714,1.000000);
C6=COLOR(0.524214,0.257214,0.218571,1.000000);
P1=COORD(1.000000,1.400000,0.000000);
P2=COORD(0.000000,0.000000,0.000000);
P3=COORD(2.000000,0.000000,0.000000);
P4=COORD(1.106000,0.679000,0.000000);
P5=COORD(1.210000,0.330000,0.000000);
P6=COORD(0.733000,0.306000,0.000000);
G0=GROUP();
G1=GROUP();
V1=VERTEX(P1,,,C1);
V2=VERTEX(P2,,,C2);
V3=VERTEX(P3,,,C3);
V4=VERTEX(P1,,,C1,P4);
V5=VERTEX(P4,,,C4);
V6=VERTEX(P5,,,C5);
V7=VERTEX(P6,,,C6);
V8=VERTEX(P4,,,C4,P1);
F1=FACE(V1,V2,V3,V4,V5,V6,V7,V8);
FACE_NORMAL(F1,N1);
GROUP_FACE(G1,F1);
L1=LINK(G0,G1);
LINK_XFORM(L1,LOAD,MATRIX,1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,0.
000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.000000,1.000000);
CONCAVE(OFF);
```

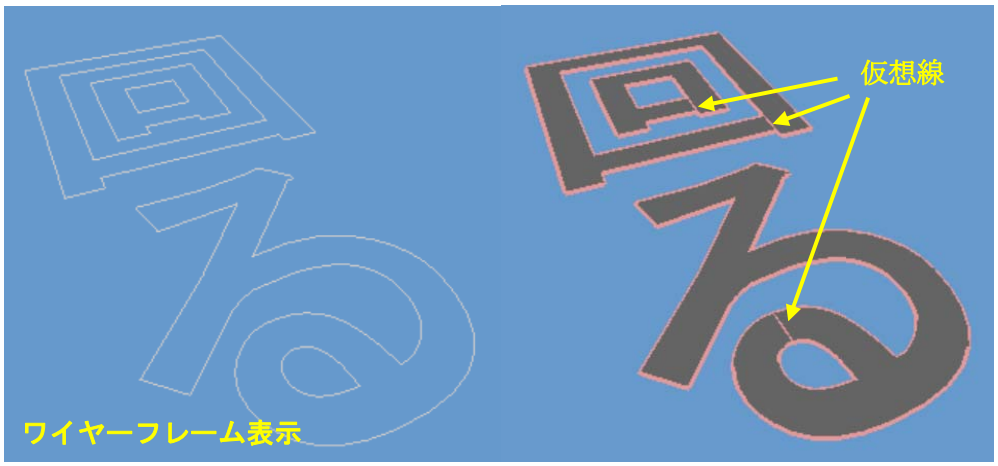


図 3-7-2：フォントファイルから生成した穴あき図形

（５） LINK の出力処理のためのライブラリ関数

① 関数の仕様と用法 cci_sqo, cci_dml における処理

シーングラフ型のデータにおいては、一つのオブジェクト（シンボル等とも呼ばれる）を、位置やスケールを変えて複数回使用することができる。例えば、４本の足と天板から成るテーブルを３０脚配置された教室を例に取ると、実際に形状が定義されているオブジェクトは天板と足の２個である。リンクの数は、教室に３０脚を配置する３０と、一つのテーブルを定義する５、合計３５である。表示されるオブジェクトは、天板３０と、足１２０の合計１５０である。リンクは、３５のリンクに順次アクセスし、カーソル・リンクのそれぞれに定義された位置情報等を取り出すことが目的である。

（メタファイルの書法については４－２で解説した）。

L_()以下の実装は、以下のような呼び出しを想定して行った。

まず、L_()関数を用いて、全てのリンクにアクセスする。

int Lp()関数で親グループの Gid を得る

int Lc()関数で子グループの Gid を得る

② 移動・スケール・回転の取得

カーソル・リンクに関して、位置情報を取得するために、以下の関数を用意した。

float Lm()関数を 16 回呼び出して、マトリクスの成分にアクセスする

int Li() 単位マトリクスなら１、それ以外ならゼロ

quat Lt() 平行移動成分を返す

quat Lr() 回転成分を返す（引数に PRE または POST を指定すると、前または後の回転を、また引数がない場合には二つの回転の合成を返す）

quat Ls() スケール成分を返す。

景観シミュレータ(1996)において未解決の問題であったが、仮想コンバータにおける四元数とテンソルの数学的な検討(4-4)に基づいて、任意の回転マトリクスを二つの回転と一つ

のスケールに分解し M(Qs,Qr,Qt)と表現する計算処理を新たに実装した。

（６）ディスプレイ・リストの取得

・ルートグループ（上方リンクがない）も表示の対象とする

・下方リンクを N 個有するグループが M 個の上方リンクを有する場合、
リンクリストでは高々M+N 個のリンクを処理すれば良いが、
ディスプレイ・リストでは少なくとも M×N 個の処理を行う必要がある。

int D_() ディスプレイ・リストを作成し、表示グループの ID を順次返す
表示グループにカーソルを当てる
終了するとゼロを返す

int Dlevel() 現在の選択グループの階層の深さを表す
ルートグループなら 0
カーソルグループがなければ-1

int Dfirst() 表示グループに対応する実体グループに最初にカーソルが当たった時 1
実体グループの内容を出力する必要がある場合に使用する

D_()ループが実行されている間は、頂点座標と法線ベクトルの意味が異なる
Sx(),Sy(),Sz(),Nx(),Ny(),Nz()は、グローバル座標における値とする
DL 配列が存在している時、Vcoord 関数が実行された時点でグローバル座標をスタティック変数 Gx 等に作成
Lm()については未定義

３－８．仮想コンバータのセキュリティ

開発段階において、仮想コンバータが正常に動作するところまで実現すれば、データファイルの長期保存という当初の目的は達成されたことになる。しかしながら、本章を執筆する 2015 年時点では、情報システムのセキュリティが社会問題となっている。

本研究を実施していた 2010~15 年当時、PC をネットワークに常時接続する使用環境が普及し、悪意あるソフトウェアからシステムやデータを保護するためのセキュリティ対策の重要性が認識され、対策が工夫されていた。

（１）セキュリティ対応のための C 関数の仕様変更への対応

C 言語で頻繁に使用する固定長配列の限界を超えたアクセスは、例えば想定外の長いファイル名を有するファイルや、ファイル名の一部にプログラムを含むようなファイルによる攻撃の侵入経路となりうることから、従来広く使われていた C 言語の入出力関数や文字列処理関数の脆弱性が指摘され、VS2005 からセキュリティを高めた仕様の関数を使用することが勧奨され、従来の標準 C の関数の使用に対して警告が表示された。一例を挙げると、

```
char buffer[30];
fscanf(fp,"%s",buffer);
```

というコードは、データファイル中に長い行が存在すると簡単にバッファ・オーバーフロ

ーを起こす。これに対処して、

fscanf_s(fp,"%s",buffer,30);

という代替関数を用い、固定長配列の限界を超えた入力をブロックする方法である。この場合、エラー終了となる。

しかしながら、このような代替関数への変更を行ったソースコードは、異なる開発環境（例えば、後述の Android アプリのための C コンパイラ NDK）ではリンクエラーを起こす。移植のためには、再び脆弱な fscanf 関数に戻さなければならない。

このため、メタファイルとデータファイルを扱う仮想コンバータの基幹部分においては、fscanf_s という代替関数を使用せず、例えば上記のような場合には、

fscanf(fp,"%30s",buffer);

のようにしてバッファ・オーバーフローを予防しつつ可搬性を確保する。

一方、メタファイル作成のために用意した scanf 関数においては、固定長のバッファに可変長のデータを受け取るような処理は記述できないようになっている。

（2）メタファイルが呼び出すライブラリ関数のセキュリティ

例えば、保存されたデータファイルとメタファイルを利活用する時点で、これを WEB や媒体を通じて配布するような状況が考えられる。その時に、悪意のある偽のメタファイルやデータファイルが配布された場合に、これがユーザの形態端末や WEB サーバから個人情報等を引き出して他者に送信するような動作が行えないようにあらかじめ機能を制約しておく、ということも重要な配慮事項となる。

本処理系においては、以下のような配慮を行っている。

1）コンパイルされた実行形式を格納する領域は、処理系が固定長配列としてメモリ上に確保した領域のみとし、長いメタファイルを処理した結果がこれを超える場合にはエラーとした。

2）メタファイルのコマンド（ライブラリ関数）でアクセス可能なメモリ空間は、インタープリタ処理系が固定長配列として用意したメモリ空間のみとし、OS や、その他の別の処理系が使用中のメモリにはアクセスできないようにした。メタファイルの中で定義する変数や配列は全て、この固定長配列の一部として割り当てる。

3）メタファイルのコマンド（ライブラリ関数）では、メモリブロックを取得するための C 言語における malloc-free 系のコマンドは提供しない。メタファイルで宣言する配列は全て固定長として宣言する。メタファイルが添付するデータファイルが特定であることから、自由に伸長可能な配列をメタファイルの記述の中で用意する必要はなく、十分な長さの固定長配列を使用することにより対応できる。

4）メタファイルが入力のためにアクセスするファイルは、データファイルのみとし、メタファイルをコンパイルして生成した実行形式をインタープリタが処理開始する時点では既に固定的なアクセス先としてオープンされているようにした。メタファイルのコマンド（ライブラリ関数）が新たな任意名称のファイルをオープンしてアクセスすることはでき

ない。
５）メタファイルが出力先とするファイルは、利活用形態別にあらかじめ定められた出力ファイルとログファイルのみとし、メタファイルのコマンド（ライブラリ関数）で新たな任意名称のファイルをオープンして書き込み動作を行うことはできない。
６）サーバ上で動作する VC-4D においては、悪意のある攻撃的なメタファイルを添付したデータファイルが投稿されることが考えられる。例えば、意味のない無限に多くの頂点や面を有する図形データを、エンドレスループで定義するような処理が定義されている場合、データベース上に巨大なテーブルを生成することも可能である。また同様に、同じ文字列を反復するだけの無限に長いデータファイルやログファイルを出力するような処理も記述可能である。また、単に終わりのないエンドレスループを定義するだけで、処理系は無限の時間を浪費する可能性がある。このような場合、データファイルへのアクセスポイントが一定時間経過しても変化しない場合に処理を打ち切るような処理を行っているが、まだ完璧とは言えず、今後工夫する余地がある。

（３）ライブラリ関数の、利活用処理系における実装

個別の処理系に関しては以下の通りである。

１）入出力

VC-1C において、purintf, logf の出力が行われた場合：ログファイルに追記

S_()などのアクセス系コマンドが実行された場合：ダミー関数で何もしない

VC-2V において、

ファイル出力時にデータ構築系のコマンド GROUP()等が実行された場合

→①メモリ空間にデータが構築されてしまう

②出力ファイルはオープンされるので、空のファイルが生成する

ファイル入力時にファイル出力系のコマンド(printf 等）が実行された場合

→ログファイルに出力が行われ、入力データファイルに影響はない

VC-4D において、

ファイル入力時にファイル出力系のコマンド(printf 等）が実行された場合→ログファイルに出力を行う。

ファイル出力時にデータ構築系のコマンド GROUP()等が実行された場合

→現段階（2015 年 10 月）では、実行されてしまう。

今後の対策として、コンパイラのエラーチェックを強化し、LSSG 系コマンドを、ファイル出力時にはコンパイル・エラーとする方法が考えられる。

２）入出力の選択をコンパイラに伝えるための方法

２－１）VC-2V.dll の場合、

2Vwnd.cpp の中の、OnBnClickedOk()関数において、

入出力選択ボタン（ボタンの外見を有するチェックボックス）の状態を b_IO フラグで見る。

b_IO が０なら、入力処理を行い、データファイルから読み出して DML に出力する。
b_IO が１なら、出力処理を行い、DML から読み出してデータファイルに出力する。

２－２）VC-4D.exe の場合、コマンドラインの第三引数で,VC-4D_exe.cpp の main 関数が、第 4 引数を IO として受け取る。upload 関数か download 関数かを選択する。
IO が０なら、ダウンロード処理を行い、SQL から読み出してデータファイルに書き込む。
IO が１なら、アップロード処理を行い、データファイルから読み出して SQL に書き込む。

３）compile 関数に対して、入出力を伝えるための方法

データベースを改変する処理、具体的には記憶系のライブラリ関数が、利活用のためのメタファイルから呼び出されると、不適切な結果となる。このようなチェックは、メタファイルの実行段階でブロックすることができるが、コンパイル段階で事前にエラーとして処理し、実行段階に移行しない方が確実である。

cci_pars.c に、ReadOnly グローバル変数を用意し、値が非ゼロにおいて記憶系のライブラリ関数が使用されるとコンパイル・エラーとした（statement 関数および factor 関数の内部での処理)。

４）サービスによる処理の区分

VC-4D のようにアップロードとダウンロードのサービスを提供する処理系においては、ユーザが保存データとメタファイルをアップロードする場合と、新たな形式を記述したメタファイルを添付してダウンロードする場合で、以下のように処理を区分している。

表 3-8-1 セキュリティ対策

処理内容	アップロード時	ダウンロード時
printf 関数	ログファイルに出力	データファイルに出力
logf 関数	ログファイルに出力	ログファイルに出力
scanf 関数	データファイルから入力	エラー
COORD 関数等	SQL データベースに入力	コンパイル・エラー
Sq()関数等	無意味な結果	SQL データベースから取得

これにより、不本意に保存データが上書きされ損傷されるリスクは回避されていると考えられるが、メタファイルの実行時に長時間応答がなくなる場合や、無意味なファイルが作成され返送される可能性が残されている。

（４）書式文字列の検査

printf、logf、scanf 関数においては、書式文字列と引数が不整合のまま実行されると、スタックを介して不正なコードが実行される可能性がある。

本処理系においては、プログラム領域とメモリ領域とスタック領域を分離し、プログラム領域以外をプログラム・カウンタが参照できないようにしている。

引数の数は０か１としているため、書式文字列の中で「%数値型」により参照する引数

は高々一つである。複数の引数が参照された場合には、コンパイル・エラーとして処理した。

標準 C 言語においては、printf(“%*d”,width,value); の形で引数により幅を指定するが、本処理系ではこの形を許容していない。

標準 C 言語において、printf(“・・・%n・・・”, &location); の形で printf の引数（アドレス）に値（出力済文字数）を代入する脆弱な方法（想定外の書式文字列により読み出し用に用意したデータが書き換えられる）は、printf(“・・・%n・・・”, location);の表記法で利用可能とした。

書式文字列としては、標準 C 言語においてはポインタ渡しで任意のものを使用可能であるが、本処理系においてはメタファイルの中でリテラル文字列として固定的に定義した文字列のみを使用している。

（５）サーバのセキュリティ

サーバのセキュリティが社会的な問題になっている。様々な攻撃パターンに対する一般的な対策の勧奨と、実施状況の報告などが求められている。これらの報告が簡単なメール添付で行われていることが多く、サーバの構成やセキュリティ対策を記載した報告様式のデータにパスワードを付すような対策を行っている。

一方、悪意のある攻撃の中には、無意味なアクセスを集中させるようなパターンがある。これは、感染マシンからのプログラムによる攻撃である場合が多いため、画面に表示した簡単なパスワードでガードするだけでもある程度は防ぐことができる。重要な点は、攻撃を検知した時に、サーバ側に生じている現象を分析し、個別に対策を考える努力を行うことではないかと考えている。

３－９．まとめ

以上解説したように、新しい処理系を作成に当たり、以下のような点を考慮しながら作業を進めることは効果があったと考える。

（１）段階的な前進

①既存のデバッグの進んだライブラリ（ソースコード）の活用

a コンパイラ処理系を、シンプルな教材用処理系から拡充して開発

b 景観シミュレータの外部関数、プラグイン DLL として実装することにより、基幹部分動作確認を既存の処理系を用いて実施

c 景観シミュレータの既存ライブラリ（DML、g3DRL）のデバッグが進んだソースコードを利活用処理系に最大限活用した

②異なるプラットフォーム上での開発における技術要素の分解

新しい処理系を実現する上で超えなければならないハードルを、異なる要素に分解し、仮設的な処理系を試作しながら一つずつ段階的に解決すること

a VC-3M のためのエミュレータ

Windows システム上でのエミュレータを開発して動作検証した上で、JAVA 言語による

Android 上の API をアウトソースにより開発した。

b VC-4D のためのエミュレータ

基幹部分に加えて、WEB サーバや SQL データベース等の複数技術を併用しているため、SQL 上での検証の上、WEB サーバへの実装を分けてアウトソースにより開発した。

③役割の終わった仮設的処理系の活用

通過点として作成した仮設的プログラムも、データ検証ツール等としての用途を見出すことができる。

a VC-1C、VC-2V は、景観シミュレーション・システムの機能拡張として活用

b VC-4D は、SQL サーバの検証ツールとして活用

（２）イノベーション

①VC-1C

a プラットフォームに依存しないコンバータ

従来の、景観シミュレータにおける各種ファイル形式のデータを入力するための外部関数は、C 言語のソースコードとして記述され、Windows 上の開発環境によりビルドされて Windows 上の実行形式として提供されていた。この実行形式は、UNIX 等の別の処理系の上では実行することができず、また修正を行うためには、開発環境に戻らなければならなかった。

b コンバータのソースコードとしてのメタファイル

VC-1C は、実行時に、まずメタファイルをコンパイルし、生成した実行形式をインタープリタが実行することにより、入力するデータファイルを変換する。したがって、メタファイルは、様々な処理系に共通で使用することができ、テキストエディタが使用できる環境においては、修正したり増補したりできる。

②VC-2V

a 中間ファイルなどを経由しないライブラリ関数の実装形態

実行時にライブラリ関数からメモリ上の構造体を構築する DML ライブラリ関数を直接実行し、直ちに表示を行う。外部ファイルを経由しない

b 出力系のライブラリ関数の実現

DML ライブラリ関数を通じ、メモリ空間から形状構成要素を取り出して、異なるファイル形式でのファイル出力処理を行うことができる。

③VC-3M

a Android 上での基幹部分の可搬性の検証

b 背面カメラ、GPS、磁気、加速度センサを用いたリアルタイム写真合成

c 簡単な操作による三次元的なキャリブレーション（「足によるキャリブレーション」）

d シャッター操作による現場活動の中での位置情報と画像の記録保存

④VC-4D

a サーバ OS 上での基幹部分の可搬性の検証

b SQL データベースへのデータの完全な展開（電源を切っても消えない）
c SQL データベースからのデータの取り出しと、任意形式のデータの再構成・出力

数年間の継続期間をもって運用されるサービスとして見た場合、処理中のデータをデータベースが管理するファイルとして保管しているこのシステムは停電等に対して有効である。また、バックアップ等の様々なデータベース関連の機能を活用することができる。

しかしながら、百年単位での長期保存という観点から見ると、WEB サーバの OS のバージョンアップへの対応を継続的に求められる点や、物理的にデータを保管しているハードディスク装置（RAID により、クラッシュに対しては対応できるが、その場合にも交換する体制を維持する必要がある）のメンテナンスが必要であり、制度的な裏付けがなければ、長期保存の目的を実現することはできない。

このような意味で、VC-4D の位置づけは、「長期保存の手段」ではなく、「利活用の手段」の一形態であり、保存されたデータをユーザーがメタファイルで指定した任意形式のファイルに変換するサービスとして位置付けることができる。

４．資料編

4. 資料編

４－１．メタファイルの文法

メタファイルの書法は、C 言語に準拠した文法規則と、景観シミュレータの LSSG 形式のコマンドを母体としたライブラリ関数により構成されている。

（1）コメント

①/*・・・*/

1)「/*」以後は、「*/」までコメントとして、コンパイラの処理において読み飛ばす。

2)「/*」に対応する「*/」がない場合（メタファイルの終端に達した場合）にはエラーとする。

3)二つの引用符「“」で囲まれたリテラル文字列の中に現れるパターン「/*」「*/」は、コメントの開始、終了として扱わない。

[例]printf(“コメントは、/*から始まり、*/で終わる¥n”);
出力：コメントは、/*から始まり、*/で終わる

4) 「//」の後の行端までの「/*」はコメントの開始としては扱わない。

5) 「/* // */」はコメントとして完結する。

6)「/**/」はコメントとして完結するが、「/*/」は完結しない（コメントは継続する)。

②//

1)「//」以後、改行またはファイル終端までコメントとして扱う。

2)リテラル文字列の中に現れるパターン「//」は通常の文字列として扱われる。

[例]printf(“行の途中の//から後ろは、コメントとして扱われる”);
出力：行の途中の//から後ろは、コメントとして扱われる

3) /* ・・・ */ の中に現れる「//」は、パターン「*/」をコメントアウトしない。

③#

1)「＃」以後、改行または終端まで、コメントとして扱われる(②「//」と同等の効果)

これにより、標準の C コンパイラで用いられている、＃で始まる制御文は、全てコメントとして処理する。

#include, #if 0, #else, #endif, #define 等

2)リテラル文字列の中の「＃」は、文字列の一部として扱う。

④COMMENT 文

1)メタファイルにおける「COMMENT(ON);」の行から後の行は、コメントとして扱われる。コメントは、「COMMENT(OFF);」の行まで続く。

2)「COMMENT(OFF);」のコマンドが上記①～③の方法でコメントアウトされている場合には、このコメント終了宣言は無効である（コメント行が継続する)。

3)「COMMENT(ON);」の状態で、「COMMENT(OFF);」がないままファイル終端に達した場合には、「COMMENT(ON);」以降は全てコメントとして処理する（エラーではない)。

4) 「COMMENT(ON);」のコマンドが先行しない領域で「COMMENT(OFF);」のコマンドが実行された場合には、無視する（エラーとしない)。

（２）数値型

変数、配列、関数は数値型を持ち、使用に先だって数値型を宣言する。

①void 型

値を返さない関数、引数を持たない関数の宣言に用いられる。

②int 型（整数型）

32 ビットの整数値(-0x80000000=-2,147,483,648～0x7fffffff=2,147,483,647)であって、４バイトのメモリに格納される。

③float 型（浮動小数型）

ISO/IEC/IEEE 60559-2011 に従う 32 ビットの単精度浮動小数であり、1 ビットの符号、8 ビットの指数、23 ビットの仮数で構成され、４バイトのメモリに格納される。

符号ビット s は、0 ならば正数、1 ならば負数を表す。

+0, -0, +∞, -∞を表現することができる。

指数 e は、1-254 の値を有し、127 でバイアスされている。

例えば、130 ならば、仮数部を 8(=$2^{(130-127)}$)倍する。

数値が無限大の場合 255 となり、仮数部は 0 となる

数値が NaN（非数、演算失敗のような場合に発生する）のとき 255 となる

（その場合、仮数部には非ゼロの値である）

数値がゼロならば、指数部仮数部ともに 0 となる。+0 ならば 32 ビット全て 0 である。

仮数部は、0~0x7fffff=8,388,607 の値をとる。二進数で表現した小数において、1.0 以上 10.0 未満を表現するため、最上位の 1 ビットを除いた小数点以下を 23 ビットで表現する。

解像度の点でみると、約 8km 程度のエリアの内部の位置座標を 1mm の精度で表現するため、一つの団地やブロックの形状を表現するためには十分であるが、地球座標系による位置を実用的な精度で表現することは困難である。

④quat 型（四元数型）

倍精度浮動小数 64 ビットで t,x,y,z の４値を束ねた、合計 256 ビットの数値型であり、座標値を 1 変数で扱うことができ、ベクトルの内積、外積、回転などを計算するために便利であるために用意した。倍精度浮動小数は、1 ビットの符号部、11 ビットの指数部、52 ビットの仮数部を有し、仮数部は 0~4,503,599,627,370,496 の数値をとることができ、1mm の精度で約 45 億 km までの長さを表現することができることから、地球座標系で建物の位置等を十分記述するにことができる。quat 型を用いた座標計算については、資料４－３で解説した。

（３）変数

[書式]

```
数値型 変数名;
```

[例]

```
quat q1;
int array[30];
float func1(float x, float y, float z);
```

①変数名

アルファベットまたは「_」から始まり、アルファベット、数値または「．」「＿」により構成される 30 文字までの文字列である。大文字と小文字は区別する。

1) プログラム言語のトークンと同じ名称の変数､配列､関数の型宣言を行ってはならない。
｛void,int,float,quat,if,for,while,do,switch,case,break,continue｝

2) 組込関数やライブラリ関数と同じ名称の変数､配列､関数の型宣言を行ってはならない。
exit, printf,logf,scanf,sin,cos,…（４－２参照)。エラー「記述が不適切（予約語を用いた宣言等)」として処理する。

②局所変数

1) 関数の中でプログラムが記述される前に宣言する（(5) 参照)。

2) 局所変数は関数の内部だけで有効であって、関数内部で代入が行われる前の初期値は動的に割り当てられるメモリ領域の値がそのまま反映され不定であり、関数が終了した後は参照することはできない。

3) 変数の宣言と同一行の中で値を初期化することはできず、プログラムの中で参照が行われる前に値を与える必要がある。

4) 数値型が省略された場合には、整数型として宣言が行われたものと見なす。
[例] `v;` は、`int v;` と同等

局所変数のメモリ領域は、memory 配列の末尾から、関数呼び出される際に確保され、関数から戻る際に解放される。具体的には、関数呼び出し元のプログラムの CALL 仮想命令の中で、現在の PC の値がスタックのトップに格納されて関数アドレスにジャンプする。

関数の最初に呼び出される ADBR 仮想命令で baseReg を必要量だけ減じることで引数と局所変数のためのメモリ領域が確保され、その先頭には、スタックのトップが、続く引数領域にはスタックから引数の値が格納される。

関数の終了時に、メタファイルの return 命令で指定された戻り値をスタックに積み、メモリ領域の先頭から戻りアドレスを取り出してこれも、スタックに積んだ上で、再度 ADBR 仮想命令で baseReg を元に戻し、RET 仮想命令の中で、スタックのトップが PC に代入される。関数の中での数値演算処理の中でスタックの位置ずれが生じても、RET による戻り先は保護される。一方、局所変数の中に定義された配列の限界を超えたアクセスが行われると、メモリ上に保存された戻り先が書き換わる危険性がある。このため、配列のアクセスに際しては、境界チェックを行う機械語(BCH)を実行するように改良した。

関数の戻り値はスタックに置かれるため、式の演算などに引き続き利用される。

③大域変数

1) 大域変数は、全ての関数の内部から参照し代入することができる。数値型にかかわらず初期値は全てのビットがゼロである。

```
大域的な整数の初期値：0
大域的な浮動小数の初期値：0.0
大域的な四元数の初期値：(0.0, 0.0, 0.0, 0.0)
```

2) メタファイルのプログラムとしての実行の間、代入が行われてから次の代入が行われる

まで同じ値を維持する。

3) 関数から代入・参照される前に、関数の外で、関数内部のプログラムから最初に参照または代入が行われる以前に宣言する。その位置は必ずしもメタファイルの冒頭である必要はない。

④１行中での複数の変数の宣言

同じ数値型の変数を１行で、コンマ区切りでまとめて宣言することができる。

但し関数のプロトタイプ宣言（後述）は、１行で一つしか宣言することができない。

```
int i1,i2,i3;          ←可
int i1(), i2(), i3(); ←不可
```

⑤宣言されていない変数への代入

その位置でその変数が int 型の局所変数として宣言されたものとみなす。但し、宣言が行われていない配列への代入が行われた場合にはエラー「[の前に ； が必要」とする。

⑥宣言されていない変数の参照

エラー「未定義の識別子(変数名)」とする。

⑦重複定義

1) 関数の定義の中で宣言される局所変数、配列の名称は、引数の名称と一致してはならない。

2) 大域的変数の名称と同一名称の関数があってはならない。

3) 同一名称の大域的な変数、配列は、数値型の異同にかかわらず複数回定義することはできない。

4) 関数定義の内部で宣言される局所変数、配列の名称は、関数名、大域的変数名、大域的配列名と同一であっても構わない。

5) 大域的変数と同一名称の局所変数が関数定義の内部で定義された場合、関数内部では局所変数を使用する。その場合、関数内部から同名の大域的変数への参照、代入は行うことができない。

6) 大域的変数と同一名称の局所変数は異なる数値型であっても良い。

⑧その他

1) 局所変数、配列は、メモリ上に動的に割り当てられ、その関数が実行されている間だけ有効である。局所変数、配列が初期化されずに参照された場合には、その値は保証されない。割り当てられたメモリ上のアドレスに、それまでの処理の結果として偶然残っていた値が参照される。

2) 宣言されただけで一度も使用されない変数に関する警告はない。

3) 関数宣言内に｛　｝でサブブロックを設け、その中で更に局所の変数を宣言して処理を行うことはできない。if, switch, for, do, while 文で用いるサブブロックも同様である。

（４）配列

次の書式で宣言する。

数値型 配列名[配列の長さ;

[例]

```
int array[30];
```

①配列名

大域的配列と同名の局所的配列が関数の内部で定義された場合、関数内部での配列の処理は局所的配列に対して行われ、大域的配列には影響しない。

②配列の長さ

1) 配列の長さは、１以上の整数値でなければならない。

2) 配列の長さを「１００＋１」等の定数式で表現することができる。

3) 配列の長さが空の場合、エラー「添字指定がない」。

4) 長さが０または負値の場合、エラー「添字が不正」。

5) 長さの指定に「0x3000」等の表記：コンパイルエラー（添字が不正）

6) 配列の長さの上限は、処理系に依存する。

7) 大局配列を確保するためのメモリが不足する場合には、コンパイルエラー「メモリ不足」となる。

8) 局所配列のためのメモリが不足する場合にはランタイムエラー「フレーム確保失敗」とする。

③配列へのアクセス

1) 配列の添字は変数等を含む整数式により定義することができる。

2) 配列を参照し、またこれに代入を行う際には、式の値である添字は、０以上、かつ配列の長さ未満でなければならない。

3) 添字の範囲の検査は実行時に行われ、ゼロ未満または配列の長さ以上の場合には直ちにメッセージを発してエラー終了となる。

（５）関数の定義

メタファイルの中で定義する関数は、以下の書式に従う。

```
数値型 関数名(引数リスト){
   局所変数の宣言
   プログラム（関数の処理内容）
}
```

メタファイルのコーディングにおいては、コンパイラに予め組み込まれた組込関数と、メタファイルの中で自由に定義する関数を使用することができる。ここでは、メタファイルの中での関数の定義方法について解説する。組込関数については(12)で解説する。

①数値型

1) 関数の型は、void、int、float、quat のいずれかである。

2) main 関数の宣言は、int 型でなければならない。

②関数名

1) 宣言済の大域的変数、大域的配列と同一名の関数が定義された場合、エラー「識別子が重複している」とする。

2) 同一名の関数が複数回定義された場合、エラー「関数が再定義されている」とする。

③引数リスト

1) 引数リストは、数値型宣言空白を空け、変数名を置く。

2) 複数の引数を使用する場合、コンマ「,」で区切って続ける。

3) 関数名と同じ名称の変数を、引数として宣言し使用することができる。

4) 数値型のみを宣言して変数名を省略された場合、エラー「記述が不適切」。

5) 数値型を省略して変数名のみが宣言された場合、エラー「型指定誤り」。

6) 引数がない関数の引数リストは、空欄とすることも、void とすることもできる。

7) void 型引数の宣言の後に変数名があってはならない。

8) void 型が、複数の引数の一つとして記述されてはならない。

```
void func(void, void, void);
```
エラー「,の前に）が必要」

```
void func(void V);
```
エラー「V の前に）が必要」

9) 配列全体を引数として受け取ることはできない。

10) 引数リストの内部で同じ名称の引数がある場合、エラー「識別子が重複している」とする。

[例]

```
int func(int var1, float var1){
    ・・・
};
```

④局所変数、局所配列の宣言

1) ある関数の内部で使用される局所変数、局所配列として、大域的変数、大域的配列と同一名称の変数、配列が宣言された場合には、その関数内における参照は、局所変数とする。この場合、同名大域変数には関数内からアクセスすることはできない。

2) 定義した関数と同一名称の局所変数、局所配列を宣言して使用することができる。

3) 引数として定義された変数と同じ名称の変数が関数内部で局所変数として宣言された場合、エラー「識別子が重複している」とする。

[例]

```
func(int var1){
    int var1;
}
```

[補足説明]コンパイラの内部処理において、局所変数の登録は、block 関数(cci_pars.c)で行っている。
set_type() set_name()を行い、tmp_tbl に変数を登録する。この時、ライブラリ関数名との重複が刎ねられる。
次に、enter 関数で記号表登録を行う。
この中で、nameChk（引数と変数の名前重複チェック）を行う
引数と局所変数を新規登録しようとする場合だけをチェックの対象とする

```
同一名称のエントリ p が存在した場合、p->level とレベルの比較を行う
  p->level >= nest ならば、識別子が重複しているエラー
nest は、ブロックの深さを表し、０が大域、１が関数、２が関数内
ここで、現在登録しようとしているものが引数である場合には、関数内の局所変数等と同一レベルに調整している。
```

⑤プログラム

1) 処理の記述で参照する関数は、それ以前にプロトタイプ宣言されているか、定義されている関数でなければならない。エラー「未定義の関数を使用(func)」とする。

2) プログラムの記述が開始してから、局所変数の宣言が行われるとエラー

3) プログラムの中で、関数の型宣言を行ってはならない。エラー「不正な記述(int)」等。

4) プロトタイプ宣言されただけで、関数定義がない関数が参照されると、最終行でのエラー「未定義の関数(func)」とする。

5) 定義されただけで呼び出されない関数は、機械語を生成するだけで実行しない。

6) プロトタイプ宣言されただけで、関数定義も参照もない関数は無視される。

⑥ 前方参照

1) 関数の定義（{} 内に記述する処理）の中で参照する大域的変数や配列は、関数の定義より前に宣言されていなければならない。

2) 関数の定義の中で参照する関数は、関数の前に定義されているか、またはプロトタイプ宣言されていなければならない。プロトタイプ宣言も関数定義もされていない関数は、参照された時点でエラー「未定義の関数を使用」

⑦return 文

[書法]

return 戻り値または式;

[例]

```
return;  //void 型の関数の場合
return 1; //int 型の場合
return x*3.14; //float 型の場合
```

1) return 文は、関数内のプログラムの任意における関数からの終了を示す。

2) return 文は、引数（関数からの戻り値）の式を括弧の外に置くこともできる。

「return(式);」と「return 式;」は同等である。

3)数値型を指定した、戻り値のある関数は、同じ数値型の値を返さなければならない。エラー「return 文に戻り値がない」「関数の型と戻り値の型が異なる」

4)void 型の関数の場合には、末尾の return 文を省略することができ、また引数のない retrun 文で終了することもできる。

5)return 文の後にプログラムが記述された場合、コンパイラは機械語を生成するが実行されることはない。

（６）関数のプロトタイプ宣言

[書式]

```
数値型 関数名(引数リスト);
```

[例]

```
float func1(float x, float y, float z);
```

1) 関数の定義よりも前の行で関数の呼び出しを行うと、エラー「未定義の識別子」。このような場合に、関数の定義、および呼び出しの前にプロトタイプ宣言が行われていれば、正しくコンパイルされる。

2) 同一名称の関数についてプロトタイプ宣言が複数回行われた場合であって、異なる型で場合には、エラーとして処理する。同一型である場合には、無視される。

3) プロトタイプ宣言とは異なる型で関数の定義が行われた場合には、エラー「関数プロトタイプが不一致」。

4) 引数リストの変数名は、関数の定義における引数リストと異なっていても、また複数回プロトタイプ宣言が行われた場合に相互に異なっていても、数値型と引数の個数が同じであれば構わない。但し、引数の変数名は省略することができない。

5) 複数の関数のプロトタイプ宣言を１行でまとめて行うことはできない。

```
int a(),b(),c();
```
→エラー「(の前に；が必要)」

6) 関数のプロトタイプ宣言を、変数宣言と同一行内で行うことはできない。

7) 関数のプロトタイプ宣言だけが行われ、関数の定義の本体が存在しない場合、メタファイルの末尾までコンパイルを終了した後に、その関数が呼び出された行でエラー「未定義の関数」。但し、他から一度も参照されない関数は無視される。また、関数が定義されている場合で、一度も参照されない場合も無視される（警告等は行わない)。

8) 組込関数は宣言や定義なしに直ちに呼び出すことができる。組込関数のプロトタイプ宣言が行われた場合、たとえ同じ数値型、引数型を正しく指定しても、エラー「予約語を用いた宣言等」とする。

（７）文

文は、statement 関数で評価する。セミコロン「;」で文の終端を示す。文は、COMMENT 文、宣言文（上記)、プログラム制御文(9)を除き、一般的に以下の形式の記述である。

①関数呼び出し

```
関数(引数リスト);
```

1) void 型の組込関数は、単独で文となる。

[例]

2) ユーザーがメタファイルの中で定義した戻り値のある関数は、単独で呼び出すことができる（スタックに残された戻り値は棄却される)。

3) 組込関数で、値を返す関数であって単独で呼び出される意味がないものについては、エラーとしている。

[例]sin(0.0); はエラー「不正な記述(sin)」

4) 値を返す組込関数の内、以下のものについては、戻り値を利用せずに単独で呼び出すことができる（スタックに残された戻り値は棄却される)。

[例]　input、scanf、SEEK

②変数への代入

変数＝式;

1) 式の評価値が、左辺の変数に代入される。

　式は、定数、変数、配列、戻り値のある関数を、二項演算子（+,-,*,/等）や括弧で結合した記述である（９）。

2) 代入文自体が式として代入に等しい値を有するため、

変数=変数＝・・・＝式; の形の表現ができる。

[例] i=i=i=i=i+1;//i に i+1 が代入される

[例] i=i+1=1;　//エラー（i+1 は左辺値ではない）

[例] i=i+(i=1); //i に 1 が代入される

[例] B =–(A=(B+=A)-A)+B; //A と B の値が交換される(swap)

[例] B－=(A=(B+=A)-A); //A に元の B の値が代入され B にはゼロが代入される

コンパイラでは、A+=・・・の表現が現れた場合に、A のアドレスとその数値の二つをスタックに取得し、処理結果を A のアドレスに格納する。よって、計算に使用される A の値は、この式の計算の実行が開始される前の値であり、結果として格納される A のアドレスは、最初に現れた場所である。
例えば、
A=1000;
A+=(A=100);
を処理すると、A には 100 ではなく、1100 が格納される。

③配列への代入

配列[式１]=式２;

1) 式１の評価を行い格納アドレスを計算した上で、式２の評価を行う。よって、式２の中で式１の評価が変化するような処理が行われても格納先に影響しない。

[例] array[i=1]=i=2; array[1]に 2 が代入される。終了後、i は 2

Windows のための VS2005 の処理系では、式 2 の評価の後に式 1 が評価され値が格納され array[2]が 2 となる。
Android NDK の処理系では、式 1 の評価の後に式 2 が評価され値が格納され array[1]が 2 となる。

④代入文における型変換

1) 整数型として宣言された変数、配列に浮動小数が代入された場合には、小数点以下を切り捨てた数値を代入する。

```
int i;
i = 3.14; //3 が代入される。
i = 3.99; //3 が代入される。
i = -3.14; //-3 が代入される。
i = -3.99; //-3 が代入される。
i = 1e30;//オーバーフローが生じる（結果は保証されない）
```

＊F2I という仮想マシン上のコードが、スタックのトップ(32bit)を、float 型から int 型に変換しているが、その際にサイズチェックは行っていない。エラーチェックが必要な場合は、メタファイルの中で処理を記述する必要がある。
特定のデータファイルに添付し保存するメタファイルで動作が確認されている場合には問題はないはずである。

2) 宣言されていない変数、配列への整数の代入が行われた場合には、整数型として宣言されているものとみなす。

3) 宣言されていない変数、配列への浮動小数の代入が行われた場合には、整数型として宣言されているものとみなして、小数点以下を切り捨てた整数値を代入する。

4) 組込関数 scanf, printf, logf の書式文字列における型変換については、（１１）の④で解説する。

（８）プログラム制御

①if～else

```
if( 式１ ){
  プログラム２
}else{
  プログラム３
}
```

1) 式１が非ゼロであれば、プログラム２が、ゼロであればプログラム３が実行される。

2) 式１が空白であってはならない。

3) 式１として、if,for 等が用いられた場合には、エラー「不可解な因子」。

4) プログラム２、プログラム３は、複数の文から成る。

5) プログラム２、プログラム３は、空白であってもかまわない。

```
if( 式１ ){
}else{
}
```

6) プログラム２，およびプログラム３が１文から成る場合には、{}を省略できる。

```
if( 式１ )
  文２;
else
  文３;
```

7) else 以下の処理が不要である場合には、省略することができる。

```
if( 式１)
  文２;
```

8) 対応する if 文のない else は、エラー「不正な記述」とする。

9) 最も簡単な記述は `if(0);` である(コード上の意味はない)。

②switch、case、default、break

```
switch( 式１ ){
case 定数２:
   プログラム３
case 定数４:
   プログラム５
   break;
```

```
case 定数６:
  プログラム７
default:
  プログラム８
}
```

1) 式１の値に応じて、これに等しい case の次にジャンプする。

2) どの case とも一致しない場合には、default:の次にジャンプする。

3) break があれば「}」の次までジャンプする。

4) break がなければ、次の case：または default: の次にジャンプする。

5) プログラム２、３、４等は空白であっても構わない。よって、複数の case に対応した処理をまとめて表現することができる。

```
case3: case 7: case 9: case 23:
プログラム;
```

6) 複数の case に同一値の定数が指定された場合にはエラー「case 文の値が重複している」

7) switch 文がなく case 文が用いられた場合にはエラー「対応する switch 文がない」

8) case の次が定数式でない場合にはエラー「case 式が定数式でない」

③do～while

```
do{
  プログラム１
}while( 式２);
```

1) プログラム１は１回最低１回実行される。

2) 式２が非ゼロであれば、do{の次に戻り、プログラムを再度実行する。

3) 式２は空白であってはならない。

4) その他 if(式)と同様、式２に不適切な表現があってはならない。

5) プログラムが空白であっても構わない。その場合、式２だけが非ゼロである間、繰り返し実行される。

6) プログラム１が１行のみであっても、{} は省略できない。

④while 文

```
while( 式１){
 プログラム２
}
```

1) 式１が非ゼロであれば、プログラム２を実行し、再度判別式を評価する。

2) 式１が成立しなければ、ループを抜けて、「}」の次にジャンプする。

3) 式１が空白であってはならない。

4) 初回に式１がゼロであれば、プログラム２は一度も実行されない。

5) その他 if(式)と同様、式２に不適切な表現があってはならない。

6) プログラム２は空白であっても構わない。

7) プログラム２が１行の場合には、{} を省略することができる。

```
while( D(); );
    //D()の戻り値がゼロとなるまで呼び出しを繰り返す。
```

⑤for 文

```
for( 文１; 式 4; 文 3){
 プログラム 2
}
```

1) 文１は、ループを開始する前に必ず１回実行される。本処理系においては、「,」区切りで複数の代入などを記述することはできない。 i=j=k=0; のような表現は可能である。初期化が不要な場合には、文１は空欄のままで構わない。

2) 式４が、ループ開始に当たって評価され、成立するとプログラム３が実行される。成立しなければ、ループから抜ける。

3) プログラム２が実行される。この中で、break 文が実行されると、無条件でループを終了する。continue 文が実行されると、以後の処理は省略され、ループ先頭に戻る。

4) 文３は、ループ内部のプログラムが終了しループ先頭に戻った後に実行され、その結果を受けて式４の評価が行われる。文３は continue 文が実行された場合にも実行される。

5) 式４が省略されると、常に条件は成立して次のループを継続する。

for の後の( )内にある文 1、式４、文 3、プログラム２はいずれも省略することができ、最も単純な for 文は、for(;;); である（エンドレスループ）

（９）式と二項演算

式は、「式（二項演算子）式」　の形式であり、expression 関数が評価する。

二項演算子は、前の式と後ろの式の演算を行う。（括弧）なしに連続する式においては、以下の優先度で先に計算を行う

優先度７： 式＊式　乗算 式／式　減算 式%式　剰余
優先度６： 式＋式　加算 式－式　減算
優先度５：(比較式) 式１＜式２　式１が式２より小さい 式１＜＝式　式１が式２以下である 式１＞式２　式１が式２より大きい 式１＞＝式２　式１が式２以上である

優先度４：(等式・不等式) 式１＝＝式２　式１と式２が等しい 式１＜＞式２　式１と式２が等しくない ・整数の 0 と-0 は等しい（いずれも、全ビットがゼロ） ・浮動小数の 0.0 と-0.0 は等しくない（符号ビットが異なる） ・式が成立すれば値は１、成立しなければ０である。
優先度３：論理積 式１＆＆式２　式１が非ゼロかつ式２が非ゼロ
優先度２：論理和 式１｜｜式２　式１が非ゼロまたは式２が非常ゼロ
優先度１：代入 変数＝式　変数に式の値を代入し、その値が演算結果となる 変数＋＝式　変数に式の値を加算し、その値が演算結果となる 変数－＝式　変数から式の値を減算し、その値が演算結果となる 変数＊＝式　変数に式の値を乗算し、その値が演算結果となる 変数／＝式　変数を式の値で除算し、その値が演算結果となる

（１０）単項演算子

単項演算子は、式または変数の前後に置かれる。定数または括弧で括られた式の前または変数の前後に置かれた場合には、二項演算の評価よりも優先して評価される。

配列の１要素は、変数と同様に単項演算子の対象となる。

＋（式）　式の値を保存する －（式）　式の符号を反転する ！（式）　式がゼロならば１に、非ゼロならばゼロにする 変数＋＋　変数に１を加える（評価値は元の値） ＋＋変数　変数に１を加える（評価値は１を加えた後の値） 変数－－　変数から１を減じる（評価値は元の値） －－変数　変数から１を減じる（評価値は１を減じた後の値）

-a=2 は、エラーとなる（不正な左辺値）

-(a=2)では、a に２が代入され、式の値は-2 と評価される

a が２、ｂが－２の時、--a*--b は、－３と評価される：１×（－３）

--a+--b は、－２と評価される：１＋（－３）

--a---b は、エラーとなる（不正な左辺値）

--a-(--b)は、－４と評価される：１－（－３）

a---++b は、３と評価される：２－（－１）

1*++a+10*++a+100*++a+1000*++a は、６５４３

1*++a+(10*++a+(100*++a+(1000*++a)))は、６５４３

1000*++a+100*++a+10*++a+1*++a は、３４５６

下記の例では、i が任意の値であるとき、

[例]i=++i--; //エラー「不正な左辺値」（++i は変数ではないため、後置の++は不可）

[例]i=(i++)-(--i);//0 が i に代入される

[例]i=(i++)-(i--); // -1 が i に代入される

[例]i=(i++)-(i++); // -1 が i に代入される

[例]i=(i++)-(++i);//-2 が i に代入される

++i の処理は、変数のアドレスを１増加させた上で、その値をスタックトップに取得する
i++の処理は、変数のアドレスを１増加させた上で、その値-1 をスタックトップに取得する

（１１）組込関数

組込関数は、システム側で予め用意している関数であり、メタファイルの中からは、宣言することなく呼び出すことができる。組込関数の名称は全てシステムの予約語であり、同じ名称を用いて関数の定義、プロトタイプ宣言あるいは変数の宣言を行おうとするとエラーとなる「記述が不適切(予約語を用いた宣言等)」。

void 型の組込関数の戻り値を参照した場合にはエラー「void 型関数が式の中で使われている」。

戻り値のある組込み関数の戻り値を参照しなかった場合には、エラー「不正な記述」。この扱いは、メタファイルの中でコーディングされた関数とは異なっている。

この処理は、メタファイルの中でコーディングされた関数とは扱いが異なっている。コンパイラの statement 関数で文頭に左辺値となりうる関数名をトークンとして評価しており、ここで評価できない場合には、不正な記述としている。

組込関数には、保存データファイルへのアクセスのための関数、座標計算等に用いられる数値関数（三角関数等)、システム制御やデバッグのための関数が含まれる。

①数値関数

座標計算などのために、数値を引数として受け取り、数値を返す以下の組込関数を用意した。

quat _Q(float t, float x, float y, float z);

４の式(浮動小数型)から、一つの四元数を構築する。

quat _Q(“文字列”);

文字列から、一つの四元数を構築する。文字列は例えば、「0.0, 1.0, 2.0, 0.3」とする。

float _Qt(q);

float _Qx(q);

float _Qy(q);

float _Qz(q);

引数 q は四元数の式であり、四元数の成分を浮動小数として取り出す。

float sin(float); 正弦関数、引数はラジアン

float cos(float); 余弦関数、同

float tan(float); 正接関数、同

float atan(float); 逆正接関数、同

float SIN(float); 正弦関数、引数は度

float COS(float); 余弦関数、同

float TAN(float); 正接関数、同

float ATAN(float); 逆正接関数、同

float sqrt(float); ルート計算

quat exp(quat); 指数関数

複素数の指数関数と同様で、虚数成分がベクトルである。

引数がスカラー成分のみの時、結果もスカラー成分のみで指数関数。

quat ln(quat); 対数関数

複素数の対数関数と同様で、虚数成分がベクトルである。

引数がスカラー成分のみの時、結果もスカラー成分のみで対数関数。

float PI(); 定数（円周率）

quat IKE2DES(quat); 緯度経度標高→地球座標系

quat DES2IKE(quat); 地球座標系→緯度経度標高

地球座標系は、地球の中心を原点、経度ゼロの赤道上の点を X、東経 90 度の赤道上の点を Y、北極を Z 軸とする座標系である。標高は地球楕円体に、ジオイド（重力場の歪みによる補正値）を加えた等重力ポテンシャル面（平均海面を陸部にも求めた面）からの高さである。単位はメートルである。ジオイドのデータは、国土地理院から提供されているファイル（gsigeome_ver5.asc）を使用しているため、日本近傍のみ有効であり、それ以外ではゼロである。

quat IKE2NAT(int k, quat); 緯度経度標高→日本測地系（k は系統を示す）

quat NAT2IKE(int k, quat); 日本測地系→緯度経度標高（同）

quat IKE2WLD(int k, quat); 緯度経度標高→世界測地系（同）

quat WLD2IKE(int k, quat); 世界測地系→緯度経度標高（同）

k で示した系統は、旧国家座標系以来の、日本近傍を１９のエリアに区分した系統番号であり、各エリアの原点となる地点で地球楕円体に接する平面に地物を投影した座標値を、接点を原点とする二次元座標値として提供されている。k は 1 ～19 の整数値とする。

いわゆる地図で描かれた平面直交座標系であり、地球楕円体を、基準点で接する平面に投影した図形であるため、ユークリッド幾何学が成立するデカルト座標系ではない。原点から離れるに従い、高さ値は現実よりも大きくなり、また平面的な形状も伸張する。

多くの地域で、西暦 2000 年頃を境に測地系が日本から世界に変更された。日本測地系を世界測地系に変換するためには、緯度経度を経由するが、旧測地系と新測地系では同じ地点の緯度経度も異なっている。また、東日本大震災に際して数mに及ぶ土地の水平移動が生じた地域も存在している。

四元数で表現した場合、(0,緯度,経度,標高)で、緯度経度の度以下は 10 進数の小数で表現し、標高の単位はメートルである。平面直交座標系においては、(0,東西座標,南北座標,標高)で、単位はメートルである。地図を扱う処理系で、南北を変数 X、東西を変数 Y で表現すするデータ形式を採るものが多い点は注意を要する。

②システム制御、デバッグ用関数

```
void exit(int);
```

プログラムを終了する。たとえ他の関数から呼び出された関数の中であっても、呼び出し元に戻ることなくメタファイルの実行を強制終了する。デバッグ等の段階で、リカバリ不能なエラーが検出された場合等に使用する。最終的なメタファイルでは通常使用されない。

実行段階において、main 関数が return 命令により終了した場合には、return 命令の引数として指定された戻り値が execute 関数の戻り値となる。正常終了した場合には、1 を返すことを標準としており、main 関数が省略された場合や main 関数末尾の return 文が省略された場合も正常終了すれば 1 を返す。

実行段階で実行時のエラーが検出され exe_err 関数が呼び出されると、エラーカウンター(exe_err_ct)が加算される。現在の 4 実装形態においては、プログラムの実行を行っている execute 関数(インタープリタ)の中で、エラーカウンターが 0 以外であることが検出された場合には、次の機械語命令に進むことなくブレークし、0 を返している。

メタファイルを開発・検証するためのデバッグ

exit 関数が実行された場合、その引数がプログラムからのリターンコードとなる（通常は正常値である 1 、上記の失敗コード 0 以外の、エラーの特定に資する数値を指定する)。保存工程においてメタファイルのデバッグが十分に行われていれば、将来の利活用段階においても正常な解読終了が期待できる。

```
int printf(“文字列”);
int printf(“書式文字列”, 式);
int logf(“文字列”);
int logf(“書式文字列”,式);
```

これらは、デバッグのために有益な情報をログファイルに出力する。

ファイルを出力する処理系の場合には、printf 等の出力先は、ライブラリ関数が出力するものと同一のファイルになる。変換出力後のデータファイルにコメント行などを挿入することができる。ライブラリ関数がファイル出力を行わない処理系の場合には、専用のログファイルへの出力を行う。

logf は、ライブラリ関数がファイル出力を行う処理系において、出力ファイルとは別のログファイルに記録を行う場合に使用する。

書式文字列において「%」から始まる変換コードの指定に従って、第二引数の式の値を出力する。複数の式を出力したい場合には、複数回ライブラリ関数の呼び出しを行う。

書式文字列に変換コードがなく、第二引数がある場合にはコンパイルエラー「,の前に）が必要」。書式文字列で２以上の変換コードが用いられた場合にも、数値型が不明のため変換コードが無い場合と同じ扱いとしている（第二引数がある場合コンパイルエラー)。

書式文字列以降に二つ以上の式が記述された場合にはコンパイルエラー「引数は二つまで」とする。

戻り値は、標準 C に従い、出力できた文字数を返す。Windows 系 OS で、改行が CR/LF の２文字を出力した場合も、１文字としてカウントする。

書式文字列は、scanf と基本は共通であるため、④で解説する。

%n が指定された場合には、引数をデータ格納先に使用する。引数が単独の変数名ではない場合には、コンパイルエラー（「＋の前に）が必要」等）として処理する。書式文字列の末尾に%n が置かれた場合には、printf 関数の戻り値を使用する場合と同じ結果となる。

int input();

コマンドラインが利用可能な実行環境で動作している場合に、画面にプロンプト「>」を表示してオペレータの入力を待つ。オペレータが文字列を入力し改行すると、結果を文字列として取得し、整数値として評価して式の値として返す。同時にログファイルに「#手入力：[入力文字列]」という記録を残す。

③データアクセス関数

メタファイルが解読する保存データファイルは、コンパイルが終了し実行する時点では、既に開かれており、固定長の配列のようにアクセスすることができる。

メタファイルが動作する環境においては、予めデータファイルはオープンされており、固定長の配列と同じようにアクセスする。従って、メタファイルの中では、C 言語の stdio ライブラリの fopen 関数のような、ファイルを開く処理を記述する必要はない。

LEN 関数で取得できるファイルの長さを上限として、ポインタがファイル上のあるアドレスを指している。文字列の検索や数値型データの読み込みなどは、このポインタの後に続く部分に対して行われる。

int LEN();

保存データの本体であるファイルの長さをバイト数で返す。

入力用データファイルが開いていない場合には、ゼロを返すので、入出力動作切り替えができる利活用処理系において動作モード識別に使用することが可能。

int SIORI();

現在ポインタが指している位置を返す。先頭はゼロである。

int SEEK(int d);

d で指定されたアドレスにポインタを移動する。成功すれば 0 を返す。d が負値あるいはファイル長さよりも大きい場合には、-1 を返す。

int GETC();

１バイト取得して、ポインタを一つ進め値(0~255)を返す。ファイル終端の場合には、-1

を返す。

int GETINT(int d);

ポインタから d 文字（バイト）分のデータを取得し、10 進数を記述した ASCII 文字列として解釈して整数値に変換する。失敗した場合には、0x80000000 を返す。

int GETS();

次の改行までの長さを返す。ポインタは、改行コードの次に進める。改行コードを除去した文字列が、内部的なバッファに読み込まれる。

int scanf(“書式文字列”);

int scanf(“書式文字列”,変数);

データファイルから書式付きのデータを読み込むために古典的な scanf 関数を使用することができる。

標準の C 言語の scanf 関数では通常、変数を複数個指定でき、戻り値として取得に成功した変数の数を返している。scanf 関数における書式文字列の処理アルゴリズムは、例えば VS2005 の場合、添付のソースコード input.c により公開されている。これに対して本処理系においては、変数の数を 1 に制約している代わりに、変数を取得しない scanf 関数においても、パターンマッチの結果を返し、失敗した場合には、読み込みのポインタを処理前の状態に保っている。

・書式文字列とのマッチングを行った上で読み込みを行う。三次元データに多い、テキスト形式のデータファイルには scanf 関数により、シーケンシャルにアクセスすることができる。

・書式文字列と一文字でも不一致があれば、ポインタは不変で、ゼロを返す。変数は変化しない。

書式文字列と一致すれば、変数に値が取得される。

変数がなく、書式文字列の中に%で始まる変換指令がない場合は、単純な文字列の比較を行い、比較の成否だけを返す。

比較が全て成功した場合には、ポインタをその次に進める。

比較が一つでも失敗した場合には、ポインタは不動である。

変数は一つでなければならない

通常の C 言語のような、変数のアドレス記号&を省略することができる。

成功した場合には、１を返す。

④書式文字列

1）数値型

書式文字列の中で、%で始まる変換指示コードに基づいて、それに続く文字列が解析されて、必要であれば引数として指定された変数に結果が返される。

%c バイナリ値として取得し、変数に整数値として取得される。

幅指定は、%[1~4 の 1 文字の数字]c の形で行う。

scanf の場合には、指定したバイト数のバイナリ値として入力し、

int 型変数に入力する場合には、整数値に変換する。%c（であれば 1 バイトコード（ASCII コード等)、%2c であれば、２バイトコード（Shift-JIS、UNICODE 等）として取得される。

%4c として４バイトを取得し float 型変数に入力する場合には、ISO/IEC/IEEE 60559-2011 に従う 32 ビットの単精度浮動小数として解釈した数値に変換する。

出力系の場合、「%80c¥n」等と記述された場合には、80 桁の中に、引数で指定された文字コードを右詰で出力し、余白にはスペース(コード:32)を補って１行を出力する。

%o　８進数として整数値を読み込み、変数に整数値を代入する

%d　10 進数として整数値を読み込み、変数に整数値を代入する

%x　16 進数として整数値を読み込み、変数に整数値を代入する

%f　浮動小数値を読み込み、変数に代入する

%qt %qx %qy %qz 浮動小数値を読み込み、引数の quat の成分に代入する。この場合、引数は一つだけであって、書式文字列中に変換指示を最大４まで含めることができる。

%n その位置の、変換開始位置からの相対位置を取得する。出力系の printf, logf の場合であっても、引数には数値が入力される。但し、引数は一つしか認められないため、固定長の文字列の長さを知る意味しかない。

%s または %[・・・]　組込関数 scanf により文字列を読み込み、記号表に登録する。記号表における登録番号（添字）を整数型の第二引数に返す。次の処理で、「＊変数」としてこの変数アドレスに数値を代入し、参照することができる。また、printf、logf 関数の書式文字列の%s に対応して、＊変数を指定すると、記号表に登録された文字列を表示することができる。

鉤括弧[・・・]内の書式表現について

マッチする文字を鉤括弧[・・・]内に列挙する。この文字にはスペース「 」や改行「¥n」も含めることができる。

例１：scanf(“%[0123456789ABCDEF]”, tid);　１６進数の表現を読み込む。

連続する場合には、ハイフン「-」でつないで省略することができる。

この場合、読み込んだ文字列を変数表に登録して添字を tid に返す。

例２：scanf(“%*[0-9A-F]”);　１６進数の表現を読み込む（例１と同じ効果)。

マッチしない文字を鉤括弧[・・・]内に、「^」文字に続けて列挙する。

例３：scanf(“%*[^ 0-9A-F]”);　１６進数の表現以外を読み込む（例１と同じ効果)。

この場合、無視したい部分を読み込んで、読み捨てるような効果がある。

「9-0」のように指定された場合には、「0-9」や「0123456789」と同等である。

例４：scanf(“%*[abc^def]”);　abc および def 以外を読み込む。

「abc」は「def 以外」に含まれるので、[^def]と変わりない。

例５：scanf(“%*[]”);　文字「]」だけから成る文字列を読み込む。

「]]]]]」など。最初の「[]」だけでは意味がない（マッチする文字が指定されていない）ため、「[」に続く「]」は鉤括弧閉じではなく、マッチする文字として解釈する。

例６：scanf(“%*[][]”);　文字「[」または「]」だけから成る文字列を読み込む。

「[][[[[]]]]」など。最初の「[]」だけでは意味がない（マッチする文字が指定されていない）ため、「[」に続く「]」は鈎括弧閉じではなく、マッチする文字として解釈し、それに続く文字「[」もマッチする文字として認識する。

例７：scanf(“%*[%d]”); 文字「%」または「d」だけから成る文字列を読み込む。

整数型に変換する「%d」としては解釈しない。

「%[文字集合]は、正規表現のスタイルを不完全に取り入れた表記法である。しかし、以下のような点に関して、処理は曖昧である。

「%[]」鈎括弧の内部が空である場合には、無意味であるため、「]」文字がマッチ文字として指定されたと解釈して解釈を続行する。「%[]]」ならば、複数個の「]」とマッチする。「%[][]」ならば、複数個の「[」または「]」から成るパターンとマッチする。
VC-1C においては、落ちる。VC-2V においては、80 文字を読み込む。
「%[いろはに]のように２バイト系の文字を並べた場合には、動作は保証されない。
「%[^]として、除外する文字の集合が空である場合には、動作は保証されない。
「%[][]」のように、鈎括弧[]の内部に鈎括弧文字自体が含まれている場合には、動作は保証されない。
「”%[abc”」のように鈎括弧[ ]が閉じていない場合には、動作は保証されない。

データファイルの中で非常に長い文字列が属性として記述されているような場合には、scanf を使用せずに、先頭アドレスと末尾アドレスを取得した上で、GROUP 関数などを用いてデータベースに属性を格納する。

出力系の printf, logf においては、「%s」の形はランタイムの記号表に登録された変数名を表示する場合にのみ使用できる。出力系で「%[・・・]」が指定された場合には、引数を参照することなく、そのまま文字列として出力される。

%s スペース、改行、タブ等の区切り文字までを取得する

[・・・]列挙した文字と一致する文字コードが続く範囲だけを取得する

[^・・・]列挙した文字以外の文字コードが続く範囲だけを取得する

注意：

文字列「A3=3;改行」を%s 変換すると、「A3=3;」が文字列として取得される。

「＝」の前までを取得したい場合には、%[^=¥n¥s]等とする。

2) 非変換指示

%の次がアスタリスク(*)である場合、変換だけ行い変数への代入は行わない。

この場合、引数は不要であり、書式文字列の中に変換指令がいくつあってもよい。

[例] scanf(“(%*f,%*f,%*f,%*f)”); (0.1,0.2,0.3,0.4)の場合→１を返す。

printf において、非変換指示が行われた場合、実行結果は保証されない。

[例] printf(“%*d”); →処理系より不定

3) エスケープ文字

%%は、変換指示ではなく、１文字の％として処理する。

なお、書式文字列は、コンパイラに入力された段階で、以下のエスケープ規則に従い解釈され、その結果がリテラル定数としてメモリ領域に記録される。従って、この書式文字列の中には、文字として印刷や表示することのできないコードも含まれている。

¥¥　１文字の「¥」(0x5c)

¥%　１文字の「%」(0x25)
¥”　１文字の「“」(0x22)
¥’　１文字の「‘」(0x2c)
¥t　水平タブ(9)
¥n　改行(10 つまり 0xa)
¥r　復帰(13 つまり 0xd)
¥x　続く２文字までを１６進数として読み込んだ１バイトコード
¥x の次が[0-9A-Fa-f]ではない場合にはエラーとする(例「¥xg」)
¥x の次が[0-9A-Fa-f]であり、その次が異なる場合には、１桁の１６進数として入力
¥x に続く１６進数として解釈可能なコードが３桁以上続く場合：２桁のみ解釈
(例「¥x300」→「00」、「¥x3333」→「333」)
¥0　ヌル(0)　書式文字列の中に¥0 が置かれた場合、書式文字列はそこで終了するため、以後の文字列は無視される。
例：「printf(“abcd¥0efg”); 」→「abcd」それ以外：¥は無かった場合と同等
この原則によると¥1~¥7 は　本来は、文字「1~7」として処理されるべきであるが、現在の処理系では、文字「¥x01～¥x07」として処理されている。

次に、scanf、printf 等が実行される段階で、%で始まる変換規則が解釈される。

書式文字列のエスケープシーケンスにより「%」に変換された場合には、scanf, printf においては、当初からの「%」と同様に評価する。
例「printf(“¥%d”, 3);」→「printf(“%d”, 3);」→「3」
例「printf(“¥x35¥x64”,3);」→「printf(“%d”, 3);」→「3」
%%　１文字の「%」
例「printf(“¥x35¥x35”);」→「printf(“%%”);」→「%」
なお、内部処理において「%%」のパターンを一時的に「¥x01¥x01」のパターンに変換しているため、書式文字列に当初から「¥x01¥x01」のパターンが存在した場合には警告を発している。

4) 書式文字列の中の空白文字
任意数の空白文字とマッチする。またタブコード、改行コードともマッチする。
例えば、ポインタが前行末にある時に、改行、タブ、スペース以外の最初の意味のある文字にアクセスする場合には、” %c”のように、書式文字列の最初にスペースを入れる。
5) 文字数の指定
%2c　２バイト分を、１６ビット整数または浮動小数に変換した値
%4c　４バイト分を、３２ビット整数または浮動小数に変換した値
%15s　１５文字までの文字列

⑤データファイル入力例

三次元データに多い、テキスト形式のデータファイルには scanf 関数により、シーケンシャルにアクセスすることができる。現在のアクセスポイントが保持されており、その値は、SIORI()関数により整数値として取り出すことができる。また、SEEK(int)関数により、アクセスポイントを設定することができる。

一方、C 言語のコンバータで、fgets 関数により一度配列にデータをコピーした上で、文字列関数で比較を行うような処理は用意していない。その代りに、scanf 関数の機能を強化してあり、書式文字列を用いて、パースを行う考え方を採っている。

scanf(書式文字列)は、現在のアクセスポイントから後の文字列が書式文字列と一致するかどうかを戻り値で示す機能を追加した。更に、一致しなかった場合には、アクセスポイントを、処理前の状態に復原するように明確化した。

一方、scanf 関数で取得する数値の変数は 1 だけに限定した。よって、複数の数値等をデータファイルから取得するためには、複数回 scanf 関数を呼び出す必要がある。

三次元形状を記述したデータファイルには、様々なフォーマットで座標値が記述されている。これを取得することが必要である。

例えば、dxf 形式のファイルでは、

```
1
30.0
2
35.0
3
1.5
```

といった形式で、X,Y,Z 座標が記述されている。

リスト 4-1-1　座標値の取得処理例

```
scanf("%d¥n", ia);
switch(ia){
   case 1:
   scanf("%f¥n",X);
   break;
   case 2:
   scanf("%f¥n",Y);
   break;
   case 3:
   scanf("%f¥n",Z);
   break;
   default:
   logf("ERROR unknown IA");
}
```

ここで、C 言語では、scanf("%f", &floatvalue)と書くが、本処理系においては、

scanf("%f",floatvalue);

という書法を用いることとした。

一般に、高度な形式を有するデータファイルのためのメタファイル中では、キーワードのマッチングを行う処理は頻出する。これは、以下のように記述可能である。

リスト 4-1-2 キーワードのマッチング処理

```
int findkeyword(){
   if(scanf("キーワード１")) return 1;
   if(scanf("キーワード２")) return 2;
     ・・・・
   Return 0;
}
```

また、メタファイル中で、記号が用いられる場合がある。例えばシンボル名称、変数名などである。これらを登録する場合には、次のようにプログラムすれば良い。

リスト 4-1-3 データファイル中のシンボルの処理例

```
int head[MAXLEN],tail[MAXLEN];
int nsymbol;
int table(h,t){
   int i;
   for(i=0;i<nsymbol;i++){
      if(tail[i]-head[i] != t-h) continue;
      for(j=head[i];j<tail[i];j++)
            if(data[head[i]+j] != data(h+j) break;
      }
      if(j<tail[i]) return i;//マッチした
   }
 head[i] = h;
 tail[i] = t;
 nsymbol++;
 return i; //新規登録したシンボル
}
```

このメタファイル中の処理において、作成するテーブルは、シンボルの文字列ではなく、そのシンボルのデータファイル中における初出位置を登録している。

データファイルの中でシンボル（変数など）が用いられている場合には、上記のようにプログラムの中で作成したシンボル・テーブル(head, tail)に更にデータ値などの属性を格納するテーブルを添える方法が可能であるが、本処理系のコンパイラ部分のシンボルテーブル管理機能をインタプリタ実行時にも利用可能であるため、次のようなメタファイルの記法が可能である。

例えば、一つの座標値が、

```
POINT0001:=1.0,2.0,3.0[改行]
```

という１行で定義されていたとする。

リスト 4-1-4 データファイル中のシンボルの記号表を用いた処理

```
int p;
quat q;
scanf("%[^:]=",p);   //シンボル POINT0001 を取得する(a)
scanf("%qx,%qy,%qz¥n",q);          //座標値を取得し変数 q に格納する(b)
*p=COORD(q);   //登録したシンボル POINT0001 のメモリ部に座標値 ID を格納する(c)
```

ここで、(a)の処理において、スキャンされた変数名「POINT0001」がシンボルテーブルから検索され、一致するものがなければ新たなエントリが作成され、その ID が変数 p に格納される。次に、(b)でスキャンされた引数（座標値）を用いて、COORD ライブラリ関数で座標値を取得し、q に代入する。この座標値を(c)でライブラリ関数 COORD を用いて登録し、返された座標値の ID を、*p が指定するシンボルのデータ格納領域に記憶し、以後の処理に利用する。

ライブラリ関数 COORD の使用方法を、従前の処理系と比較すると、以下のような違いがある。

１．景観シミュレータの外部ファイル（LSSG 形式）における記述

P=COORD(1.0, 2.0, 3.0);

インタプリタ IP が、(1.0, 2.0, 3.0)の位置座標のオブジェクト P を一つ作成する。

２．仮想コンバータのメタファイルにおける直接記述

P=COORD(1.0,2.0,3.0);

コンパイル後の実行段階で、ライブラリ関数 COORD に３の数値が引数として渡され、実行結果として返された座標値の ID（整数）が整数型の変数 P に代入される。

３．仮想コンバータのデータファイルにおける解読処理

P=COORD(1.0,2.0,3.0);

これを、メタファイルにおいては、次のように解読処理する。

scanf("%[^=]", i);　　・・・「P」という変数名を記号表に登録し ID を i に代入する。

scanf("=COORD(%qx,%qy,%qz);", q);　　・・・座標値を、quat 型変数 q に取得する。

*i = COORD(q);　　・・・ライブラリ関数 COORD が実行され、返された座標の ID 値を P のデータ格納域に格納される。

例えば、VERTEX 関数にこの座標値格納変数を渡す場合には、VERTEX(*i);を実行する。

（１２）ライブラリ関数

組込関数の内、利活用処理系を駆動するための関数を、特にライブラリ関数と呼ぶ。ライブラリ関数については、関数型と引数リストの形式が定められているが、この規約を守った上で、内部の処理を目的に応じて将来自由に設計し実装することができる。

メタファイルの記述の中では、データファイルを解析して形態要素を取り出した上で、ライブラリ関数を呼び出して形状の生成を行う。その結果がどのように利用されるかは、ライブラリ関数の実装内容によって異なる。個別のライブラリ関数については、４－２で解説することとし、本項では代表的な COORD 関数を例に、実装による違いを解説する。

①モデル構築系ライブラリ関数

国土交通省版・景観シミュレーションシステムにおいて建物や地形の形状を記録するためフォーマットである LSS-G 形式を構成するコマンド群を原型として、メタファイルの記述を容易かつコンパクトにするための関数を追加したものである。

コマンドの基本形には、

変数＝コマンド（引数, ・・・）;

の形式のオブジェクト定義型と、

コマンド（引数, ・・・）;

の形式の属性追加型の２種類がある。ライブラリ関数も、これと同様の方法で使用するが、LSS-G コマンドにおいては定数（数値）のみを引数としていたコマンドに、変数や式を引

数として渡すことを可能としたため、より柔軟に形状や色彩などの定義を行うことができる。

[例] 座標値を定義する「COORD」の LSS-G コマンドとメタファイルのライブラリ関数における違い
LSS-G 形式におけるコマンドでは、
P0=COORD(0.0,1.0,2.0); //座標値を表すオブジェクト P0 を定義する。引数は定数でなければならない。
メタファイル用のライブラリ関数では、整数型の関数である。
P[i]=COORD(x, y, z+1.5); //結果（座標値の ID）の格納先を配列としたり、引数を変数や関数を含む式とすることができる。また、四元数 quat q に座標値が格納されている場合には、
P=COORD(q); //引数は quat 型変数
という記述も可能である。

ライブラリ関数の戻り値および引数の数値型はあらかじめ定められており、インクルードファイル等でプロトタイプ宣言する必要はない。

ライブラリ関数が実行された場合に、処理系がどのような動作を行うかは、例えばこの COORD 関数が実行された場合に実際の処理を行うプログラムに依存する。メタファイルにおいて

変数名＝COORD(式 1, 式 2, 式 3);

という文が実行された場合について、３．３～６で解説した４実装例を比較すると、

1) VC-1C においては、cci_ip.c が定義する COORD 関数を使用する。この実装では、スタックから３の座標値を取り出して、定義済の頂点リストとの照合を行い、すでに存在すればその ID を、また一致するものがなければリストに追加した上で、新たな ID 番号を生成し、スタックのトップに返している。更に、新たな ID が生成された場合に限り、P%d=COORD(%f,%f,%f);という形式で座標を定義する LSS-G コマンド（引数は定数）をファイル出力している。結果的に、任意のファイル形式を LSS-G 形式に変換するコンバータ機能を実現している。なお、この処理系はメタファイルの記述においてオブジェクト（変数名）とは無関係に「P 整数」、例えば「P32」というオブジェクト名を自動的に生成してファイル出力している。また、同じ座標値を引数とする複数回の関数呼び出された場合に、出力ファイルには一度しか出力が行われず、スタックには同じ戻り値が返され、変数への代入などの処理が行われる。

[同じ頂点座標が重複して定義された場合の処理例]
メタファイルにおける記述

```
a=COORD(0,0,0);
b=COORD(0,0,0);
c=COORD(0,0,0);
va=VERTEX(a);
vb=VERTEX(b);
vc=VERTEX(c);
```

は、出力ファイルにおいて、以下のように変換される。

```
P0=COORD(0.000000, 0.000000, 0.000000);
V0=VERTEX(P0);
V1=VERTEX(P0);
V2=VERTEX(P0);
```

ここで、P0,V0,V2,V2 などの変数名は、コンバータ（利活用処理系としての VC-1C におけるライブラリ関数の実装）が自動的に与えているものである。

2) VC-2V、VC-3M においては、cci_dml.c が定義する COORD 関数を使用する。この実装では、同様にスタックから３の座標値を取り出してメモリ上に二分木ソートした座標値のリストを作成し、登録済の座標値についてはその ID を、また新たな座標値の場合にはリス

トに追加した上で新たな ID を与え、スタックに返す。処理系では、引き続きこの ID を引数として面や線の定義を行い、画面表示に使用している。

3) VC-4D においては、cci_sql.c が定義する COORD 関数を使用する。この実装では、同様にスタックから 3 の座標値を取り出し、これを用いて、次のような SQL 文を生成する。

「SELECT id FROM coord WHERE　X ＝(式 1 の値), Y =(式 2 の値), Z =(式 3 の値)」。

これによりデータベースの座標値を格納テーブル「coord」の検索を行い、一致する座標値があればそれをスタックに返す。なければ、次の SQL コマンドを発行してこの座標値をテーブルに追加した上で、ID を返す。

「INSERT coord VALUES(式 1 の値, 式 2 の値、式 3 の値)」

これにより、大規模で複雑な三次元形状を、データベース上に格納することができ、WEB アプリである VC-4D（三次元データ保管庫）は、それを用いて次の段階で、新たなメタファイル（出力用）を用いて別の形式のファイルを出力する処理を行っている。

LSS-G 形式に由来するモデル構築のためのライブラリ関数は上記の COORD 関数を含めて３０種類あり、それぞれの用法については、４－２（３）で解説する。

②シーン記録用ライブラリ関数

景観シミュレーションシステムにおいて、LSS-S 形式のファイルの中で用いられているコマンド群である。上記①モデル構築関数において、記録対象である建物等に固有の三次元形状や色彩等を定義することができる。これがどのように見えるかは、視点位置や、光源の状況、時刻等によってことなる。このような環境条件を記述するための関数群である。

利活用処理系の内、現場に携行してデータを表示し、シャッターで記録を取る機能を有する VC-3M（むかしめがね）において、シャッターをユーザーが操作した場合に、携帯端末の位置や姿勢や日時・時刻を記録に取り、これを後刻再生表示するために使用している。

この閲覧記録を取ることにより、建物等のモデルの記録が変更を受けることはない。

③モデル要素取得用ライブラリ関数

モデル構築関数①によりメモリやデータベース上に構築された、表示可能な建物等のデータから、立体や面や頂点座標や色彩などの要素を系統的に取り出すための関数である。この関数と printf 関数を用いて、任意形式のデータファイルを出力することができる。

この関数は、保存対象であるデータファイルの解読においては使用されることがないため、データファイルに添付して保存されるメタファイルの中では使用されることは想定していない。保存工程から長期間経過した後の利活用工程において、新たな別のデータ形式によるデータファイルが必要である場合に、これらの関数を用いて出力ファイルを作成する処理を記述するための、利活用処理系のために用意したものである。

これまでに例示的に試作した 4 種類の利活用処理系の内、SQL データベースを構築する VC-4D のために作成し、データベース上に展開され構築された建物のデータを系統的に取り出すために使用される。更に、メモリ上にデータを構築する VC-2V においてもこれと互換の動作を実現した。

（１３）メタファイルのデバッグと、エラーメッセージのための処理

システムが処理中に出力するエラーメッセージには、メタファイルのコンパイルエラーと、コンパイルが成功後、実行時に発生するエラーメッセージがある。

この他に、デバッグ段階で、メタファイル中で logf 関数を用いてメッセージを出力し、デバッグに活用することができる。

このほか、利活用形態の中でエンドレス状態を監視して強制的に終了するデバッグ手段を組み込むことができる。

①コンパイルエラー

２－５（４）の表 2-4-1 に一覧表を掲載した。

コンパイルエラーは、メタファイルの１行について複数検出された場合であっても、最初の一つしか表示しない。コンパイルエラーが生じる場合には、一つの行をなるべく細かく分割して、豊富なエラーメッセージを得ることが有効な場合がある。

```
T=func(c=a/b);
```

この行では、ライブラリ関数 T()と同じ文字列 T が変数として使用されている。この場合、「＝の前に（が必要です）というメッセージがトークン解析の段階で出力されるので原因がわかりにくい。このような場合、

```
T=
func(
  c =
  a/b
);
```

と分解して再度コンパイルすることにより、発生源を更に特定することができる。

②ランタイムエラー

２－５（４）の表 2-4-2 に一覧表を掲載した。

ランタイムエラーは、メモリ等のリソースの不足に起因する場合や、データによって不適切な処理（例えばゼロ除算）が発生する場合に生じる。

関数への再帰的な呼び出し（コンバータには多い）において、メタファイルの欠陥により階層が深くなりすぎて局所変数に割り当てるメモリが不足するような場合にも、コンパイルは正常に終了した後、ランタイムエラーが生じる。

③メタファイルの中でコーディングするエラー処理

メタファイル中で logf 関数を用いてメッセージを出力し、デバッグに活用することができる。また、exit 関数を用いて、任意の場所でプログラムを中断することができる。

長時間処理が終了しないような場合には、エンドレスループが発生している可能性があり、ループ回数のカウンタを設けて監視するデバッグ方法が有効である場合がある。

④オペレータによる強制的なブレーク

利活用システムの中でデバッグ環境を提供することもできる。VC-2V においては、プロ

グレス・インジケータを設けて大きなデータファイルへのアクセス状況を監視し、必要であれば画面操作によりブレーク（強制終了）することができる。強制終了されたことを終了時のメッセージに表示すると共に、その時点でのプログラムカウンタ、データファイルへのアクセス番地、およびデータファイルの長さを表示している。

この VC-2V プログラムのブレーク動作機構については、３－４で解説した。強制的ブレークによりメタファイルによる解読処理が中断した場合には、最後に、ブレークポイントによる中断であることを含むメッセージをログファイルに出力する。

```
#C コンパイラ　ログ
# 仮想コンバータ・メタファイル：[C:¥@keikan¥kdb¥geometry¥C--サンプル¥固定形状¥頂点カラー0.c--]
# 変換対象ファイル：[C:¥@keikan¥kdb¥geometry¥C--サンプル¥固定形状¥頂点カラー0.c--]
#C:¥@keikan¥kdb¥geometry¥C--サンプル¥固定形状¥頂点カラー0.c--を fileOpen した。
#fileClose した。
#(l=3) warning: main()がない形式として再解釈(関数の外で変数が左辺値)
#C:¥@keikan¥kdb¥geometry¥C--サンプル¥固定形状¥頂点カラー0.c--を fileOpen した。
#fileClose した。
#コンパイル成功
#----------実行開始----------
#ランタイム  printf 出力先：outfile
#----------実行終了----------
#ブレークによる強制終了            //画面操作による強制的中断で終了した場合
#終了コード：0
#終了時  PC:116, SIORI=0           //強制的中断時点での状況を報告
```

なお、現段階でメタファイルを作成しデバッグする作業環境は十分整備されているとは言い難い。メタファイルで使用できる組込関数やライブラリ関数は、小さな処理系で実装できるシンプルなものとなっており、言語としてデータファイルを記述するようなコマンドや文法とはなっていない。目的が長期保存であり、未来のマシンや OS 上で容易に再現できるコンパイラ処理系を目指したためである。

一方、メタファイルのためのデバッガと、それと連動するテキストエディタがあれば作業は能率化されるであろう。そのような補助的なツールは、主役である保存するデータ、それに添付するメタファイル、利活用のための将来の処理系の３者とは別に、その外側に例えば現在普及している Windows 上で動作するアプリケーションとして開発することが可能であり、それを保存データに添付する必要はない。VC-1C_D.exe を単独で起動した場合に、簡単なエディタとデバッガを使用することができる。

そのような、保存データ（データファイルとメタファイル）そのもの（仕様等）を変えることなく、保存工程における作業能率を向上するための更なる改善は、本稿以後の課題としたい。

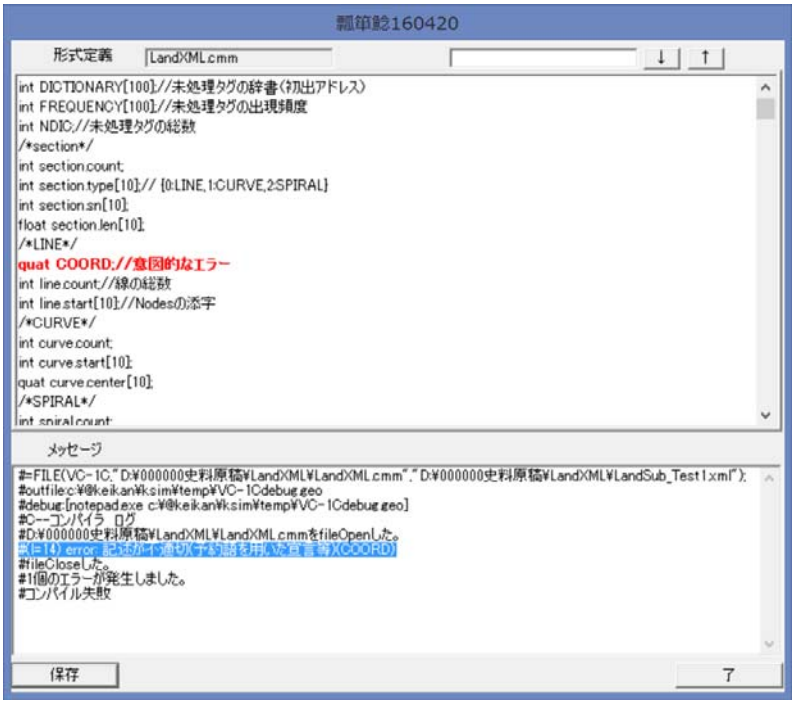


図 4-1-1 メタファイル作成のための試作的なエディタ・デバッガ

４－２．ライブラリ関数

本処理系は、C 言語を簡略化したコンパイラに、データファイルを解読し三次元モデルを記述するための組込関数を特徴としている。組込関数の内、将来の利活用処理系の構築のために様々な処理内容を再定義することを想定した関数群を、ライブラリ関数と呼ぶ。

データファイルの解読方法を記述するメタファイルの書法の基本は、C 言語に準拠した文法規則によるデータファイルの解読と、景観シミュレータの LSSG 形式のコマンドを母体としたライブラリ関数によるデータ構築から構成されている。本節では、個々のライブラリ関数の使用法を主に解説する。文法の概要（２－１）と詳細（４－１）、現在のライブラリ関数の一覧（３－２(2)）を示した。

データファイルの長期保存工程においては、将来(例えば数十～数百年先)における利活用工程でどのような方法が必要かつ可能かについて予想することできないが、少なくとも現時点で構想できる４種類の実用的な実装例を作成した。新たな利活用処理系の開発とは、本質的には所定のライブラリ関数に対する新たな処理内容の実装である。将来新たな利活用システムを開発し、処理系の動作（ファイル出力、画面表示、データベース入出力、マシンコントロール等）に応じて必要となるライブラリ関数毎の処理内容を実装しようとする者に、これまでに試作し動作確認した４種類の実装形態の構築過程を解説することにより、将来の新たな利活用処理系の構築のための参考資料になると考える（３－２(4)）。

また、当面の保存工程においても、これら実装例を活用して、メタファイルの指示に従って正しくデータファイルが読み込まれることを検証する具体的作業が可能である。

本節では、３－２(2)に掲げた個々のライブラリ関数について、メタファイル作成の上で参考となる実際の動作や特徴等について、詳細に解説する。

メタファイルの記述においては、大文字と小文字は区別した上で、これらのライブラリ関数の名称は、コンパイラがトークンとして処理している。従って、ライブラリ関数と同名の変数や配列や関数を、メタファイルの中で使用することはできない。例えばライブラリ関数に存在する T（）関数と同名の T という変数を宣言しようとすると、コンパイル・エラーが生じる。

（１）データ構築系のライブラリ関数

データファイルの解読する方法を記述する形式定義ファイル（メタファイル）を作成するための、データの長期保存にとって本質的な部分を担う中核的な関数群である。

```
int COORD(x,y,z);
```

```
int COORD(q);
```

三つの浮動小数（式）または一つの四元数（式）から座標値を定義する。戻り値は座標値の ID であり、それまでに入力されている座標値と比較するか否かは処理系に依存する。

```
int COLOR(float r, float g, float b);
```

```
int COLOR(float r, float g, float b, float a);
```

色彩を定義する。引数は式で、r は赤、g は緑、b は青, a はα値（不透過性）を定義する。

それぞれ、0.0～1.0 の値を持つ。a が省略された 3 値だけの引数の場合、α値が 1.0 と指定された場合と等価である。色彩は、面または頂点に対して定義する。頂点に色彩を定義するのは、タイル分割された面の色彩変化を滑らかに表現するような場合である。戻り値は色彩値の ID であり、それまでに入力されている座標値と比較するか否かは処理系に依存する。

int NORMAL(float x, float y, float z);

法線ベクトルを定義する。引数の x, y, z（式）は。長さが 1 のベクトルの 3 成分を表す。法線ベクトルは、通常は面に対して定義するが、回転体や多面体で近似した自由形状の立体の表面を滑らかに表示するために使用とする場合には、隣接する面が共有する頂点に対して定義する。三つの浮動小数から法線ベクトルを定義する。戻り値は法線ベクトルの ID であり、それまでに入力されている法線ベクトルと比較するか否かは処理系に依存する。

int TCOORD(float u, float v);

テクスチャ座標を定義する。テクスチャ画像の横方向を u、縦方向を v とし、左下隅と右上済をそれぞれ(0.0, 0.0)、(1.0,1.0)とする二次元の座標系において、二つの浮動小数(式)から当該頂点に対応する画像上の点の座標値を定義する。戻り値はテクスチャ座標の ID であり、それまでに入力されているテクスチャ座標と比較するか否かは処理系に依存する。

int TEXTURE(“文字列”);

画像ファイル名からテクスチャを定義する。戻り値はテクスチャの ID であり、それまでに入力されているテクスチャと比較するか否かは処理系に依存する。

int MATERIAL(“文字列”);

材料定義ファイル名からマテリアルを定義する。戻り値はマテリアルの ID であり、それまでに入力されているマテリアルと比較するか否かは処理系に依存する。

int VERTEX(P, N, T, C, VP);

頂点を定義する。引数 P,N,T,C,VP は、整数型の変数名または配列の要素である。

P は、頂点座標の ID を格納する。

N は、頂点の法線ベクトルの ID を格納する。

T は、頂点のテクスチャ座標の ID を格納する。

VP は、穴あき図形を表現しようとする場合の、仮想線の渡り先の座標点の ID を格納し、座標点 P から座標点 VP に移動する辺は仮想線と見なして、ワイヤーフレーム表示から削除している。

VERTEX 関数の引数の内 P を除き、必要ないものは、省略することができる。例えば、法線、テクスチャ、色彩を使用せず、仮想線だけを定義する場合には、VERTEX(P,,,,VP);と記述することができる。また、空間座標とテクスチャ座標だけを必要とし、他を必要としない場合には、VERTEX(P,,C);と記述することができる。

また、以後の引数が全て空白である場合には、引数が 5 より小さくともかなわない。

int FACE(v1,v2,v3,・・・);

頂点列から面を定義する。引数は頂点を格納した変数または配列要素であり、引数の数は無制限である。空白の引数は許されない。面の ID を返す。最後の頂点と最初の頂点の間には辺が一つ定義される。同じ頂点が指定された場合には、重複した頂点となり異常な面が生成する。頂点の数は１以上であれば３に満たなくとも良い。頂点列が左回り（反時計回り）に見える側が、面の表側である。

void FACE_VERTEX(f, v);

引数は変数または配列の要素であり、定義済の面 f の頂点列の末尾に、頂点 v を追加する。

void FACE_NORMAL(f,n);

引数は変数または配列の要素であり、定義済の面 f に定義済の法線ベクトル n を指定する。

void FACE_COLOR(f,c);

引数は変数または配列の要素であり、定義済の面 f に定義済のカラーc を指定する。

void FACE_TEXTURE(f,t);

引数は変数または配列の要素であり、定義済の面 f に定義済のテクスチャ t を指定する。

void FACE_MATERIAL(f,m);

引数は変数または配列の要素であり、定義済の面 f に定義済のテクスチャ m を指定する。

int LINE(v1,v2,v3,・・・);

頂点列から線を定義する。引数は頂点を格納した変数または配列要素であり、引数の数は無制限である。空白の引数は許されない。面の ID を返す。最後の頂点と最初の頂点の間には線分は生成しない。閉じた折線を作成するためには、最初の頂点を再度、最後に引用する必要がある。

void LINE_VERTEX(l,v);

引数は変数または配列の要素であり、一度定義した折れ線 l の頂点列の末尾に、頂点 v を追加する。

void LINE_NORMAL(l,n);

引数は変数または配列の要素であり、定義済の折れ線 l に定義済の法線ベクトル n を指定する。

void LINE_COLOR(l,c);

引数は変数または配列の要素であり、定義済の折れ線 l に定義済のカラー c を指定する。

void LINE_TEXTURE(l,t);

引数は変数または配列の要素であり、定義済の折れ線 l に定義済のテクスチャ t を指定する。

void LINE_MATERIAL(l,m);

引数は変数または配列の要素であり、定義済の折れ線 l に定義済のテクスチャ m を指定する。

int GROUP();

int GROUP(“属性文字列”);

一つの意味あるまとまりを意味するグループを定義する。このまとまりは、面や線分の集合体、他のグループを含むことができる。また、表示されるべき実体をもたない空のグループや、属性文字列だけをもつグループであっても構わない。

void GROUP_FACE(g,f1,f2,f3・・・);

引数は変数または配列要素であり、定義済のグループ g に定義済の面群 f1,f2,・・・を指定する。ある面は一つのグループにしか帰属することができない。

void GROUP_LINE(g,l1,l2,l3・・・);

定義済のグループ g に定義済の折れ線群 l1,l2,・・・を指定する。ある折れ線は一つのグループにしか帰属することができない。

void GROUP_TEXTURE(g,t);

定義済のグループ g に定義済のテクスチャ t を指定する。グループに指定された面や折れ線にテクスチャが指定されている場合には、そちらが優先される。

void GROUP_MATERIAL(g,m);

ID が g のグループに ID がｍのテクスチャを指定する。グループに指定された面や折れ線にマテリアルが指定されている場合には、そちらが優先され、この指定は無視される。

void GROUP_ATTR (int g, int s, int l);

ID が g であるグループに対して、データファイルから取得する属性文字列を指定する。文字列は、データファイル中の開始アドレス s と、長さ l（バイト数）により特定する。グループに属性文字列が既に指定されている場合には、そのあとに別の文字列として追加される。一つのグループは、属性文字列をいくつでも持つことができる。

int LINK(int g1,int g2);

ID が g1 であるグループに、ID が g2 である子グループを結びつける。一つのグループは子グループをいくつでも持つことができる。また、一つのグループはいくつでも親グループを持つことができる。但し、あるグループが自分自身の子グループとなることができない。更に、いくつかの中間段階を経て親子関係が循環的に連鎖してはならない。

LINK_XFORM(int l, 適用順序, 指定方法, 引数群);

ID が l の親子関係のリンクに、相対的な位置情報を指定する。

位置情報は、内部的にはリンク・マトリクス（４×４の同次マトリクス）として管理されている。これに対して、LINK_XFORM は、位置情報を累加的に定義し、内部処理としては既存のリンク・マトリクスに対する乗算が実行される。このリンク・マトリクスは、LINK コマンドが実行された時点では、単位マトリクス（対角成分の 1.0、残りは 0.0）に初期化されている。

・適用順序は、LOAD, PRE, POST（予約語）のいずれか

LOAD は、既存のマトリクスを置き換える

PRE は、既存のマトリクスに前から掛ける。

POST は、既存のマトリクスに後ろから掛ける。

（例えば、太陽に対する地球の位置がリンク・マトリクスで定義されていたとする。これに対する回転を LINK_XFORM で追加指定する場合、LOAD であれば位置関係が解消して太陽と地球は重なる。PRE であれば、地球が公転する。POST であれば、地球が自転する。）
・指定方法は、IDENTITY, TRANSLATE, ROTATE_X, ROTATE_Y, ROTATE_Z, ROTATE_A,SCALE, MATRIX のいずれか（予約語）である。
IDENTITY は、単位行列（変位なし）を示す。適用順序が LOAD であれば初期化する。それ以外であれば変化はない。
TRANSLATE は、並進を示す。引数は浮動小数３。
ROTATE_X, ROTATE_Y, ROTATE_Z は、座標軸の周りの回転を示す。引数は回転角を上から見た反時計回りに度で、浮動小数一つにより指定する（左螺子)。ロール・ピッチ・ヨーで回転を表現するような場合には、ロール（LOAD, ROTATE_Ｙ)、ピッチ（LEFT, ROTATE_Ｘ)、ヨー（LOAD, ROTATE_Ｚ）の順に３回に分けて順次適用する。建築物を敷地に配置するような場合には、ROTATE_Z（水平回転）だけで十分である場合が多い。
ROTATE_A は、任意の回転軸周りの回転を示す。回転軸を示すベクトル３値と、回転角を浮動小数の合計４値を引数とする。
SCALE は、３軸に関する拡大縮小を示す３の浮動小数を引数とする。
MATRIX は、リンク・マトリクスの全ての値を直接指定する１６の浮動小数を引数とする。

（２）携帯端末を用いた現場活動記録のためのライブラリ関数

保存データの解読に使用することはない。

携帯端末で閲覧を行う際に、シャッターボタンが操作されると、その時点で背面カメラが取得して合成表示に使用していた画像、表示されていた三次元モデル、センサーが取得した視点位置、カメラアングル（端末の姿勢)、時刻等の情報をメモリ上に記録する。この記録は、アプリ終了時点で、記録ファイル mobile.ja.scn として保存される。このファイルは、次のアプリ起動時点で読み込まれる。この記録ファイル mobile.ja.scn の読み込みのために使用されるライブラリ関数群である。

記録ファイルが読み込まれた後に、シャッターが操作された場合には追記されていく。

記録再生の機能が選択されると、このファイルに基づいて背景画像の縮小画像がマトリクスで表示され、ユーザーがその中から、あるシーンを選択して削除したり再生表示したりすることができる。

この記録には、景観シミュレータにおけるシーンファイルに準拠した外部ファイル形式であり、読み込まれた後のメモリ上でのデータの管理にも、景観シミュレータと同様に sml ライブラリが用いられている。

VC-3M は、mobile.ja.scn ファイルを、メタファイルの一種としてコンパイルし、実行する。それぞれのコマンドに対応する景観シミュレータの sml ライブラリ関数を呼び出す実行処理の結果、メモリ上に過去のシャッター操作の記録が格納される。終了時点では、景

観シミュレータの ip ライブラリを用いて、シーンファイルとしての保存を行う。

このファイルが実際に生成した mobile.ja.scn の例をリスト 3-1 に示す。一つのシャッター操作が、一つの SCENE コマンドに対応している。このファイルの例では、４回分のシャッター操作の結果を記録している。

リスト 4-2-1 記録ファイル mobile.ja.scn の例

```
# mobile.ja.scn
# 4 scenes
	CAM000 = CAMERA(4.59636449813843, 7.70967817306519, 4.25015115737915
		,4.48174047470093, 6.71801471710205, 4.19128608703613
		,-0.00861490704119205, -0.0582613088190556, 0.998264610767365
		,35.8967247009277, 1.77777802944183, 0.100000023841858, 317.575653076172);
	E000 =
EFFECT(IKE,"0.060198","9.449400","0.000000","/storage/emulated/0/VirtualConverter/meta/LSSG.cmm?/storage/emulated/0/VirtualConverter/data/hamakaze2.geo");
	EG000 = EFFECTGROUP(E000);
	M000 =
MODEL("/storage/emulated/0/VirtualConverter/meta/LSSG.cmm?/storage/emulated/0/VirtualConverter/data/hamakaze2.geo");
	I000 = IMAGE("BI20141008.102441.jpg");
	S000 = SCENE(1, I000, , M000, , EG000, CAM000, );
	CAM001 = CAMERA(-70.7441101074219, -20.876443862915, 40.0000038146973
		,-69.801025390625, -21.1194190979004, 39.7729606628418
		,0.209715604782104, -0.09530408680439, 0.973107099533081
		,35.8967247009277, 1.77777802944183, 0.100000023841858, 375.665069580078);
	I001 = IMAGE("BI20141008.103332.jpg");
	S001 = SCENE(1, I001, , M000, , EG000, CAM001, );
	CAM002 = CAMERA(-71.5213470458984, -21.5806560516357, 39.9000015258789
		,-70.6920623779297, -22.0271797180176, 39.5639953613281
		,0.289003819227219, -0.171933308243752, 0.941762506961823
		,35.8967247009277, 1.77777802944183, 0.100000023841858, 373.618560791016);
	E001 =
EFFECT(IKE,"0.060198","9.449400","0.000000","/storage/emulated/0/VirtualConverter/meta/LSSG.cmm?/storage/emulated/0/VirtualConverter/data/hamakaze6t.geo");
	EG001 = EFFECTGROUP(E001);
	M001 =
MODEL("/storage/emulated/0/VirtualConverter/meta/LSSG.cmm?/storage/emulated/0/VirtualConverter/data/hamakaze6t.geo");
	I002 = IMAGE("BI20141008.103506.jpg");
	S002 = SCENE(1, I002, , M001, , EG001, CAM002, );
	CAM003 = CAMERA(-93.8856811523438, 9.42799949645996, 42.5999984741211
		,-93.2221374511719, 8.74769973754883, 42.2887077331543
		,0.2197475284338, -0.220506623387337, 0.950309813022614
		,35.8967247009277, 1.77777802944183, 0.100000023841858, 398.871368408203);
	I003 = IMAGE("BI20141008.103825.jpg");
	S003 = SCENE(1, I003, , M001, , EG001, CAM003, );
# i3_outputClose start
# i3_outputClose end
```

int TIME(float 時刻);

時刻を記録する。VC-3M の処理系においては、背景画像のファイル名を時間の記録として使用しているため、mobile.ja.scn には使用していない。

int CAMERA(float 視点 x,y,z, 注視点 x,y,z, 上方ベクトル x,y,z,
float 視野角, float 縦横比, float 最近距離, 最遠距離);

視点情報を定義する。

視点、注視点、上方ベクトルは３組の座標値で表す。

視野角は焦点距離の逆数にあたる数値で、画像中心から右端までの角度を度で表す。

縦横比は、画像の横幅に対する縦の比率、

最近距離、最遠距離は、対象物の存在範囲。

視野角の範囲と、距離が台形の立体的な範囲を定義し、その範囲内にあるオブジェクトだけが表示対象となる。

int LIGHT(int タイプ, float 位置 x,y,z,w, float 色彩 r,g,b);

光源を定義する。

位置は座標値 x,y,z

w は点光源 0 か平行光源 1 かを示す。

色彩の r,g,b 値は、0.0~1.0 の浮動小数値

int LIGHTGROUP(int 光源１, 光源２, ・・・);

定義済の光源の ID を組み合わせて光源グループを定義する。

OpenGL の制約条件として光源は最大 8 であるが、ファイル構成上は無制限である。

int MODEL(“ファイル名”);

三次元モデルを定義する。本処理系においては、

メタファイル名？データファイル名

という形式で、二つのファイル名を「？」で繋いで表現している。

IMAGE(“背景画像ファイル名”);

モデルと合成表示する背景画像、前景画像を定義する。本処理系においては、携帯端末の背面カメラにより取得され、シャッターボタンが押された時点でファイル保存された画像ファイル名を記録する。

EFFECT(効果 ID, 引数群・・・);

本処理系(VC-3M)においては、IKE という名称の効果だけを実装している。このコマンドは、シャッター操作が行われた時点でのキャリブレーション値等を記録している。

EFFECTGROUP(int 効果 1,・・・);

様々な EFFECT の定義(数は無制限)を束ねて一つの SCENE に登録するためのコマンドであるが、本処理系(VC-3M)においては、IKE という名称の効果だけを実装していないので、形式的な使用となっている。

SCENE(int タイプ, int 背景画像 ID, int 前景画像 ID, int モデル ID, int 時刻 ID, int 効果グループ ID, int 視点 ID, int 光源グループ ID);

タイプは、本処理系では 1 に固定である。

背景画像 ID は､背面カメラが撮影した画像をシャッター操作時点で JPEG 形式に保存したファイルの名称であり、撮影時刻を反映している。

前景画像は、本処理系では使用していない。

モデル ID は、メタファイルとデータファイルの組を指定している。

時刻は使用していない。

効果は、撮影段階での緯度経度のキャリブレーションを記録している。

視点は、シャッター操作時点のカメラの位置と姿勢等を記録している。

光源グループは、本処理系では使用していない。

引数の総数は一定数（９）であるが、使用しない項目はセパレータ「,」の間が空欄であって構わない。

（３）座標系に関するライブラリ関数

二次元のベクトルデータを扱う GIS ソフトには定番の機能である。

```
quat IKE2DES(quat ike);
quat DES2IKE(quat des);
quat IKE2NAT(int k, quat ike);
quat NAT2IKE(int k, quat nat);
quat IKE2WLD(int k, quat wld);
quat WLD2IKE(int k, quat ike);
```

int k は、日本周辺を平面直角座標系に分割したときの ID 番号である。

ike は経度が x 成分、緯度が y 成分である。度以下の分・秒は度の小数点以下として表現。

nat,wld は、東が x 成分、北が y 成分とし、m単位で記述する。

des は、経度０の赤道上を X 軸、東経９０度の赤道上を Y 軸、北極を Z 軸とする。

なお、平面直角座標系は、回転楕円面を接平面に投影したものであるため、ユークリッド座標系とは異なる。隣接する区域との境界は不連続である。

また、2001 年を境に、緯度経度も変更されている。旧版の国家座標系から世界測地系に移行するためには、旧版国家座標系→旧版緯度経度→新版緯度経度→新版世界座標系の経路で変換を行う必要がある。

（４）解読・入力済データへのアクセスと利活用のためのライブラリ関数

保存データの解読に使用することはない。解読が終了し、データベースやメモリ上に展開・格納された保存データにアクセスし利活用するためのライブラリ関数群である。

VC-2D、VC-4D において、既に読み込まれているメタファイル（形式定義ファイル）とデータファイルの内容にアクセスし、座標値や図形や色彩等に関する情報を取り出すためのコマンド群であり、別形式のファイルとして出力するためのメタファイル（形式定義ファイル）を記述するために必要となる。利活用の一形態として、保存ファイルを一度データベースに読み込んだ後、別のメタファイルで定義した別の任意形式のファイルに出力する方法を例示したものである。

```
int G();
```

読み込まれている全てのグループに順に選択する。全てのグループへのアクセスが終了したら、ゼロを返し、それ以外の場合には残りのグループ数を返す。

int Gid();

現在選択されているグループの ID を返す。

int Gattribute();

現在選択されているグループの属性 ID を取得する。

int Gmaterial();

現在選択されているグループのマテリアル ID を取得する。

int Gtexture();

現在選択されているグループのテクスチャ ID を取得する。

int F();

G()ループの中で現在選択されている実体グループ、または D()ループの中で現在選択されている表示グループに所属する面および線を順次選択し、残りの面の数を返す。

上位ループによりグループが選択されていない場合には、存在する全ての面を対象として順次アクセスする。

int Fshp();

選択されている面の種類を返す。

０：凸多角形　１：線　２：凹多角形　３：異常な面

int Fnormal();

選択されている面の法線を選択する。法線が設定されていない場合にはゼロを返す。

法線の各成分を取得するためには、Nx(), Ny(), Nz()関数を使用する。

int Fcolor();

選択されている面の色彩を選択する。色彩が設定されていない場合にはゼロを返す。

色彩の各成分を取得するためには、Cr(), Cg(), Cb(),Ca 関数を使用する。

int Fmaterial();

選択されている面のマテリアル ID を返す。マテリアルが設定されていない場合にはゼロを返す。

int Ftexture();

選択されている面のテクスチャ ID を返す。テクスチャが設定されていない場合にはゼロを返す。

int F3();

選択されている面を三角形に分割し、順次アクセスする。残りの三角形の数を返す。

選択されている三角形の要素へのアクセスは、面の要素へのアクセスと同様である。

int V();

選択されている面の頂点に順次アクセスし、残りの頂点の数を返す。

int Vcoord();

選択されている頂点の座標 ID を返す(sql)。

選択されている頂点の各座標値を取得するためには、Sx(),Sy(),Sz()関数を使用する。

int Vcolor();

選択されている頂点の色彩 ID を選択する。

選択されている頂点の各色彩値を取得するためには、Cr(),Cg(),Cb(),Ca()関数を使用する。

int Vnormal();

選択されている頂点の法線 ID を選択する。

選択されている法線の各軸の値を取得するためには、Nx(),Ny(),Nz()関数を使用する。

int Vtcoord();

選択されている頂点のテクスチャ座標 ID を選択する。

選択されているテクスチャ座標の値を取得するためには、Tx(),Ty()関数を使用する。

int Vvirtual();

選択されている頂点の仮想線フラグを取得する。このフラグが 1 であるとき、この頂点と次の頂点を結ぶ辺は、外周から内部の島を繋ぐための仮想線であり、ワイヤーフレーム表示モードにおいては線を表示しない。面の表示だけを行う場合や、面の面積を計測する場合などにおいては、無視しても構わない。

int S();

グループや面が選択されていない状態において、全ての頂点座標に順次アクセスする。残りの頂点座標の数を返す。

float Sx();

S()関数によるループの場合、全頂点リストから選択されている頂点座標の x 座標値を返す。F()関数と V()関数によるループの場合、F()で選択されている面の中で V()で選択されている頂点の頂点座標の x 座標値を返す。

float Sy();

Sx と同じ呼び出し条件下で、選択されている頂点座標の y 座標値を返す。

float Sz();

Sx と同じ呼び出し条件下で、選択されている頂点座標の z 座標値を返す。

quat Sq();

選択されている頂点座標の 3 座標値を四元数の形で返す。 t 成分はゼロである。

int N();

グループや面が選択されていない状態において、全ての法線ベクトルに順次アクセスする。残りの法線ベクトルの数を返す。水平垂直の要素だけからなる建築物の場合、非常に複雑な形状であっても、法線ベクトルは 6 種類しかない、という場合もありうる。

float Nx();

選択されている法線ベクトルの x 座標値を返す。

float Ny();

選択されている法線ベクトルの y 座標値を返す。

float Nz();

選択されている法線ベクトルの z 座標値を返す。

quat Nq();

選択されている法線ベクトルの３座標値を四元数の形で返す。 t 成分はゼロである。

int C();

グループや面が選択されていない状態において、全ての色彩に順次アクセスする。残りの色彩の数を返す。

float Cr();

選択されている色彩の R 値（赤）を返す。

float Cg();

選択されている色彩の G 値（緑）を返す。

float Cb();

選択されている色彩の B 値（青）を返す。

float Ca();

選択されている色彩のα値（不透過性）を返す。

quat Cq();

選択されている色彩の４構成値を四元数の形で返す。

αが t 、R が x 、G が y 、B が z 成分に代入される。

int T();

グループや面が選択されていない状態において、全てのテクスチャ座標に順次アクセスする。残りのテクスチャの数を返す。全てのテクスチャ付きの面が長方形で、それぞれの画像について全体をマッピングされている場合、テクスチャ座標は、(0,0), (1,0), (1,1), (0,1) の４種類しかない場合もありうる。

float Tx();

選択されているテクスチャ座標の x 座標値（横方向）を返す。

float Ty();

選択されているテクスチャ座標の y 座標値（縦方向）を返す。

quat Tq();

選択されているテクスチャ座標の２座標値を四元数の形で返す。 t 成分,z 成分はゼロである。

int L();

全てのリンクに順次アクセスし、残りのリンク数を返す。

int Lp();

選択されているリンクの親グループの ID を返す。

int Lc();

選択されているリンクの子グループの ID を返す。

float Lm();

選択されているリンクに設定されているリンク・マトリクスの 16 値を順次読み出す。

int Li();

選択されているリンクに設定されているリンク・マトリクスが単位行列なら 1 を返す。

quat Lt();

選択されているリンクのリンク・マトリクスの並進成分を返す。

quat Lr(); Lr(PRE); Lr(POST);

選択されているリンクのマトリクスの回転成分を四元数形式で返す。

リンク･マトリクスを二つの回転とスケールに分解した時､PRE は左側の回転行列､POST は右側の回転行列を返し、引数がない場合には、この二つの回転を掛けた値を返す。

Lr の取得と Ls の取得は一体の処理であり、同一のマトリクスを分解して得る。

スケールで表現される変形が存在する場合、Lr()は、近似的な意味しかない。

quat Ls();

選択されているリンクのリンク・マトリクスのスケール成分を x,y,z 値で返す。 t 成分は体積膨張率である。

リンク・マトリクスを並進、スケール、および二つの回転に分解する計算方法とその幾何学な意味については、４－４で解説する。

int D();

全ての表示グループを順次選択し、残りの数を返す。例えば、１００の部品（子グループ）から成る住宅が 4 棟配置されているとする。この時、グループの総数は、

１（ルート）＋４（住宅）＋１００（部品）＝１０５である。

リンクの総数は、

４（ルートと住宅の間）＋１００（住宅と部品の間）＝１０４である。

一方、表示グループの総数は、

１（ルート）＋４（住宅）＋４×１００（部品）＝４０５である。

この方法でアクセスしたグループにおいては、上方のリンク・マトリクスは全て計算され、ルートの座標系における座標値が取得される。

＊

メタファイルが解読した保存データが、シーングラフ系か、単純ソリッド系か、原始的図形系か等によってループの組み方が変化する。この系統の違いは、保存データのファイル形式よりはむしろ、データ構築段階でのモデリングの方法に依存する。複雑なシーングラフが記述できるデータ形式を用いて、原始的な構造（例えばグローバル座標で記述された三角形による表面タイルの集合体）も記述することができる。

①LSSG 形式のファイル出力

景観シミュレータにおいては、ip ライブラリの ipoutput 関数で出力している処理を、VC-2V 上で動作するメタファイル outlssg4.cmm により実行する(リスト 4-2-2)。このメタファイルにおいては、最初に法線ベクトルと色彩を全て出力した上で、個々のグループ→

面→頂点の順に処理する。

リスト 4-2-2 LSS-G 形式のファイル出力を行うメタファイル例(シーングラフ型)

```
# outlssg4.cmm (151002)
# 151002 VC-2Vにおいて GSI動作確認
# 140531 VC-2Vにおいて動作確認
# 140628 G終了条件の修正 (ゼロを返す)
int main( ){
          int inputlen;
          int i,j,k,nv;
          int iN,iC,iT,iV;
          inputlen = LEN();
          if(0<inputlen) exit(-1); //入力ファイルが指定されているエラー
          else printf("#入力ファイルは指定されていないことを確認¥n");
          printf("CONCAVE(ON);¥n"); //とりあえず、全ての場合
          while(i=N()){
                    printf("N%d=NORMAL(",i);
                    printf("%f,",Nx());
                    printf("%f,",Ny());
                    printf("%f);¥n",Nz());
          }
          while(i=C()){
                    printf("C%d=COLOR(",i);
                    printf("%f,",Cr());
                    printf("%f,",Cg());
                    printf("%f,",Cb());
                    printf("%f);¥n",Ca());
          }
          while(G()){
                    i = Gid();
                    printf("G%d=GROUP();¥n",i);
                    while(j=F()){
                              for(nv=k=V();k;k=V()){
                                        printf("P%d=COORD(",Vcoord());
                                        printf("%f,",Sx());
                                        printf("%f,",Sy());
                                        printf("%f);¥n",Sz());
                              }
                              for(nv=k=V();k;k=V()){
                                        iN = Vnormal();
                                        iC = Vcolor();
                                        iT = Vtcoord();
                                        iV = Vvirtual();
                                        printf("V%d=VERTEX(",nv-k+1);
                                        printf("P%d",Vcoord());
                                        if(iN+iC+iT+iV == 0){
                                                  printf(");¥n");
                                                  continue;
                                        }else printf(",");

                                        if(iN){
                                                  printf("N%d",iN);
                                        }
                                        if(iT+iC+iV== 0){
                                                  printf(");¥n");
                                                  continue;
                                        }else printf(",");

                                        if(iT) printf("T%d",iT);
                                        if(iC+iV == 0){
                                                  printf(");¥n");
```

```
                                        continue;
                                }else{
                                        printf(",");
                                }
                                if(iC) printf("C%d",iC);
                                if(iV == 0){
                                        printf(");¥n");
                                        continue;
                                }else{
                                        printf(",P%d);¥n",iV);
                                }
                        }
                        if(!nv){
                                printf("#no face vertex¥n");
                                continue;
                        }
                        printf("F%d=FACE(V1",j);
                        for(k=1;k<nv;k++){
                                printf(",V%d",k+1);
                        }
                        printf(");¥n");
                        if(Fnormal()){
                                printf("FACE_NORMAL(F%d,",j);
                                printf("N%d);¥n",Fnormal());
                        }
                        if(Fcolor()){
                                printf("FACE_COLOR(F%d,",j);
                                printf("C%d);¥n",Fcolor());
                        }
                        switch(Fshp()){
                        case 0: //face
                        default:
                                printf("GROUP_FACE(G%d",i);
                                printf(",F%d);¥n",j);
                                break;
                        case 1: //line
                                printf("GROUP_LINE(G%d",i);
                                printf(",F%d);¥n",j);
                        }
                        //printf("F%d",j);
                        //printf("%c;¥n",')');
                }
        }
        while(i=L()){
                printf("L%d=LINK(",i);
                printf("G%d,",Lp());
                printf("G%d);¥n",Lc());
                printf("LINK_XFORM(L%d,LOAD,MATRIX",i);
                for(j=0;j<16;j++){
                        printf(",%f",Lm());
                }
                printf(");¥n");
        }
        printf("CONCAVE(OFF);¥n");
}
```

立体オブジェクト並置型のデータを三角形の集合体として LSS－G 形式で出力する例をリスト 4-2-3 に示す。

リスト 4-2-3 LSS-G 形式のファイル出力を行うメタファイル例(三角形分割)

```
# F3.cmm (130110)
```

```
int main( ){
          int i,j,k,iv;
          int iS,iN,iC,iT,g;
          quat q;
          float f;
          logf("EXT=geo¥n");
#         g = GROUP(att'ribute);
#         GROUP_ATTRIBUTE(g,att'ribute);
          printf("#全立体の出力¥n");
          while(0<=(i=G())){//全ての立体を順次選択するループ
                    printf("G%d=GROUP();¥n",i);
                    while(j=F()){//選択された立体に属する全ての面を順次選択するループ
                      printf("#F[%d]¥n",j);
                      while(k=Ft()){//選択された面を三角形に分割し全ての断片を順次選択するループ
                              printf("#F3[%d]¥n",k);
                              while(iv=V()){//選択された三角形に属する全ての頂点を順次選択するループ
                                        printf("#V[%d]¥n",iv);
                                        Vcoord();
                                        q = Sq();
                                        printf("%q¥n",q);
                              }
                      }
                    }
          }
}
```

②VRML 形式のファイル出力

シーングラフ型のデータの VRML 形式によるファイル出力例を、リスト 4-2-3 に示す。

リスト 4-2-4 VRML 形式のファイル出力を行うメタファイル例

```
# VRMLOUT.cmm

int main(){
  int level,ig,virgin;
  level = 0;
  virgin = 1;
  printf("#VRML V2.0 utf8¥n");
  logf("#Ovrml.cmm¥n");
  while(ig=D()){
            virgin = 0;
      logf("#G[%d],",ig);
      logf("level=%d,",Dlevel());
      logf("first=%d¥n",Dfirst());
    if(Dlevel()==0){//ルートグループ
      if(level==0){
        printf("Group{#%d¥n",ig);
        printf(" children[¥n");
        level = 1;
      }else{
        for(;Dlevel()<level;level--) printf(" ]次Group検出、levelを戻す%d¥n",level);
        printf("}¥n");
        printf("Group{#%d¥n",ig);
        printf(" children[¥n");
        level = 1;
      }
    }else{ //子グループ
      if(level < Dlevel()){//階層が深くなる
              printf(" child長(%d,",ig);
        printf("level(%d)¥n",Dlevel());
```

```
        level = Dlevel();
      }else if(level == Dlevel()){//同じ深さ
              printf(" child次(%d,",ig);
        printf("level(%d)¥n",Dlevel());
      }else{//階層が浅くなる
          for(;Dlevel()<level;level--){
            printf(" ](Dループ内)¥n");
          }
      }
    }
  }
  printf("#Dループ終了¥n");
  if(0<level){
    printf(" ");
    for(;0<level;level--){
              printf("]");
    }
    printf("#終了時¥n");
  }
  if(virgin == 0) printf("}終了時¥n"); //Group
}
```

③DXF 形式のファイル出力

[例２]シーングラフ型のデータを単純な面群として DXF 形式で出力する

同じシンボルが繰り返し、異なる場所に配置されているような場合であっても表示される全ての面にアクセスする必要がある。このような場合、上記の D 関数を用いる。

リスト 4-2-5 DXF 形式のファイル出力例

```
# DXFOUT.cmm

void OutputPolyline(){
  int iS;
  while(F()){//表示グループの全ての面を順次選択するループ
    printf("  0¥nPOLYLINE¥n");
    printf(" 66¥n1¥n");
    if(Fshp()==0)  printf(" 70¥n1¥n");//凸ポリゴン
    else printf(" 70¥n9¥n");//凹ポリゴン
    while(V()){//面の全ての頂点を順次選択するループ
      printf("  0¥nVERTEX¥n");
      iS = Vcoord();//頂点の座標を選択
      printf(" 10¥n%f¥n",Sx());
      printf(" 20¥n%f¥n",Sy());
      printf(" 30¥n%f¥n",Sz());
    }
    printf("  0¥nSEQEND¥n");
  }
}

int main(){
          int i;
          logf("EXT=dxf¥n");
          printf( "  0¥nSECTION¥n");
          printf( "  2¥nHEADER¥n");
          printf( "  0¥nENDSEC¥n");

          printf( "  0¥nSECTION¥n");
          printf( "  2¥nENTITIES¥n");
          while(D()){ //全ての表示グループを順次選択し、DXF-POLYLINE形式で出力するループ
             OutputPolyline();
```

```
            }
            printf( ″  0¥nENDSEC¥n″);
            printf( ″  0¥nEOF¥n″);
            return 1;
}
```

④STL 形式のファイル出力

シーングラフ型のデータを、三角形分割した STL 形式で出力する例をリスト 4-2-6 に示す。STL 形式は３Ｄプリンターの入力形式として広く使われており、ローコストで模型を作成することができる。但し、図形的に閉じていなければならない（ソリッド・モデル)。

シーングラフ型のデータでは一般に、同じモデルが様々な位置に、様々な角度で、様々な縮尺で複数個配置されている。このような場合、D()ライブラリ関数を用いて、ディスプレイリストとしてソリッドを取得すると、実際に画面で立体表示される見かけ上のオブジェクトを全て取得することができる。次に、一つのソリッド（ここでは Polyline と呼ぶ）を構成する各面を F()ライブラリ関数を用いて取り出す。さらに、これを三角形に分割するために、選択されている面を対象として、Ft()ライブラリ関数を用いて、三角形として取得し、その各頂点座標を STL 形式に従って OutputTriangle 関数の中でファイル出力する。

リスト 4-2-6 STL 形式のファイル出力を行うメタファイル例（三角形分割）

```
/*stlout.cmm*/

void OutputTriangle(){
   int iS;
   while(Ft()){
      printf("facet ");
      printf("normal ");
      printf("%f ",Nx());
      printf("%f ",Ny());
      printf("%f¥n", Nz());
      printf("outer loop¥n");
      while(V()){
         iS = Vcoord();
         printf("vertex %f ", Sx());
         printf(" %f",Sy());
         printf(" %f¥n",Sz());
      }
      printf("endloop¥n");
      printf("endfacet¥n");
   }
}

void OutputPolyline(){//多角形の一つを処理
      while(F()){
                  OutputTriangle();
            }
}

int main(){
            int i;
            quat q;
            printf( "solid¥n");
            while(D()){
                OutputPolyline();
            }
            printf( "endsolid¥n");
}
```

４－３．三次元データ形式とメタファイル作成例

（１）本研究以前の状況

本処理系の開発に至った背景には、景観シミュレーション等における様々な三次元データ形式への、主にコンバータによる対応の変遷や、ほぼ同時代的に進行してきた、トップダウン的なデータ形式の標準化に向けた様々なアプローチの失敗の歴史がある。

このことの根本的な原因は、社会的インフラとも言える計算環境の急速な発展、とりわけマイクロチップや記憶媒体に関する技術の発達と短命な各種製品の変遷が、主に同時代的な通信速度の向上に資する方向でビジネス化される一方、時代間の継承という点にあまり配慮されてこなかなった点である。このため、陳腐化した形式で作成され保存されているデータの可読性は低い状況になった。例えば、1980 年代に旧建設省建築研究所が Intergraph 社の Microstation という CAD ソフトで作成し、８インチフロッピーディスクや､128MB の MO に保存したデータは､そのままの形で利用することはかなり困難である。

①貿易コンバータにより対応した各種データ形式

1993 頃には、景観シミュレーションに読み込んで表示・閲覧可能な LSS-G 形式データを作成するために、各種形式を対象とする個別のファイルコンバータを作成した。これらの独立した実行形式は、MS-DOS のコマンドラインで利用できる実行形式（現在では、コンソール・アプリと呼ばれる）として作成された。これを Windows アプリケーションから選択して起動できるラウンチャとして、「貿易コンバータ」を作成した。これは、処理対象とするデータ形式（入力・出力）と、各種のコンバータを起動するためのパラメータ入力画面群から構成されていた（図 4-3-1）。

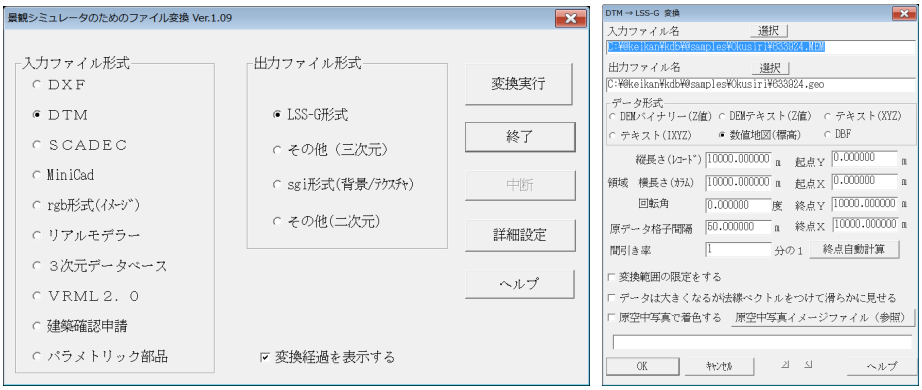


図 4-3-1 貿易コンバータ（メイン画面と DEM 変換の詳細設定画面）

当時、以下のようなデータを作成した。

1）建築用 CAD データからのインポート

福岡県住宅供給公社が峰花台団地を中層から高層集合住宅に建替えたの際の設計データは、上記の CAD ソフト MicroStation で入力後、コンバータを作成して LSS-G 形式に変換し、OpenGL を搭載した WindowsNT マシン(NT3.51)上で動作する景観シミュレータ 2.01~2.05 を用いた地元での検討に初めて実運用された(1995.5.13 旧建設省建築研究所)。

次いで Macintosh 上で動作する MiniCAD も橋梁や住宅等のデータの図面からの入力に

使用された。作業は非常勤職員により行われ、作成したデータは、後に景観データベースに追加された。このためにプロッタを駆動するためのコマンドのような MiniCAD 形式のデータを三次元データに変換するためのコンバータ MC2LSS を作成した。これは貿易コンバータにおける入力データ形式の一つとして実装されている。MiniCAD というソフトウェアは現在使用することができないが、VectorWorks というソフトウェアに継承されている。

当時から、CAD データ交換のためのデファクト形式として使用されていた DXF テキスト形式を変換するためのコンバータを作成し、東京神田の市街地データや、ダムの計画データの LSS-G 形式への変換と表示に用いた。このコンバータは後に、2006 年頃に、インドネシアの住宅地計画案として現地の設計事務所が作成した計画案の入力にも使用した。

旧建設省土木研究所は SiliconGraphics 社製の、グラフィックワークステーション Indigo2 上で動作する Attract アプリケーション（国産）を用いて景観データベースの基本部分を作成した。これには様々な樹木、自動車や信号機などの点景、代表的な建築物サンプルなどが含まれている。また同様の方法を用いて、下記の釜石～宮古間の高規格道路の内の釜石ジャンクション付近の景観、および福島西道路周辺の景観検討のためのデータが作成され、景観シミュレータ 2.01~2.05 を用いて地元での検討に初めて使用された。

図 4-3-2 釜石ジャンクション景観検討データ

なお当時検討されていた釜石ジャンクションのデータ作成区域は、やや内陸にあって 2011 年の東日本大震災に際して津波の被害は受けなかった。2012 年 2 月時点での現況写真撮影を行った。復興庁の釜石支所はこの区域の中に設置された。

福島西道路に関しては、2012 年 3 月時点で現況確認を行い、既に完成していた橋梁等の記録写真を、建設前の検討案と比較可能な視点から撮影した。

2) 写真からのモデリング

デジタルカメラを用いて撮影した現場写真からの町並データ作成は、同上の WindowsNT マシン上で動作する、写真からの三次元モデリングソフト「RealModeler」(富士通静岡エンジニアリング社製)で作成し保存した SKV 形式のデータを、1995～6 年の官民共同研究により作成したコンバータ SKV2LSS を用いて LSS-G 形式に変換して使用した。幕張駅南口駅前再開発、福岡市内の土地区画整理事業の地元説明会のために使用された。後者に関し

ては、最終的に土地区画整理事業は成立しなかったものの、主目的であった街路拡幅事業は実現した。この街路拡幅に伴い、移築（曳家）が行われた場合に形成される町並みを、デジタル写真から作成した沿道の建物を拡幅後の沿道に配置する方法で作成した。

国総研では、本研究の一環として 2012 年度に、同上のシステムを Windows 7 上に再度セットアップし動作を確認した上で、これを用いて、1995 年までに撮影された奥尻島青苗地区等の古写真を解析し、保存データを例示的に作成した。更に、この保存データを用いて、本処理系をセットアップした携帯端末で表示を行う体験教室を、2014 年 10 月 8 日に奥尻島の岬公園で実施した（第 2 章参照）。

なお、同上のシステム(RealModeler および SKV2LSS)は残念ながら、Windows8 以降の 64 ビット Windows マシンにはセットアップすることができない。InstallShield を用いて作成したインストーラの互換性の問題に起因すると考えられる。このインストーラは、レジストリの設定を行っているため、必要な実行形式をコピーして手動でレジストリ設定を行う作業は容易ではない。

3) 地形及び既存市街地の DEM

ステレオ空中写真から自動解析した結果を格納する DEM 形式を LSS-G 形式に変換するためのコンバータを作成した。この DEM 形式のデータは、1995 年頃に釜石～大槌を結ぶ高規格道路の景観検討のために作成された。当時、海岸に高規格道路を計画すると、長大な法面が形成されるため、海岸の景観を大きく変化させることが問題視されていた。そこで、橋梁やトンネル等の建設費はかかるものの、それに対する代替案として、内陸に道路を計画することが検討されていた。震災直前の 2011 年 1 月頃に、この区域は山田西道路として開通した。立地条件から被災せず、結果的に道路としての機能を果たした。

ステレオ空中写真から地形と、直方体近似した建物を半自動生成する手法を用いて作成したデータベース（3 DDB 形式）を変換するコンバータを作成し、幕張駅西口再開発等に使用された。同じ方法を用いて、2001 年度に１５のまちづくり現場に関するデータが作成され、再開発計画等のデータを構築するための下図となる現状データとして使用された。多くの場合、地形データと共に空中写真が入手可能である。この場合、地形に空中写真を貼り込むことにより、位置の特定がはるかに容易になり、また景観検討のための遠景としても利用可能となる。一般的には、画像ファイルをテクスチャとして扱い、地形を構成する個々の面の各頂点にメッシュ上の座標値を定義する方法が行われるが、相対的に画像が粗い場合には地形を構成する個々の面や頂点に色彩を定義し、データファイルをロードする段階では画像ファイルは使用しないような方法も、貿易コンバータでは選択可能とした（図 4-3-1 右）。

当時は DEM データという名称は一般的なもので､測量各社は異なる方法で納品を行っていた。その後 2004 年には、レーザースキャナを搭載した航空機による地形測量が行われるようになり、DEM データの標準化が進み、首都圏や大都市圏の電子地図標高として、有償で購入し利用することができるようになった。

更に 2005 年頃、防災や地球温暖化対策の目的のために、太平洋沿岸地域に関しては国土地理院による５m メッシュの詳細なデータが作成され、行政内部で配布された。2011 年の東日本大震災の直後には、被災した沿岸地域のレーザースキャナによる再測量が行われ、防災や復興計画策定のために利用可能となった。このデータは、基本的に数値地図標高と同じような形式である。

現在では、基盤地図情報として、登録したユーザーは無償で電子地図に含まれる地形データを WEB サイトからダウンロードし利用することができる。現在配布されている地形データには、５ｍメッシュと１０ｍメッシュの二種類があり、前者はレーザースキャナにより計測されたデータであり、後者は最新の 1:25,000 地形図の等高線から作成されたものである。いずれも、XML 形式をベースとしたテキスト形式で記録されており、従来の電子地図とは異なる形式となっている。

これらついては全て、直近の貿易コンバータ(2.09)によっても入力することができる。また読み込んだ地形データに対して 2011 年に開発した景観シミュレータのためのプラグインである「高台整地機能(flow.dll)」により平坦化や法面形成等の編集作業を行うことができる。

地形データの場合には、データ全体は広いエリアをカバーする巨大なものであるため、必要な区域を検索し切り出すような作業環境が便利である。

4)GIS 形式のデータ

旧建設省建築研究所、国総研においては、この他 GIS の標準交換形式として広く使われていた shape ファイルに関するコンバータを作成し、例えば、みちのく国営公園等の地形データに適用された。また、インドネシアの国土地理院(BAKOSURTANAL)が作成した地形の電子データも shape 形式で配布されており、変換を行った。なお、SHP という拡張子を有するデータには、二次元図形を表現したベクトルデータから、地形等のメッシュ等を表現したものまで様々なモデリング方法が包含されており、ヘッダファイルでフォーマットが記述されている。このような多彩なヴァリエーションを含むデータ形式に対しては、その全ての可能なフォーマットを包含する総合的で巨大なコンバータを用意するよりも、本処理系のアプローチのように、保存すべき特定の記録データを解読するために十分な形式定義を含むメタファイルを添付し、そのペアを処理する方法が有効である。

②景観シミュレータにおける外部関数により作成したコンバータ

景観シミュレータの外部関数は、当初は少ないパラメータを入力して直方体、角錐、円錐、球などの基礎的な原始図形を簡便に生成するためのツールとして導入された。

データ形式が明確で補足的なパラメータの入力（その試行錯誤）が不要なデータ形式に関しては、この外部関数の引数としてファイル名を指定する方法が有効であるため、可能な場合には外部関数としてコンバータを作成した。VRML 形式、建築確認申請形式、延焼シミュレーション形式、STL 形式、SCADEC 形式、LandXML 形式、IFC 形式、KML 形式、電子地図形式、ポイントクラウド形式どのコンバータをこの方法で作成・実装した。これらを表 4-3-1 に一覧する。

表 4-3-1：景観シミュレータのための外部関数として作成したコンバータ

表示名称	実行形式	ファイル形式	拡張子	最終更新
BS2LSS	bs2lss.exe	建築確認申請形式	110	2006 年 12 月 5 日
FIRE2LSS	fire2lss.exe	延焼シミュレータ形式	dat	2009 年 10 月 5 日
FONT	font.exe	TrueTypefont 形式	ttf	2009 年 10 月 7 日
KML	kml2lss.exe	KML 形式	kml,kmz	2011 年 8 月 20 日
GEOID	GEOID.exe	ジオイド ASCII	asc	2011 年 12 月 15 日
SCADEC	scadec.exe	電子納品形式	sxf	2012 年 4 月 20 日
VRML2LSS	vrml2lss.exe	VRML 形式	wrl	2013 年 1 月 31 日
LandXML	landxml.exe	LandXML 形式	xml	2014 年 4 月 23 日
IFC	IFC.exe	IFC 形式	ifc	2014 年 12 月 21 日
STL	stl2lss.exe	STL 形式	stl	2015 年 10 月 5 日

外部関数による方法は、様々なファイル形式のデータを変換する方法を独立した実行形式として作成し、システムの内部に取り込むための入力手段を、基幹部分を更新することなしに追加する方法としては有効であるが、システム内部のデータを外部ファイルとして出力する方法としては使用できない。その理由は、外部関数からシステムのメモリ上の既存データ（現在表示されている物体）にアクセスする方法が用意されていなかったからである。このため、システム本体のビルドの中に出力処理のソースコードを含めなければならず、新たなファイル形式のための出力処理を追加するためには、基幹部分の実行形式自体を更新する必要があった。ファイル保存におけるファイル選択ダイアログにおいて、拡張子を指定することにより保存するファイル形式を選択している。

③プラグイン DLL によるデータ形式変換

2011 年に公開した、景観シミュレータ 2.09 においては、ユーザーが作成したプラグイン DLL を自由に追加する方法を導入した。プラグインからは、実行されている全てのライブラリ関数にアクセスすることができ、これを通じてファイル出力することも可能となった。

これを用いて、「高台整地プラグイン(flow.dll)」を作成し、その中で、国土地理院が太平洋沿岸に関して作成したデータ等の入力・編集作業と共に、三次元プリンタ等に入力できる SLT 形式等で出力するメニューも用意した。

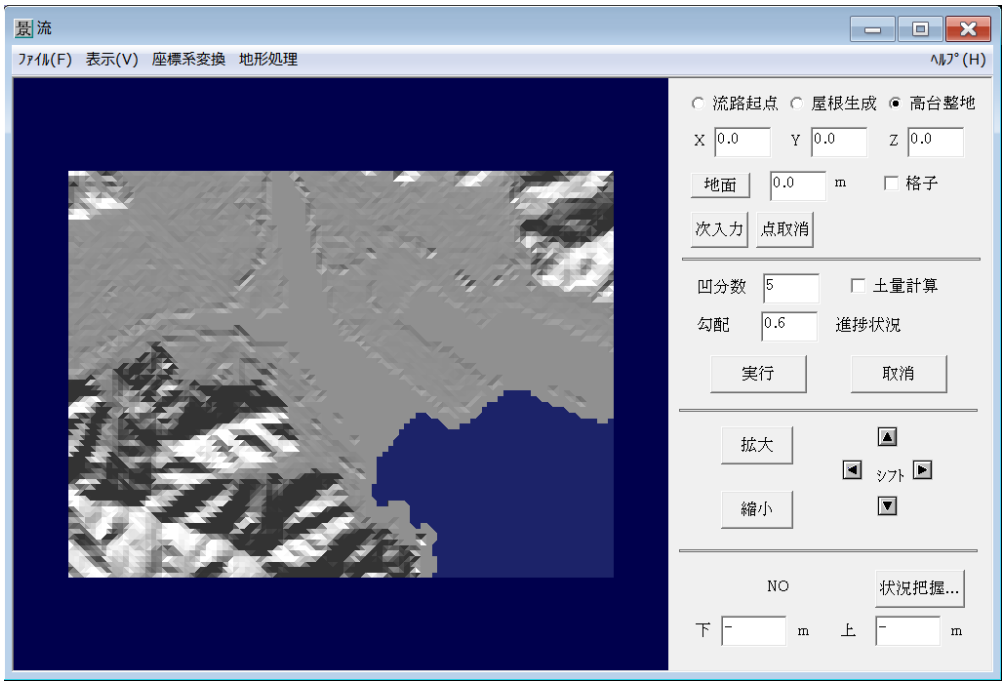


図 4-3-3 プラグイン flow.dll の操作画面

画面右のコントロール部の構成は、以下の通りである。

流路起点、屋根生成、高台整地
-機能をラジオボタンで３択
ＸＹＺ
-画面クリックした地点の座標を表示する
-数値を入力してフォーカスを外すことにより指定した地点の位置を変更する
地面　画面クリックした地点の高さを地面の高さに合わせるかどうかを選択する
　地面に合わせる場合、その右の高さm欄に入力された高さを地面の高さに追加する
格子　グリッドが表示されていれば格子点、表示されていなければ至近の点にスナップ
点取消　入力済の点列から最後の点を削除する
拡大　縮小　シフト　画面表示範囲の変更
状況把握　地形データが存在する高さの範囲を取得し、結果を表示する

プルダウン・メニューの構成は、以下の通りである。

[ファイル]
-[軌跡読込] 保存された整地エリア外形線ファイルの読込
-[軌跡保存] 画面入力した整地エリア外形線をファイル保存
-[上書保存] 画面入力した整地エリア外形線を同名ファイルに再保存
-[コンバータ]
--[ＳＴＬ出力(asc,bin)]ＳＴＬ形式で出力
--[数値地図] 数値地図の入力
-[ジオイド]Geoid ファイルの読み込み
-[LEM メッシュ] ＬＥＭ形式の地形データの読み込み
-[終了] プラグインの終了
[表示]
-[GRID]
-[メジャー] 縮尺の表示
-[上下範囲] 表示する標高の範囲
-[縦横範囲] 視点の移動速度を調整
-[表示モード]
--[テクスチャ表示]以下３択
--[シェーディング表示]
--[ワイヤーフレーム表示]以上３択
--[オプション設定]（特殊な表示モード）

---[地面のみ表示]（地面属性のある要素だけを表示）
---[地面テクスチャ表示]
--[オプション解除]
[座標系変換]
-[IKE2CAR] 緯度経度をＸＹ座標に変換する
-[DLL]その他の組み合わせ
--[FROM]変換元の座標系の選択
--[TO]変換先の座標系の選択
--[変更]座標系変更を実行
[地形処理]
-[ソリッド化] 地形等に側面と底面を付加して閉じた立体とする
-[複数モデル接合]隣接するＤＥＭ地形を接合する
--[選択されたグループをＧ１とする]
--[選択されたグループをＧ２とする]
--[接合実行]Ｇ１とＧ２の間の隙間を埋め、接合する
-[等高線からＴＩＮ作成] 等高線データから三角形で構成された地形を作成する
-[外形線の取得]地形の外周を折れ線として取得する
-[鳥籠]地球楕円体を表す緯度経度のワイヤーフレームを追加する
[ヘルプ]解説テキストを表示する。使用言語は、メイン画面で設定された言語である

（２）各種データ形式の調査・収集と仮想コンバータの開発要件

① 各種データ形式の調査

上記の経緯を踏まえ、本研究では初年度の 2010 年度に、これまで未対応の様々な三次元データ形式で広く使われているものに関する調査を行い、データの仕様書やサンプルデータの収集を行った。現在流通している三次元データ形式は既に 300 種類を超え、更に増え続けている。このような状況を見る限り、近い将来に統一的な形式が出現し、記録形式が統一されている見通しは、現時点では薄いことが判明した。

多様化の主な理由は、CAD-CAM 技術の生産合理化への活用が主に製造分野において進展し、様々な機械の制御等のために、三次元データに素材の力学特性、熱特性属性や、品質管理に関する情報がデータに不可され多用されていることに起因している。建築分野でも、BIM として三次元データに様々な建築生産に関係した情報を付加することが一般化している。特に、最近のデータ形式の多くは、XML 形式をベースとしてそれに固有のタグを追加したようなフォーマットになっていることが多く、自由に様々な属性情報を埋め込んでいくことが簡単にできる。そのようなデータは、独自の拡張子を用いて、特定の目的に使用されている。

三次元データには、モデリングの方法において、点群として表現する方法、断面図の集合として表現する方法、立体の表面を構成する面を記述する方法があり、更にパラメータを用いて関数的に表現する方法も可能である。

点群表現として従来は二次元的なメッシュの上に高さを与えて地形を表現する方法 DEM：Digital Elevetion Model が代表的であったが、最近ではレーザースキャナによる計測データ等で、xyz３座標のリストとして表現する方法(Point Cloud)も普及した。DEM は地形などの自由な形状をコンパクトに表現することができるが、一つの場所に定義される高さ値は一つに限られるため、垂直以上に切り立った崖や洞窟内部の形状を表現することはできない。

GIS(地理情報システム)において、平面的な座標（緯度・経度やXY座標）の上に属性（土地利用等）を記述する方法が広く行われている。この属性の一種として標高値が定義されれば立体的な形状が表現される。このことから、1990年代にはDEMのような形式で表現されたデータは2.5次元データと呼ばれることもあった。また逆に、三次元データに様々な属性（例えばタイムスタンプ）を付したデータが、N次元データと呼ばれることもあった。

断面図の集合として形状を表現する古典的な方法は、等高線による地形の表現である。地形図の等高線の形にコルク板を切り抜いて積み重ねることにより敷地を表現する方法は、1980 年代以前の建築模型では広く行われていた。最近では、CT スキャン(Computed Tomography)やNMR(Nuclear Magnetic Resonance, 核磁気共鳴)による生体計測で、立体を断面図（画像）の集合体として記録する方法が広く用いられている。

立体を、表面を構成する面の集合体として表現する方法（サーフェスモデル）は、CGの古典的な方法である。画面表示を行うだけであれば、立体が閉じている必要は必ずしもないが、体積を計算し、複数立体間の切り欠きなどの図形演算を行うためには、表面を構成する全ての面が隙間なく定義されている必要がある。サーフェスモデルの内、特に三角形だけを用いて表面を記述したデータを、TIN(Triangular Irregular Network)と呼ぶ。

本稿執筆時点で普及しつつある、３D プリンタ（データから立体形状を生成する）のための入力データとして、立体として閉じた面群を完備したソリッドモデル（例えば三角形に分割したSTL形式：後述）が広く用いられている。

ソリッドモデルと称するデータ形式には、このように単に表面が隙間なく閉じた面群として定義されているサーフェスモデルを指すだけの場合と、内部構造まで記述したデータを指す場合とがある。立体加工には前者で十分であるが、三次元メッシュを切って、それぞれのメッシュ（画像を構成するピクセルと対比して、ボクセルと呼ばれる）の属性（物質やその温度や圧力）を表現することにより、機械設計や気象予測などが行われている。

関数とパラメータ（引数）を用いて三次元形状を表現する方法は、歯車やカムなどの機械部品、ロボットの腕、サッシュやドア等の拘束された運動、更には構造フレームの変形等をコンパクトに記述し、力学計算を行う目的に適している。

このような立体の形状をモデリングする方法は、特定の名称と拡張子を有するあるデータ形式において、一つに限定される場合もあるが（例えば上記のSTL形式)、一つのデータフォーマットが、多様なモデリング方法を可能としている場合多い。従って、汎用的なコンバータを作成するためには、一つのデータフォーマットの多様な仕様に対応する必要がある。

一方、本処理系（仮想コンバータ）においては、保存データに添付するメタファイルは、保存データの内部で実際に用いられている範囲の記述方法（文法と単語）を定義できればメタファイルとしては十分である、という考え方に立っている。

② 仮想コンバータ処理系の作成と、可搬性の検証

本研究においては、様々な使用目的のために、様々な形式で作成されたデータを、いわ

ば「デジタル古文書」として扱い、長期保存のための方法を開発することを目的とした。その結果として、解読方法を記述したメタファイルを添付する方法が有効であると考える。この方法においては、保存すべきデータを従来のようにある形式（例えば現在普及している標準的な交換形式）に変換する必要がなく、原本となるオリジナルデータを、様々な属性データを含んだままで保存する。

以上のような状況を踏まえ、本処理系においてはまず、記録され属性を載せた画面表示が行えることを最低限の目標とした。ここでは、各種の属性情報について、長さ無制限のテキストファイルとして扱っている。属性データの解読と解釈もまた、メタファイルの中で処理可能であり、データの作成目的に応じて記述されている属性情報は、テキストデータとしてシステムの内部に取り込まれる。メタファイルの中で、それらに関する解説を付け加えることも可能である。それらを将来、表示や利活用に際してどのように活用するかは、将来のユーザーに委ねられており、そのような試みを行おうとする将来の人々に対して、有効な参考となる情報を、例えば本稿のような形で永久保存しておくことが有効であると考える。

このことが保証されるためには、仮想コンバータの「仮想性」、言い換えると、特定のハードウェアやその上のオペレーティング・システム（OS）に依存せずに、様々な実行環境の上に容易に移植できることが求められる。本研究においては、前述のように、林晴比古氏がコンパイラ開発手法の教科書として作成した文献の付録 CD－ROM に参考添付されていたコンパイラ処理系を、発行元の許可を得て開発の出発点とし、整数型のみであった数値型を浮動小数、四元数に拡張するとともに、LSS-G 形式のコマンドを読み込む処理、およびそれらのコマンドが実行された場合に、処理系別に異なる動作を行う 4 の処理系を作成することで、メタファイルのコンパイル・実行処理の信頼性を高めると共に、異なる OS 上での動作の確認を行った。

（３）IFC 形式

IFC 形式は、BIM の標準形式として勧奨されているファイル形式である。本研究においては、平成 24 年度に、除却建物の実測図（１６棟分）からサンプルデータを作成し、これを対象として、仮想コンバータのメタファイルを作成した。

①資料等

IFC 形式に関しては、解説書が公開されている。

Thomas Liebich “IFC2xEdition3 Model Implementation Guide” Version 2.0 May 18, 2009, building SmMART International, Modeling Support Group

②データ形式の概要

このデータ形式は、深い階層を有する。また、パーツを組み合わせて上位の空間単位の中に配置する際に、原点位置の相対移動と方位を指定することができる。

上記の解説書によると、IFC 形式のデータを構成するコマンドは多岐に亘っており、同一の建物を異なる方法で表現することが可能である。簡単のため、本研究において作成し

た除却建物のデータに用いられているコマンドを中心に概要を解説する。

1) 形式宣言

除却建物のデータは、冒頭に

ISO-10303-21;

の宣言があり、これに対応して末尾に

END-ISO-10303-21;

の宣言がある。ISO-10303 は、STEP のデータ交換形式として使用されているファイル形式である。

リスト 4-3-1 IFC 形式の保存データのヘッダ部分

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(
/* description */ ('A minimal AP214 example with a single part'),
/* implementation_level */ '2;1');
FILE_NAME(
/* name */ 'demo',
/* time_stamp */ '2003-12-27T11:57:53',
/* author */ ('Lothar Klein'),
/* organization */ ('LKSoft'),
/* preprocessor_version */ ' ',
/* originating_system */ 'IDA-STEP',
/* authorization */ ' ');
FILE_SCHEMA (('AUTOMOTIVE_DESIGN { 1 0 10303 214 2 1 1}'));
ENDSEC;
DATA;
#10=ORGANIZATION('00001','LKSoft','company');
#11=PRODUCT_DEFINITION_CONTEXT('part definition',#12,'manufacturing');
#12=APPLICATION_CONTEXT('mechanical design');
#13=APPLICATION_PROTOCOL_DEFINITION('','automotive_design',2003,#12);
#14=PRODUCT_DEFINITION('0',$,#15,#11);
#15=PRODUCT_DEFINITION_FORMATION('1',$,#16);
#16=PRODUCT('A0001','Test Part 1','',(#18));
#17=PRODUCT_RELATED_PRODUCT_CATEGORY('part',$,(#16));
#18=PRODUCT_CONTEXT('',#12,'');
#19=APPLIED_ORGANIZATION_ASSIGNMENT(#10,#20,(#16));
#20=ORGANIZATION_ROLE('id owner');
ENDSEC;
END-ISO-10303-21;
```

2) ヘッダ部分

HEADER;

から始まって、

ENDSEC;

までがヘッダ部分である。

ファイル名、データ形式のバージョン番号等が記されている。

3) データ部分

DATA;

から始まって、

ENDSEC;

までがデータ部分である。

データ部分は、

#行番号=IFCxxxxx(引数);

の形式のコマンド行により記述されている。行番号は、必ずしも昇順になっていることを要しないが、本研究で作成した記録データ(４－６)の場合には、１から始まって１づつ増加するように整理されている。

コマンドは、IFC から始まる、C 言語の関数のような形式で記述されており、すべて大文字である。左辺は変数名（シンボル）ではなく、＃数字という形式で行番号が示されている。右辺のコマンド（シンボル文字列）の後に続く（引数）には、「#320」の書式による行番号による下位オブジェクトの参照、「’part’」のような文字列、「3.14」のような数値、「＄」による省略形（デフォルト値）などがある。また、引数の１項目として、「（#10,#20,#30）」のように、複数のオブジェクトを（）で束ねたような表現も用いられれている。

立体を構成する面は基本的には三角形を単位として記述している。窓を有する壁等については、穴あき図形の表現を使用せず、三角形に分割した形で表現している。

リスト 4-3-2 IFC 形式の入力データ例の構成

```
  このデータにおけるデータの構成について
(1)面のレベル（総数
  #31=FACE(32)
  #32=FACEOUTERBOUND(#33,.T.)
  #33=POLYLOOP(#34,#35,#36);
  #34,#35,#36 CARTESIANPOINT
(2)面群(総数)
  #21=BUILDINGELEMENTPROXY(#25)
  #25=PRODUCTREPRESENTATION(#26)
  #26=SHAPEREPRESENTATION(#27)
  #27=FACEBASEDSURFACEMODEL(#30)
  #30=CONNECTEDFACESET(#長いリスト)
(3)属性(総数４)
  #757=COLOURRGB(3値)
  #758=SURFACESTYLERENDERING(#757)
  #759=SURFACESTYLE(#758)
  #760=PRESENTATIONSTYLEASSIGNMENT(#759)
  #761=STYLEDITEM(#27面群,#760)：個別の面ではなく面群に適用している
  #762=RELCONTAINEDINSPATIOALSTRUCTURE(#21,#13)
(4)配置情報（総数４）
  #1=プロジェクト=敷地(#15812)＋建物(#15817)
  建物(#11)＝各階(4)(#13,#15,#17,#19)
  #13=BUILDINGSTOREY(日本語,#15822)
  #14=RELAGGREGATES(#11,#13)総数６：BUILDINGSTOREY(4),SITE(1)
  #15822=LOCALPLACEMENT(#15818)
  #15818=AXIS2PLACEMENT3D(三つの方向と座標)
(5)全体情報（総数１）
  執筆者、名称、所在、単位
```

座標値は、IFCCARTESIANPOINT(x,y,z)　により定義されている。

IFCPOLYLOOP(座標値,・・・,座標値);　により、頂点列が定義されている。このデータの場合には全て三角形である。

IFCFACEOUTERBOUND(頂点列番号,.T.);　により、閉多角形が定義されている。面の

中に穴あきがあるような場合には IFCFACE((閉多角形番号)); により面が定義されている。

面は、外形と中島（穴）を含む記述ができるようであるが、今回扱ったデータの場合には全て単純な三角形となっている。

IFCCONNECTEDFACESET(面,・・・,面); により面の集合体が定義されている。面の列挙数は場合によっては数千に及ぶ、長い１行となっている。

4) 上位構造

建物全体を表す PROJECT の配下に敷地、階別のデータがあり、それぞれが相対的な位置・方位で配置された階層的な構造が記述されている。

5) 色彩等の定義

IFC 形式においては、色彩は面群で構成されたオブジェクトに対して設定される。しかも、オブジェクトの定義に際して、色彩等の属性を必須項目として参照することはなく、ファイルの後方で、すでに定義されたオブジェクトに対する選択的な追加属性として定義されるようになっている。具体的には、まず、IFCCOLOURRGB コマンドで RGB の各値を定義する。

次にこれを、IFCSURFACESTYLERENDERING コマンドで参照する。

次にこれを IFCSURFACESTYLE で参照する。

次にこれを IFCPRESENTATIONSTYLEASSIGNMENT で参照する。

次にこれを IFCSTYLEDITEM で、IFCFACEBASEDSURFACEMODEL と関連づける。このコマンドは二つの引数をもち、第一引数で適応対象を、第二引数で適用属性を示す。

IFCFACEBASEDSURFACEMODEL は、直下に IFCFACEBASEDSURFACEMODEL オブジェクトを複数有する階層であり、上位に IFCSHAPEREPRESENTATION、さらにその上位に IFCPRODUCTREPRESENTATION を有する中間的な階層である。

更に上位には IFCBUILDINGELEMENTPROXY がある。

その上位に IFCRELCONTAINEDINSPATIALSTRUCTURE がある。

更に IFCBUILDINGSTOREY があり、複数階と敷地が合わさって、プロジェクトのトップレベルがある。

以上解説したように、IFC 形式においては、末端の座標値や図形を直接記述した階層から、上位の敷地や建物を定義した階層までの間に多くの中間的な階層が存在するが、その多くは、一つの上位階層オブジェクトが一つの下位階層オブジェクトだけを参照するような関係となっている。

従って、IFC 形式で記録された建物を解読して表示などの目的に利活用するためには、これら多くの中間的な階層を丹念に再現する必要は必ずしもないように見える。

③ 階層図

オブジェクトは図 4-3-4 のような階層で構成されている。

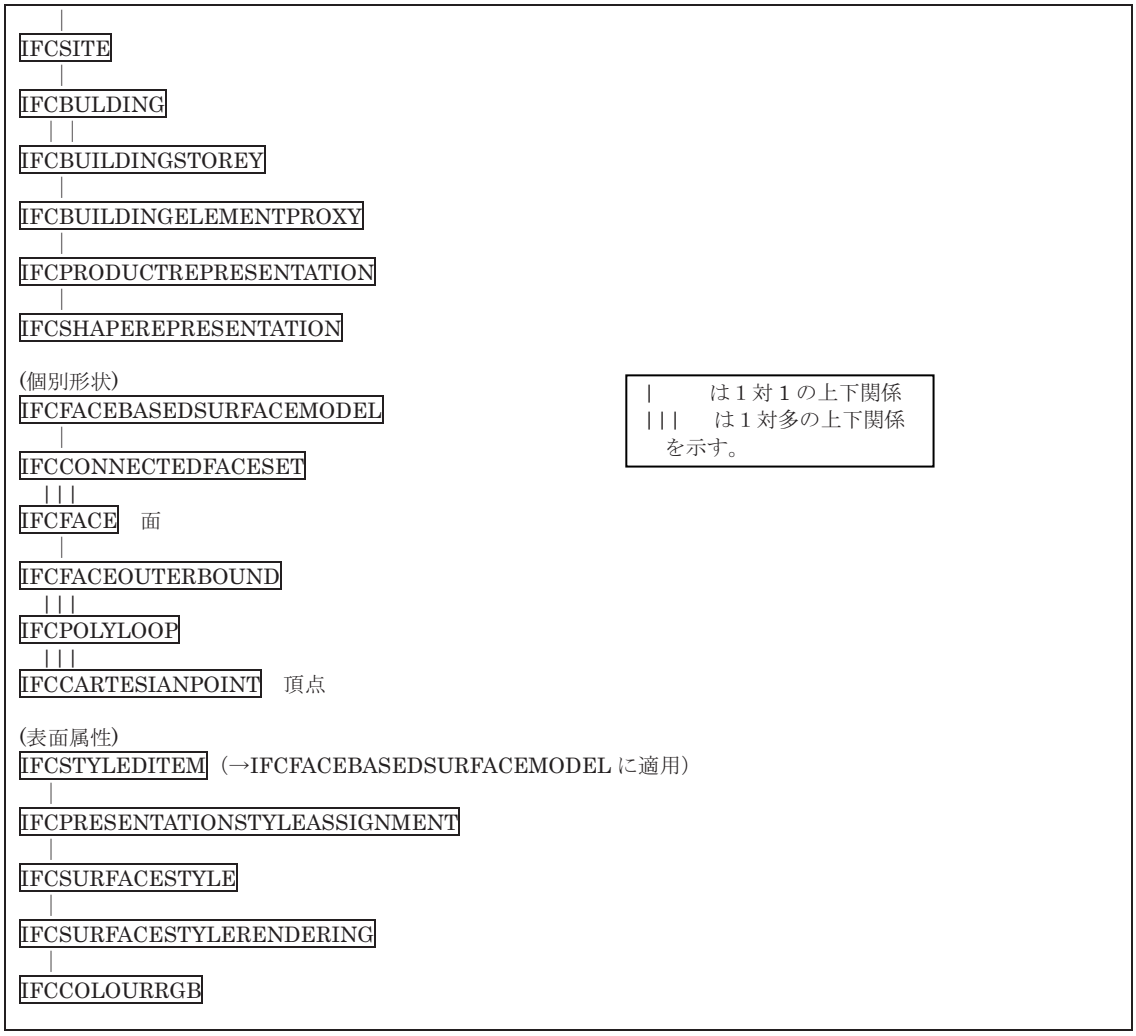


図 4-3-4 IFC 形式階層図

この階層構造を記述する方法として、上位オブジェクトの引数として下位オブジェクトが参照されている場合(a)と、二つのオブジェクトを連結するためのリンクオブジェクトの引数として上位オブジェクトと下位オブジェクトの両方を記述する方法が採られている場合(b)がある。

a. 上位オブジェクト（下位オブジェクト）

b. ンクオブジェクト（上位オブジェクト、下位オブジェクト）

④ メタファイル

この IFC 形式の場合には、上位の要素（例えば面の定義）から下位の要素（例えば頂点座標）に、前方参照が行われている。従って、ワンパスのボトムアップでデータを構築することができず、未解決の参照をスタックする必要がある。

形式定義ファイルは、IFC 形式の全ての仕様に対応する必要はなく、保存対象となる除却建物の記録データに含まれる意味のある情報を全て取り出すことができれば十分であることから、以下の方法とした。

1) ２パスで解析することとし、最初のパスで、データファイルで使用されている全ての

IFCxxxx という名称のコマンドをリストを作成するとともに、全ての行の開始番地を格納したリストを作成する。このリストは整数型の配列として作成し、行番号を添字として、各行の IFC の次のアドレスを格納する。

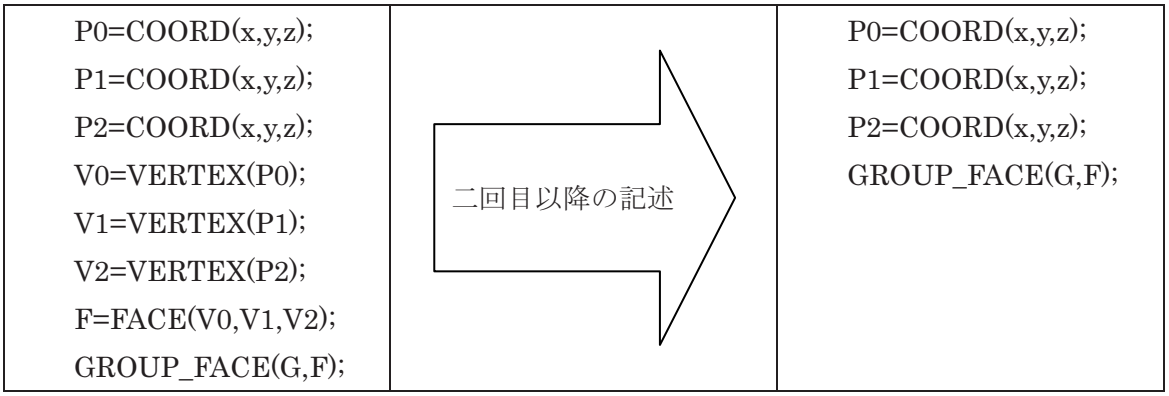
2) IFCCONNECTEDFACESET コマンドを起点として、それが参照する IFCFACE コマンド行に順次アクセスし、面のデータを生成する。

3) IFCFACE の解析にあたっては、一つの IFCFACE が参照する一つの IFCFACEOUTERBOUND, さらに一つの IFCPOLYLOOP の行にアクセスする。

4) IFCPOLYLOOP が参照する 3 の IFCCARTESIANPOINT にアクセスし、三角形を一つ生成する。

5) 面（三角形）の生成にあたっては、それが参照する 3 頂点の座標が全て異なっていることを確認し、もし重複する 2 以上の頂点があれば、面を生成しない。

6) 面の生成に当たっては、まずオブジェクト G を GROUP（）コマンドにより作成した上で、これに所属する面を GROUP_FACE コマンドで割りつけていく。



その際、このように三角形だけで構成されたモデルの場合には、全ての面の登録に際してこの 8 行を再三実行する必要はなく、二回目以降は P0～P2 の再定義を行った上で、GROUP_FACE(G,F)を実行するだけでよい（4 行の記述)。言い換えると、最初の面の定義に使用した面オブジェクト F を、以後の面定義のテンプレートとして使用することができる。

7) 属性文字列

上位のオブジェクトには、属性を示すための、例えば「歴史的建物」といった文字列が定義されている。Shift-JIS の日本語の文字を 16 進数で表現するために、以下のフォーマットが使用されている。

'¥X¥89¥X¥AE¥X¥8D¥X¥AA¥X¥00¥X¥AA'

これは、0x89ae（屋）, 0x8daa（根） という 2 文字の漢字を表現している。

⑤ データの欠陥

データ形式としてのルールには従っているものの、幾何学的には問題のあるデータが記述され混在している。これらについては、形式定義ファイル（メタファイル）の中でエラーチェック方法を定義し、入力しないようにする必要がある。データ保存用の形式定義フ

ァイルの基本的な考え方として、保存データ自体の編集は行わず、保存データの不適切な部分についてはあるがままに受け入れ、支障のない入力処理を定義することとしている。

1) 大きさのない面

三角形を定義する頂点の内の二つが、同じ頂点座標をもつ場合に、面積のない面が生成し、表示に支障が生じる。このような面の入力はスキップした。

2) 面の逆転

3) 閉じていない立体

立体の表面を構成する面の一部が欠損したり逆向きとなっている欠陥があると、立体として接続した際に、体積が不定となる（４－６(5)②参照)。

リスト 4-3-3 IFC 形式のメタファイル

```
/***********************************
 *         IFC.CMM 150830           *
 ***********************************/

/*グローバル変数*/
int COMMANDS[1000],NC;//命令初出番地表、表長さ
int STAS[100],KANA[100],KIRI[100];//コマンド文字列辞書作成用
quat QUNIT;
int maxGF;//GROUP_FACEの最大面数

int posG; //現在処理中のグループの栞
int posGF; //グループ中のFACE
int tailGF; //前方参照Fの末尾ID
int VERTECES[4000];
int FACES[4000];
int LineCounter[40000];
int HOD; //データ領域の先頭
int Face;
int P0,P1,P2,V0,V1,V2;
//150829 プリコンパイル方式では、巨大な配列を必要とする
//最小の土質実験棟でも行番号：行、面のスタック
//このような場合、インタープリター方式とする必要がある

/*共通関数*/
void initglobal(){
  NC=0;
  maxGF=0;
  QUNIT=_Q(1.0,0.0,0.0,0.0);
}

void reportglobal(){//終了時、グローバル変数を表示する
  logf("#----------------REPORT-------------------¥n");
  logf("#NC=%d¥n",NC);
  logf("#maxGF=%d¥n",maxGF);
  logf("#QUNIT=%q¥n",QUNIT);
}

void thru(){//１行分、文字列をコメント出力する
          for(;;){
                    c = GETC();
                    if(c == 13){
                      logf("¥¥r");
                      continue;
                    }
```

```
                    if(c == 10){
                      logf("¥¥n");
                      break;
                    }
                    logf("%c", c);
          }
          logf("¥n");
}

void skip(){//１行分、スキップする
          for(;;){
                    c = GETC();
                    if(c == 13) continue;
                    if(c == 10) break;
          }
}

int standard(){//冒頭の１行：ISOコード等
          int c;
          logf("#");//コメントアウト
          thru();
          return 1;
}

int header(){
  logf("#HEADER SECTION¥n");
  while(!scanf(" ENDSEC;")){
    logf("#=");
    thru();
  }
  logf("#ENDSEC;¥n");
  return 0;
}

int strcmp(int pos1, int pos2){
  //「(」をターミネータとして、二つの文字列を比較
  int c1,c2,pos,i;
  pos = SIORI();
  for(i=0;;i++){
          SEEK(pos1+i);
          c1 = GETC();
          SEEK(pos2+i);
          c2 = GETC();
    if(c1<c2) return -1;
    else if(c1>c2) return 1;
    else if(c2=='(') break;//テキスト側の終端を検出
  }
  return 0;
}

int strcmp1(int pos){
  int i,rc;
  if(NC == 0){
    COMMANDS[0] = pos;
    STAS[0] = 1;
    NC = 1;
    return 1;
  }
  for(i=0;;){
    rc = strcmp(COMMANDS[i],pos);
    if(!rc){
      STAS[i]++;
      return 0;//既存
```

```
    }else if(rc<0){
      if(KANA[i] == 0){
        STAS[NC] = 1;//新規登録
        KANA[i] = NC;
        COMMANDS[NC++] = pos;
        return 1;
      }else{
        i = KANA[i];
        continue;
      }
    }else{
      if(KIRI[i] == 0){
        STAS[NC] = 1;//新規登録
        KIRI[i] = NC;
        COMMANDS[NC++] = pos;
        return 1;
      }else{
        i = KIRI[i];
        continue;
      }
    }
  }
  return 0;
}

void showcommand(int pos){
    logf("#");
          SEEK(pos);
          for(;;){
            c = GETC();
            if(c=='(') break;
            logf("%c",c);
          }
          logf("¥n");
}

void showprevcommand(int pos){
  int c,i;
  for(i=0;;i++){
    SEEK(pos-i);
    c = GETC();
    if(c == 10) break; //lf
  }
  thru();
}

void ERROR(int nl, int code, int pos){
  logf("#ERROR(%d)",code);
  logf(" in #%d¥n",nl);
  showprevcommand(pos);
  logf("#PROGRAM TERMINATED¥n");
  exit(0);
}

void showdic(){
  int i;
  logf("#-------------COMMANDS[%d]------------¥n",NC);
  for(i=0;i<NC;i++){
    logf("#COMMAND[%03d]:",i);
    logf("[%04d]:",STAS[i]);
          showcommand(COMMANDS[i]);
  }
}
```

```
void tree(int top){
   if(KIRI[top]) tree(KIRI[top]);
   showcommand(COMMANDS[top]);
   if(KANA[top]) tree(KANA[top]);
}

void showtree(){//木表現
   logf("#------------COMMANDTREE------------¥n");
   tree(0);
}

void inittriangle(){
  P0=COORD(0,0,0);
  P1=COORD(1,0,0);
  P2=COORD(1,1,0);
  V0=VERTEX(P0);
  V1=VERTEX(P1);
  V2=VERTEX(P2);
  Face=FACE(V0,V1,V2);
}

/*コマンド関数*/
int cartesianpoint1(int nl){
  int pos,i;
  int rc,ip,iv;
  quat q;
  pos = SIORI();
  SEEK(LineCounter[nl]);
  rc = scanf("CARTESIANPOINT((%qx,%qy,%qz)",q);
  q *= QUNIT;
  if(rc != 3) ERROR(nl,rc,SIORI());
  skip();
  SEEK(pos);
  return COORD(q);
}

int polyloop1(int nl){
  int pos,i,p0,p1,p2,rc;
  pos = SIORI();
  rc = 0;
  SEEK(LineCounter[nl]);
  if(!scanf("POLYLOOP(")){
   logf("#NOT POLYLOOP COMMAND¥n");
   thru();
  }else{
  //  logf("#%03d=POLYLOOP1 COMMAND¥n",nl);
      rc = scanf("(#%d",p0);
      rc += scanf(",#%d",p1);
      rc += scanf(",#%d)",p2);
      if(rc != 3){
        logf("#POLYLOOP FORMAT ERROR¥n");
        rc = 0;
      }else if(p0==p1){
        logf("#POLYLOOP ERROR(p0==p1)¥n");
        rc = 0;
      }else if(p1==p2){
        logf("#POLYLOOP ERROR(p1==p1)¥n");
        rc = 0;
      }else if(p2==p0){
        logf("#POLYLOOP ERROR(p2==p0)¥n");
        rc = 0;
      }else{
```

```
                        P0 = cartesianpoint1(p0);
                        P1 = cartesianpoint1(p1);
                        P2 = cartesianpoint1(p2);
                        rc = 1;
            }
  }
  SEEK(pos);
  return rc;
}

int faceouterbound1(int nl){
  int pos,i,rc;
  pos = SIORI();
  rc = 0;
  SEEK(LineCounter[nl]);
  if(scanf("FACEOUTERBOUND(")){
//    logf("#FACEOUTERBOUND COMMAND¥n");
      if(scanf("#%d,.T.",i)) rc = polyloop1(i);
      else logf("#FACEOUTERBOUND FORMAT ERROR¥n");
  }else{
    logf("#NOT FACEOUTERBOUND COMMAND¥n");
    thru();
  }
  SEEK(pos);
//  logf("#faceouterbound1 rc=%d¥n",rc);
  return rc;
}

int face1(int nl){
  int pos,i,rc;
  pos = SIORI();
  rc = 0;
  SEEK(LineCounter[nl]);
  if(scanf("FACE(")){
//    logf("FACE COMMAND¥n");
      if(scanf("(#%d)",i)) rc = faceouterbound1(i);
      else logf("#FACE FORMAT ERROR¥n");
  }else{
   logf("#NOT FACE COMMAND¥n");
   thru();
  }
  SEEK(pos);
  return rc;
}

void connectedfaceset(int nl){
  int rc,n,nf,iface,g,fedge,f;
  n = nf = fedge = 0;
  g = GROUP();
  rc = scanf("(");
  while(scanf("#%d",iface)){
    if(!scanf(",")) break;
//    Face = face1(iface);
    if(face1(iface)){
              GROUP_FACE(g,Face);
          }else{
            logf("#GROUP_FACE 不能(%d)¥n",nl);
          }
    n++;
  }
  logf("#GROUP_FACES:%d¥n",n);
  rc += scanf(")");
  if(rc != 2) ERROR(nl,rc,SIORI());
```

```
  skip();
}

void direction(int nl){
  skip();
}

void axis2placement3d(int nl){
  skip();
}

void localplacement(int nl){
  skip();
}

void siunit(int nl){
  int rc,n;
//(*,.LENGTHUNIT.,.MILLI.,.METRE.)
  rc = scanf("*,.LENGTHUNIT.,.MILLI.,.METRE.");
  if(!rc) ERROR(nl,rc,SIORI());
  QUNIT=_Q(0.001, 0.0, 0.0, 0.0);
  skip();
}

void unitassignment(int nl){ skip(); }//何もしない

/*解析ループ*/

//前方参照を多数含むデータの場合、メモリを大量に必要とする
// これを避けるためには、インタープリター方式でトップダウンの解析を行う
// 上位レベルの定義（グループ等）が前出する場合に、これをスタックする方が小さい
//→マイクロスタック方式（後方参照は即出力、前方参照は前方の範囲を含めてスタック
// 解析行番号が前方参照の範囲を超えた要素についてはスタックから外して出力する

int pass1(){//行番号に対応する、栞位置を登録する
  int addr,nl;
  int pos,i;

  HOD = SIORI();

  if(scanf(" DATA;")){
          logf("#DATA SECTION pass1¥n");
  }else{
    logf("#DATA SECTION ERROR¥n");
    return 0;
  }

  while(!scanf(" ENDSEC;")){
//    addr = SIORI();
    rc = scanf(" #%d=IFC",nl);
    if(rc){
      LineCounter[nl] = SIORI();
      pos = SIORI();
      i = strcmp1(pos);
      SEEK(pos);
      skip();
    }else{
      logf("#rc=%d¥n",rc);
      logf("#ERROR after nl=%d¥n",nl);
      break;
    }
  }
  SEEK(HOD);
```

```
  logf("#pass1 end¥n");
  return 1;
}

int pass2(){
  int addr,nl;

  SEEK(HOD);
//  initglobal();
  inittriangle();
//  return 0;
  if(scanf(" DATA;")){
          logf("#DATA SECTION pass2¥n");
  }else{
    logf("#DATA SECTION ERROR pass2¥n");
    return 0;
  }

  while(!scanf(" ENDSEC;")){
    addr = SIORI();
    rc = scanf(" #%d=IFC",nl);
    if(rc){
      if(scanf("CONNECTEDFACESET(")) connectedfaceset(nl);
      else if(scanf("SIUNIT(")) siunit(nl);
      skip();
    }else{
      logf("#rc=%d¥n",rc);
      logf("#ERROR after nl=%d¥n",nl);
      break;
    }
  }
  SEEK(HOD);
  logf("#pass2 end¥n");
  logf("#NC=%d¥n",NC);
  showdic();
  showtree();
  return 1;
}

void data(){
  pass1();
  pass2();
}

int main(){
  int i,n;
  float x,y,z;
  float xmax,xmin,ymax,ymin;
  standard();
  header();
  data();
  exit( -1);
}

/*参考資料
  このデータにおけるデータの構成について
 (1)面のレベル (総数
   #31=FACE(32)
   #32=FACEOUTERBOUND(#33).T.
   #33=POLYLOOP(#34,#35,#36);
   #34,#35,#36 CARTESIANPOINT
 (2)面群(総数)
   #21=BUILDINGELEMENTPROXY(#25)
```

```
   #25=PRODUCTREPRESENTATION(#26)
   #26=SHAPEREPRESENTATION(#27)
   #27=FACEBASEDSURFACEMODEL(#30)
   #30=CONNECTEDFACESET(#長いリスト)
 (3)属性(総数)
   #757=COLOURRGB(3値)
   #758=SURFACESTYLERENDERING(#757)
   #759=SURFACESTYLE(#758)
   #760=PRESENTATIONSTYLEASSIGNMENT(#759)
   #761=STYLEDITEM(#27麺,#760) : 個別の面ではなく麺に適用している
   #762=RELCONTAINEDINSPATIOALSTRUCTURE(#21,#13)
 (4)配置情報 (総数４)
   #1=プロジェクト=敷地(#15812)＋建物(#15817)
   建物(#11)＝各階(4)(#13,#15,#17,#19)
   #13=BUILDINGSTOREY(日本語,#15822)
   #14=RELAGGREGATES(#11,#13)総数６ : BUILDINGSTOREY(4),SITE(1)
   #15822=LOCALPLACEMENT(#15818)
   #15818=AXIS2PLACEMENT3D(三つの方向と座標)
 (5)全体情報 (総数１)
   執筆者、名称、所在、単位
COMMAND[011]:[0004]:BUILDINGSTOREY
          COMMAND[012]:[0004]:BUILDINGELEMENTPROXY
          COMMAND[013]:[0004]:PRODUCTREPRESENTATION
          COMMAND[014]:[0004]:SHAPEREPRESENTATION
          COMMAND[015]:[0004]:FACEBASEDSURFACEMODEL
          COMMAND[016]:[0004]:CONNECTEDFACESET
COMMAND[017]:[0004]:COLOURRGB
COMMAND[018]:[0004]:SURFACESTYLERENDERING
COMMAND[019]:[0004]:SURFACESTYLE
COMMAND[020]:[0004]:PRESENTATIONSTYLEASSIGNMENT
COMMAND[021]:[0004]:STYLEDITEM
COMMAND[022]:[0004]:RELCONTAINEDINSPATIALSTRUCTURE
*/
```

（４）ＬＳＳ－Ｇ形式

ＬＳＳ－Ｇ形式は、国土交通省版・景観シミュレーション・システム（1996）において、建物や地形等の形状を記述するための外部ファイル形式であり、メタファイル形式のサブセットとなっている。即ち、この形式のファイルをメタファイルとして処理することにより、固定形状を記述したメタファイルとなる。

この形式のデータファイルを保存のためにデータファイル側として指定し、この形式の定義を記録するためには、ＬＳＳ－Ｇ形式を定義したメタファイルを作成した。

①資料等

ＬＳＳ－Ｇ形式については、国総研報告第４２号[34]に解説している。また、まちづくり・コミュニケーション・システムのＷＥＢサイト（４－９①）からも公開されている。

②メタファイル

奥尻島の町並、福島市内の町並、釜石市内の町並等はＬＳＳ－Ｇ形式で作成されている。これらのデータを保存するために、ＬＳＳ－Ｇ形式を解読するメタファイルを添付し、データファイルとして処理できるようにした。

リスト 4-3-4 LSS-G 形式のメタファイル

```
int RESULT,CONCAVE_STATUS,COMMENT_STATUS,DEBUG;
```

```
void version(){
		logf("#test 14¥n");
		logf("#140429 VERTEXコマンドの様々な書式への対応¥n");
		logf("#140628 LINE, LINE_VERTEXコマンドへの対応¥n");
		logf("#140719 COMMENT(ON/OFF) 文に対応,#[EOF]に対応¥n");
}

void check(){
		int c;
		while((c = GETC())!=';'){
		 if(c<32) logf("(0x%x)",c);
		 else logf("%c",c);
		 if(c<0) break; //EOFなど
		}
		logf(";¥n");
}

int error(int siori){
		SEEK(siori);
		logf("#unknown command[");
		check();
		logf("]¥n");
		RESULT = 9;
		return 0;
}

int commentout(int siori){ //#
		int c;
		SEEK(siori);
		while((c = GETC())!='¥n'){
			if(c == '#') logf("#");
			else if(c == '¥t') logf("[tab]");
			else if(c == -1) break; //EOF
			else logf("%c",c);
		}
		logf("¥n");
		return 1;
 }

int errorline(int siori){
		int c;
		SEEK(siori);
		logf("#ERROR:");
		while((c = GETC())!='¥n'){
			if(c == '¥t') logf("[tab]");
			else logf("%c",c);
		}
		logf("¥n");
		return 0;
}

int labeldebug(){
		int val,rc,H,K;
		scanf(" ");//これで、改行、タブ、スペースを全て読み飛ばす
		H = SIORI();
		rc = scanf(" %[^ =]",val);

		K = SIORI();
		logf("label[");
		for(SEEK(H);SIORI()<K;){
			c = GETC();
			logf("%c",c);
		}
		logf("]");

		if(rc) logf("(%d)¥n",val);
		else logf("(-1)¥n");
```

```
		if(rc) return val;//ラベル変数のアドレスを返す
		return -1;
}

int labelnolog(){
		int val,rc;
		scanf(" ");//これで、改行、タブ、スペースを全て読み飛ばす
		rc = scanf(" %[^ =]",val);
		if(rc) return val;//ラベル変数のアドレスを返す
		return -1;
}

int label(){
		if(DEBUG) return labeldebug();
		return labelnolog();
}

int comment(int siori){ //COMMENT(ON/OFF);
		int rc;
		rc = scanf("(");
		if(scanf("ON")) COMMENT_STATUS = 1;
		else if(scanf("OFF")) COMMENT_STATUS = 0;
		else return error(siori);
		scanf("%*[^)] )");
		rc += scanf(");");
		logf("COMMENT(%d)",COMMENT_STATUS);
		logf(" rc=%d¥n", rc);
		if(rc != 2) return error(siori);
		return 1;
}

int normal(int siori){
		int pn;
		float x,y,z;
		SEEK(siori);
		scanf(" %[^= ] =",pn);
		scanf(" NORMAL(%f",x);
		scanf(",%f",y);
		scanf(",%f);",z);
		*pn = NORMAL(x,y,z);
		return 2;
}

int color(int siori){
		int pc,h;
		float r,g,b,a;
		SEEK(siori);
		h = scanf(" %[^= ] =",pc);
		h = h + scanf(" COLOR(%f",r);
		h = h + scanf(",%f",g);
		h = h + scanf(",%f",b);
		h = h + scanf(",%f",a);
		scanf(" );");

		if(h==4){
		  *pc = COLOR(r,g,b);
		}else if(h==5){
		  *pc = COLOR(r,g,b,a);
		}
		return 2;
}

int group(int siori){
		int pg,hasil;
		SEEK(siori);
		hasil = scanf(" %[^ =]",pg);
		hasil += scanf(" = GROUP();");
```

```
	if(hasil != 2){
		return error(siori);
	}
	*pg = GROUP();
	return 3;
}

int group1(int siori){//属性付き立体
	int pg,head,len;
	int hasil;
	SEEK(siori);
	hasil = scanf(" %[^ =]",pg);
	hasil += scanf(" = GROUP(");
	head = SIORI();
	hasil += scanf(" %*[^)]%n",len);
	hasil += scanf(");");
	if(hasil != 3){
		logf("#GROUP(str); scan fail(h=%d)¥n",hasil);
		RESULT = 3;
		return 0;
	}
	*pg = GROUP();
	GROUP_ATTR(pg,head,len);
	return 3;
}

int material2(int siori){ //140428 マテリアル
	int pm, fname[2], hasil;
	SEEK(siori);
	hasil = scanf(" %[^ =]",pm);
	hasil += scanf(" = MATERIAL(");
	fname[0] = SIORI();
	hasil += scanf(" %*[^)]%n",fname[1]);
	hasil += scanf(");");
//	logf("# MATERIAL(pf=%d)¥n", pf);
	if(hasil != 3){
		logf("# MATERIAL scan fail(h=%d)¥n", hasil);
		logf("#fname[]=(%d,",fname[0]);
		logf("%d);¥n",fname[1]);
		RESULT = 3;
		return 1;//for debug
	}
//	*pm = MATERIAL(*fname);
	return 3;
}

int material(int siori){ //140428 マテリアル
	int pm, pf, hasil;
	SEEK(siori);
	hasil = scanf(" %[^ =]",pm);
	hasil += scanf(" = MATERIAL(");
	hasil += scanf(" %[^)]",pf);
	hasil += scanf(");");
//	logf("# MATERIAL(pf=%d)¥n", pf);
	hasil += scanf(");");
	if(hasil != 4){
		logf("# MATERIAL scan fail(h=%d)¥n", hasil);
		RESULT = 3;
		return 0;
	}
	*pm = MATERIAL(*pf);
//	*pm = MATERIAL(pf);
	return 3;
}

int coord(int siori){
	int i;
	float x,y,z;
```

```
		i = -1;
		SEEK(siori);
		i = label();
//		scanf(" %[^= ]",i);
		scanf(" = COORD(%f",x);
		scanf(",%f",y);
		scanf(",%f);",z);
//		logf("#coord(%d)",i);
//		logf("[%d]¥n",*i);
		if(0<=i){
			*i = COORD(x,y,z);
			return 4;
		}else{
			logf("#coord label error(i=%d)¥n",i);
		}
		return 0;
}

int rnolog(){//現在の場所から定義済の変数名を読み込み、その内容を返す。空白なら-1を返す(検証ログなし)
		int val,rc;
		rc = scanf(" ");

		val = -1;
		rc = scanf("%[^ ,)]",val);

		if(rc) return val;//変数アドレスを返す
		return -1;//変数アドレスが-1 なら、引数なし
}

int rlog(){//現在の場所から定義済の変数名を読み込み、その内容を返す。空白なら-1を返す(検証ログあり)
		int val,rc,H,K;
		rc = scanf(" ");

		H = SIORI();
		val = -1;
		rc = scanf("%[^ ,)]",val);
		K = SIORI();
		logf("[");
		for(SEEK(H);SIORI()<K;){
			c = GETC();
			logf("%c",c);
		}
		logf("]");

		if(rc) logf("(val=%d)",val);
		else logf("(val=undefined)");

		if(rc) return val;//変数アドレスを返す
		return -1;//変数アドレスが-1 なら、引数なし
}

int r(){
		if(DEBUG) return rlog();
		return rnolog();
}

int vertexlog(int siori){
		int v,p,n,c,t,l,h,pv;//,kata;
		SEEK(siori);
		p = n = c = t = pv = -1;
		logf("#V");

		v = label();
		logf("#(%d)=",*v);
		if(!scanf(" = VERTEX( ")){
			return error(siori);
		}
```

```
		logf("VERTEX(");

		p = r();
		logf(":P[%d],",p);

		if(scanf(" );")){
			h = 3;//正常終了
			logf(");¥n");
			*v = VERTEX(*p);
			return 1;
		}
		if(!scanf(" ,")){//nへ続行しない
			return 0;//error(siori);
		}

		n = r();
		logf(":N[%d],",n);

		if(scanf(" );")){
			h = 3;//正常終了
			logf(");¥n");
			*v = VERTEX(*p,*n);
			return 1;
		}
		if(!scanf(" ,")){//cへ続行しない
			return 0;
		}
		c = r();
		logf(":C[%d],",c);

		if(scanf(" );")){
		    logf(");¥n");//正常終了
			*v = VERTEX(*p,*n,*c);
			return 3;
		}
		if(!scanf(" ,")){//tへ続行しない
			h = 4;//異常終了
			return 0;
		}

		t = r();
		logf(":T[%d],",t);

		if(scanf(" );")){
		    logf(");¥n");//正常終了
			*v = VERTEX(*p,*n,*c,*t);
			return 4;
		}
		if(!scanf(" ,")){//pvへ続行しない
			h = 4;//異常終了
			return 0;
		}

		pv = r();
		logf(":PV[%d]",pv);

		if(scanf(" );")){
		    logf(");¥n");//正常終了
			*v = VERTEX(*p,*n,*c,*t,*pv);
			return 5;
		}
		errorline(siori);
		return 0;
}

int vertex(int siori){
		int v,p,n,c,t,l,h,pv;//,kata;
```

```
	if(DEBUG) return vertexlog(siori);

	SEEK(siori);
	p = n = c = t = pv = -1;

	v = label();
	if(!scanf(" = VERTEX( ")){
		return error(siori);
	}

	p = r();

	if(scanf(" );")){
		h = 3;//正常終了
		*v = VERTEX(*p);
		return 1;
	}
	if(!scanf(" ,")){//nへ続行しない
		return 0;//error(siori);
	}

	n = r();

	if(scanf(" );")){
		h = 3;//正常終了
		*v = VERTEX(*p,*n);
		return 1;
	}
	if(!scanf(" ,")){//cへ続行しない
		return 0;
	}
	c = r();

	if(scanf(" );")){
		*v = VERTEX(*p,*n,*c);
		return 3;
	}
	if(!scanf(" ,")){//tへ続行しない
		h = 4;//異常終了
		return 0;
	}

	t = r();

	if(scanf(" );")){
		*v = VERTEX(*p,*n,*c,*t);
		return 4;
	}
	if(!scanf(" ,")){//pvへ続行しない
		h = 4;//異常終了
		return 0;
	}

	pv = r();

	if(scanf(" );")){
		*v = VERTEX(*p,*n,*c,*t,*pv);
		return 5;
	}
	errorline(siori);
	return 0;
}

int relog(){ /*定義済の変数をスキャンする。空白は許されない。
    成功すると、変数のアドレスを返す
	変数が見いだせなければ０
	未定義の変数ならば-1を返す*/
```

```
		int val,rc,H,K;
		rc = scanf(" ");
		rc = scanf("%*[ ¥t¥r¥n]");

		H = SIORI();
		val = 0;
		rc = scanf(" %[^ ,)¥n¥r]",val);
		K = SIORI();
		logf("#[");
		for(SEEK(H);SIORI()<K;){
			c = GETC();
			logf("%c",c);
		}
		logf("]");

		if(rc) logf("(val=%d)",val);
		else logf("(val=undefined)");

		if(rc) return val;//変数アドレスを返す
		return 0;//変数アドレスが-1 なら、引数なし
}

int re(){
		int val,rc,H,K;
		if(DEBUG) return relog();
		rc = scanf(" %*[ ¥t¥r¥n]");
		val = 0;
		rc = scanf(" %[^ ,)¥n¥r]",val);
		if(rc) return val;//変数アドレスを返す
		return 0;//変数アドレスが-1 なら、引数なし
}

int face(int siori){
		int i,pf,j,k[1000],l,v;
		SEEK(siori);
		scanf(" %[^ =] = ",pf);
		scanf(" FACE(%[^ ,)] ,",k[0]);
//logf("###k[0]=%d¥n",k[0]);
		for(j=1;j<1000;j++){
//			if(!scanf(" %[^ ,)] ,",k[j])) break;
			k[j] = re();//空白は許されない
			if(!k[j]) break;
			scanf(" ,");
		}
//		logf("#face頂点数:%d¥n",j);
		if(j == 1000){
			logf("#face:頂点(=%d)が以上は未対応¥n",j);
			scanf("%*[^)]);");
			return 6;
		}else{
			scanf(" );");
		}
		switch(j){
		case 0:
		case 1:
		case 2:
			logf("#face:頂点が未満は不正¥n");
			return 6;
		default:
			*pf = FACE();
//logf("#face(%d)¥n",*pf);
			for(l=0;l<j;l++){
				FACE_VERTEX(*pf,*k[l]);
//				logf("#FACE(%d)",pf);
//				logf("[%d]=V",*pf);
//				logf("(%d)",k[l]);
//				logf("[%d]¥n",*k[l]);
```

```
                        }
        }/*switch*/
//      printf("#face%d",j);
//      logf("(%d)",pf);
//      logf("[%d]¥n",*pf);
        return 6;
}

int line(int siori){
        int i,pf,j,k[1000],l,v;
        SEEK(siori);
        scanf(" %[^ =] = ",pf);
        scanf(" LINE(%[^ ,)] ,",k[0]);
//logf("###k[0]=%d¥n",k[0]);
        for(j=1;j<1000;j++){
//              if(!scanf(" %[^ ,)] ,",k[j])) break;
                k[j] = re();//空白は許されない
                if(!k[j]) break;
                scanf(" ,");
        }
//      logf("#face頂点数:%d¥n",j);
        if(j == 1000){
                logf("#face:頂点(=%d)が以上は未対応¥n",j);
                scanf("%*[^)]);");
                return 6;
        }else{
                scanf(" );");
        }
        switch(j){
        case 0:
        case 1:
                logf("#line:頂点が未満は不正¥n");
                return 0;
        default:
                *pf = LINE();
if(DEBUG) logf("#LINE()戻り値 : %d¥n",*pf);
                for(l=0;l<j;l++){
                        LINE_VERTEX(*pf,*k[l]);
if(DEBUG){
                        logf("#LINE_VERTEX(%d",pf);
                        logf("[%d],",*pf);
                        logf("(%d)",k[l]);
                        logf("[%d])¥n",*k[l]);
}
                }
        }/*switch*/
//      printf("#line%d",j);
//      logf("(%d)",pf);
//      logf("[%d]¥n",*pf);
        return 6;
}

int face_normal(int siori){
        int pf,pn;
        SEEK(siori);
        scanf(" FACE_NORMAL(%[^,],",pf);
        scanf(" %[^)]);",pn);
//      logf("#face_normal(%d)",pf);
//      logf("[%d],",*pf);
//      logf("(%d)",pn);
//      logf("[%d]¥n",*pn);
        FACE_NORMAL(*pf,*pn);

        return 7;
}

int face_color(int siori){
```

```
		int pf,pc,hr;
		hr = 0;
		SEEK(siori);
		hr += scanf(" FACE_COLOR(%[^,],",pf);
		hr += scanf(" %[^)]);",pc);
		FACE_COLOR(*pf,*pc);
		return 7;
}

int face_material(int siori){
		int pf,pm,hr;
		hr = 0;
		SEEK(siori);
		hr += scanf(" FACE_MATERIAL(%[^,],",pf);
		hr += scanf(" %[^)]);",pm);
		FACE_MATERIAL(*pf,*pm);
		return 7;
}

int group_face(int siori){
		int pg,pf;
		int rc;
		SEEK(siori);
		rc = scanf(" GROUP_FACE(%[^, ]",pg);
		if(!rc) logf("#GROUP_FACE:*pg 取得失敗¥n");
//		else logf("#GROUP_FACE refer G[%d]¥n",pg);
		while(scanf(" ,")){
			rc = scanf(" %[^), ]",pf);
			if(!rc) logf("#GROUP_FACE:*pf 取得失敗¥n");
//			else logf("#GROUP_FACE refer F[%d]¥n",pf);
//logf("GROUP_FACE(%s,xx);¥n",*pg);
			GROUP_FACE(*pg,*pf);
//			logf("#group_face(%d)",pg);
//			logf("[%d],",*pg);
//			logf("(%d)",pf);
//			logf("[%d]¥n",*pf);
		}
		if(scanf(" );")) return 8;
		logf("#GROUP_FACE format error¥n");
		RESULT = 8;
		return 0;
}

int group_line(int siori){
		int pg,pf;
		int rc;
		SEEK(siori);
		rc = scanf(" GROUP_LINE(%[^, ]",pg);
		if(!rc) logf("#GROUP_LINE:*pg 取得失敗¥n");
		while(scanf(" ,")){
			rc = scanf(" %[^), ]",pf);
			if(!rc) logf("#GROUP_LINE:*pf 取得失敗¥n");
			GROUP_LINE(*pg,*pf);
		}
		if(scanf(" );")) return 8;
		logf("#GROUP_LINE format error¥n");
		RESULT = 8;
		return 0;
}

int group_material(int siori){
		int pg,pm,hasil;
		SEEK(siori);
		hasil = scanf(" GROUP_MATERIAL(%[^, ],",pg);
		hasil += scanf(" %[^), ]",pm);
		GROUP_MATERIAL(*pg,*pm);
//		logf("#GROUP_MATERIAL(*pm=%d)",*pm);
```

```
//              logf("pm=[%d]¥n",pm);
//              logf("str=%s",pm);
                hasil += scanf(" );");
                if(hasil == 3) return 8;
//              logf("#GROUP_MATERIAL format error(hasil=%d)¥n", hasil);
                RESULT = 8;
                return 0;
}

int link(int siori){
                int pk,pg1,pg2;
                int hasil;
                SEEK(siori);
                hasil = scanf(" %[^= ] =",pk);
                hasil += scanf(" LINK( %[^ ,] ,",pg1);
                hasil += scanf(" %[^ )] );",pg2);
                if(hasil != 3){
                                logf("#LINK scan fail¥n");
                                RESULT = 9;
                                return 0;
                }
                *pk = LINK(*pg1,*pg2);
                return 9;
}

int link_xform(int siori){
                int i,pk,n,order,form,size;
                float m[16];
                SEEK(siori);
                scanf(" LINK_XFORM(%[^,]",pk);
//              logf("##pk=%d¥n",pk);
//              logf("##*pk=%d¥n",*pk);
                order = 0;
                if(scanf(" , LOAD")) order = LOAD;
                else if(scanf(" , PRE")) order = PRE;
                else if(scanf(" , POST")) order = POST;
                else{
                   RESULT = 101;
                   return 0;
                }
//              logf("#link order=%d¥n",order);

                form = size = 0;
                scanf(" ,");
                if(scanf(" IDENTITY")){
//                 form = 1; size=0;
                   form = IDENTITY; size=0;
                }else if(scanf(" TRANSLATE")){
//                 form = 2; size=3;
                   form = TRANSLATE; size=3;
                }else if(scanf(" ROTATE_X")){
//                 form = 3; size=1;
                   form = ROTATE_X; size=1;
                }else if(scanf(" ROTATE_Y")){
//                 form = 4; size=1;
                  form = ROTATE_Y; size=1;
                }else if(scanf(" ROTATE_Z")){
//                 form = 5; size=1;
                  form = ROTATE_Z; size=1;
                }else if(scanf(" ROTATE_A")){
//                 form = 6; size=4;
                  form = ROTATE_A; size=4;
                }else if(scanf(" SCALE")){
//                 form = 7; size=3;
                  form = SCALE; size=3;
                }else if(scanf(" MATRIX")){
//                 form = 8; size=16;
                  form = MATRIX; size=16;
```

```
		}else{
		  logf("#unknown form[");
		  check();
		  logf("]¥n");
		}

		if(form <1){
		   logf("#LINK_XFORM form error(n=%d)¥n",n);
		   RESULT = 102;
		   return 0;
		}

		for(i=0;i<size;i++){
		   scanf(" ,%f", m[i]);
//		   logf("#m[%d]=",i);
//		   logf("%f¥n",m[i]);
		}
		scanf(" );");

//
		LINK_XFORM(*pk,LOAD,MATRIX,m[0],m[1],m[2],m[3],m[4],m[5],m[6],m[7],m[8],m[9],m[10],m[11],m
[12],m[13],m[14],m[15]);
		switch(size){
			case 0:
				LINK_XFORM(*pk,order,form);
				break;
			case 1:
				LINK_XFORM(*pk,order,form,m[0]);
				break;
			case 3:
				LINK_XFORM(*pk,order,form,m[0],m[1],m[2]);
				break;
			case 4:
				LINK_XFORM(*pk,order,form,m[0],m[1],m[2],m[3]);
				break;
			case 16:
		//
		LINK_XFORM(*pk,POST,MATRIX,m[0],m[1],m[2],m[3],m[4],m[5],m[6],m[7],m[8],m[9],m[10],m[11],m[
12],m[13],m[14],m[15]);
		//
		LINK_XFORM(*pk,242,MATRIX,m[0],m[1],m[2],m[3],m[4],m[5],m[6],m[7],m[8],m[9],m[10],m[11],m[12
],m[13],m[14],m[15]);

		LINK_XFORM(*pk,order,MATRIX,m[0],m[1],m[2],m[3],m[4],m[5],m[6],m[7],m[8],m[9],m[10],m[11],m[
12],m[13],m[14],m[15]);
		}
		return 10;
}

int onoff(int pos){
		if(scanf("ON")) return 1;
		SEEK(pos);
		if(scanf("OFF")) return 0;
		error(pos);
		return -1;
}

int concave(int siori){
		int rc;
		SEEK(siori);
		scanf(" CONCAVE(");
		rc = onoff(SIORI());
		if(rc){
			CONCAVE_STATUS = 1;
			printf("CONCAVE(ON);¥n");
			scanf(");");
			return 1;
		}else{
```

```
                        CONCAVE_STATUS = 0;
                        printf("CONCAVE(OFF);¥n");
                        scanf(");");
                        return 1;
        }
        return 0;
}

int sub(){
        int siori;
        siori=SIORI();
        if(scanf(" #")) return commentout(siori);
        if(scanf(" COMMENT")) return comment(siori);
        if(COMMENT_STATUS) return commentout(siori);
        if(scanf(" FACE_NORMAL(%*[^,],%*[^)]);")) return face_normal(siori);
        if(scanf(" FACE_COLOR(%*[^,],%*[^)]);")) return face_color(siori);
        if(scanf(" FACE_MATERIAL(%*[^,],%*[^)]);")) return face_material(siori);
        if(scanf(" GROUP_FACE")) return group_face(siori);
        if(scanf(" GROUP_LINE")) return group_line(siori);
        if(scanf(" GROUP_MATERIAL")) return group_material(siori);
        if(scanf(" LINK_XFORM")) return link_xform(siori);
        if(scanf(" CONCAVE")) return concave(siori);
/*state,emt型のコマンドを全て網羅しておかないと、以下の処理でラベルとして認識される*/
    if(scanf(" %*[^=] = NORMAL(%*f,%*f,%*f);")) return normal(siori);
        if(scanf(" %*[^=] = COLOR(%*f,%*f,%*f,")) return color(siori);
        if(scanf(" %*[^=] = MATERIAL(")) return material(siori);
        if(scanf(" %*[^= ] = GROUP();")) return group(siori);
        if(scanf(" %*[^= ] = GROUP(%*[^)]);")) return group1(siori);//とりあえず
        if(scanf(" %*[^=] = COORD(%*f,%*f,%*f);")) return coord(siori);
        if(scanf(" %*[^=] = VERTEX(%*[^)]);")) return vertex(siori);
        if(scanf(" %*[^=] = FACE(%*[^)]);")) return face(siori);
        if(scanf(" %*[^=] = LINE(%*[^)]);")) return line(siori);
        if(scanf(" %*[^ =] = LINK")) return link(siori);
        if(scanf(" %*s")) return error(siori);
        logf("#EOF¥n");
        return 0;
}

int main(){
        DEBUG = 1;
        RESULT=1;
        version();
        while(sub());
        return RESULT;
}
```

（5）ＰＯＩＮＴ　ＣＬＯＵＤ（ＣＳＶ）形式

ポイントクラウド形式は、レーザースキャナで三次元形状を計測することにより作成された表面形状のデータで、データ形式は単純な行の繰り返しである。2004 年頃には、主に航空機から計測した地形・建物・樹冠のデータが使用されていたが、2014 年時点では、車載器(MMS)を用いた地上からの町並計測データも広く使用されるようになった。

```
21480.277,14905.067,27.781,47,0,3,3
21475.756,14905.546,27.782,20,0,2,2
21480.999,14903.186,27.758,28,0,2,2
```

データは、Ｘ座標、Ｙ座標、Ｚ座標と、信号強度などのデータに関するフラグが続く。１行が１点に相当する。

本データの場合、車載型のスキャナー（MMS）を用い、走行しながら沿道の建物と樹木からの反射波を記録している。

建物の外壁からの反射波の反射点の座標は、連続した、ほぼ１直線上のスキャンラインとして、建物外壁上のハッチラインのように追っている。一方、樹木から反射してくる波は、ランダムな位置座標を返している。そこで、ランダムな反射を避け、壁面をスキャンしている点群を折れ線のグループとして格納している。

更に、点の数が非常に多いことも特徴であり、処理に時間がかかることから、プログレス・インジケータに進捗状況を表示している。

メタファイルのプログラミング上の工夫としては、内部バッファがオーバーフローすることを防止するために、処理が終了したデータは出力した後にコンバータに割り当てられたメモリ上に残留しないように、出力する折れ線群のバッファは、繰り返し使用し、折れ線の数が、それまでに出力した最大数を超えた場合にのみ、バッファを伸長するようにしている。

リスト 4-3-5 ポイントクラウド形式のメタファイル

```
int nerror;
int G0,G1,P0,P1,V0,V1,L1;
quat Qpre;
int Count,Cpre,Pcount,Jcount;
int Warna,Wpre;//反射パルス数による建物と樹木の判定用
int Llen,Llenst;

quat getq(int i){
          int n;
          float x,y,z;
          n  = scanf("%f ,",x);
          n += scanf("%f ,",y);
          n += scanf("%f ,",z);
          n += scanf("%*d ,"); //intensity
          n += scanf("%*d ,"); //unknown factor;
          n += scanf("%d ,", npulse);
          n += scanf("%d ¥n", ipulse);
          if(n != 7){
                    nerror = 1;
                    return _Q(0.0,0.0,0.0,0.0);
          }
          if(npulse + ipulse == 2) Warna = 0;
          //反射パルス数が１で、その内の最初のパルスなら建物と判定
          else Warna = 1;//それ以外なら、おそらく樹木
}

void template(){
  G0 = GROUP();
  G1 = GROUP();
  P0=COORD(0,0,0);
  P1=COORD(0,0,1);
  V0=VERTEX(P0);
  V1=VERTEX(P1);
  L1=LINE(V0,V1);
}

int isqnol(quat q){
          if(_Qt(q) != 0.0) return 0;
          if(_Qx(q) != 0.0) return 0;
          if(_Qy(q) != 0.0) return 0;
          if(_Qz(q) != 0.0) return 0;
          return 1;
```

```
}

float Diff(quat q1, quat q2){
          quat q;
          float t;
          q = q2 - q1;
          q = q * q;
          t = _Qt(q);
          return -t;
}

void line_to(quat q){
          if( isqnol(Qpre) ){//リセット
                    Qpre = q;
                    logf("qnol¥n");
                    return;
          }
          if(diff(Qpre,q)<0.25){//２点間距離が0.5m未満（自乗が0.25未満）なら同一壁面と判定
            if(Count == (Cpre+1)){//ユニークな座標点総数が一つ増加することをもって順調に継続と判定
                    P0 = P1;//前の線分を起点とする
                    Llen ++;
            }else{//同一点に停留？または、離れた点に飛躍？
                    P0 = COORD(Qpre); //離れた別の線分の始まり
                    if(Llenst<Llen) Llenst = Llen;//最長の線分を統計
                    Llen = 0;
            }
            P1 = coord(q);
            if(0 == Wpre + Warna){
              if(Llen % 3 == 0) GROUP_LINE(G1,L1);//３区間毎に破線点線を１本描画する。
            }
            Cpre = Count;
          }else Jcount++;
          Qpre = q;
          Wpre = Warna;
}

int main(){
  int i,n1,n2,n3;
  quat q;
  nerror = 0;
  template();
  for(i=0;;i++){
    q = getq(i);
    if(nerror) break;
          scanf("%*[^¥n] ¥n");
//        logf("#i=%d¥n",i);
          line_to(q);
  }
  logf("#count=%d¥n",i);
  logf("#x[%f",xmin);
  logf("-%f¥n",xmax);
  logf("#y[%f",ymin);
  logf("-%f¥n",ymax);
  logf("#z[%f",zmin);
  logf("-%f¥n",zmax);
//GROUP_LINE(G1,L1);
}
```

このメタファイルを処理する利活用処理系においては､厖大な頂点座標を受ける COORD 関数の処理において、以下のような工夫を行っている。

①VC-1C

CAD データ等で、同一座標値をソーティングし、COORD 関数の引数として繰り返し渡された同一頂点座標については、同一の ID 値を返すようにしてある。この方法は、同一座標値が繰り返される場合に、メモリ使用量を節約する効果がある。しかし同一座標値がほとんど繰り返されることがない点群データの場合、この頂点座標の比較処理、およびバッファリングは逆に厖大なデータを扱うためのメモリ使用と計算時間の制約要因となる。

そこで、VC-1C においては、1000 の座標値まではソーティングを行い、同一座標値が繰り返し入力された率を計算している。その率が１％以下であれば、点群などのランダムな座標値を有するデータと判定し、以後のソーティングは行わない。

②VC-2V, VC-3M

COORD 関数で渡される座標値を、ソーティングを行った上で、メモリ上のバッファに蓄積し、GROUP_FACE 関数が実行された時点での座標値を処理系で使用するように処理している。このバッファは、大きくなるとソーティングおよびバッファサイズの再調整(Realloc)処理で時間を消費するようになる。そこで、座標値入力数が 10000 を超えた段階で、同一座標値の再現率が１％未満である場合、ランダムな座標値のデータと判定する。以後は、GROUP_FACE 関数が実行されるたびに、バッファを再初期化する。上記の例のように、２つの座標値だけを有する線分の集合体として表示するような場合には、バッファの長さは高々２あれば十分である。

（６）ＳＴＬ形式

ＳＴＬ形式(Stereo Lithography)は、三次元プリンタ等に入力するために、閉じた立体の表面を三角形の集合体として表現するためによく使用されるデータ形式である。バイナリとテキストの二種類があり、内部のフォーマットの思想は少し異なっている。形式的には四角形以上の多角形や穴あき図形も表現可能な形式となっているが、慣習的に三角形で構成されたデータだけが扱われている。

①資料等

The StL Format Standard Data Format for Fabbers Reprinted from Section 6.5 of Automated Fabrication by Marshall Burns, Ph.D., Technical source:

StereoLithography Interface Specification, 3D Systems, Inc., October 1989

バイナリ形式と、テキスト形式があり、使用する文字コードが異なるのみならず、データ構造も若干異なっている。

１）バイナリ形式

８０バイトのヘッダ部分

COLOR

MATERIAL

等の補足情報が含まれている。

本体

ascii テキスト形式のラベル「SIZ」に続く４バイトの整数１個（面総数）

SIZ で指定された回数の、面を単位とする繰り返し

面

法線　４バイトの浮動小数バイナリが３個

頂点　４バイトの浮動少数バイナリ３個から成る頂点座標が３個

色彩　２バイトの整数バイナリ１個に、パックされている

表 4-3-2　５ビット単位の R,G,B 値と、１ビットのフラグ

１ビット	５ビット	５ビット	５ビット
フラグ	B 値	G 値	R 値

２）テキスト形式

ヘッダ部分はなく、冒頭はキーワード「solid」である。

solid 行から endsolid 行までの間が一つの立体である。

facet 行から endfacet 行までの間が一つの面である。

out loop 行から endloop 行までの間が一つの外周である。

（この文法は、inner loop（穴抜き）の記述も可能であるが用いられていない）

vertex X 座標　Y 座標　Z 座標

の行が、一つの頂点を定義している。

頂点座標の行を４以上並べて多角形を表現することも可能な形式であるが、流通しているデータは３行（三角形）のみである。

リスト 4-3-6 STL テキスト形式の例

```
solid

facet normal -1.000000 0.000000 0.000000
outer loop
vertex 11810.000000 -5199.780000 5481.950000
vertex 11810.000000 -3218.780000 5337.950000
vertex 11810.000000 -5199.780000 5337.950000
endloop
endfacet

facet normal -1.000000 0.000000 0.000000
outer loop
vertex 11810.000000 -3218.780000 5337.950000
vertex 11810.000000 -5199.780000 5481.950000
vertex 11810.000000 -3218.780000 5481.950000
endloop
endfacet

facet normal 0.000000 -1.000000 0.000000
outer loop
vertex 12102.000000 -5199.780000 5481.950000
vertex 11810.000000 -5199.780000 5337.950000
vertex 12102.000000 -5199.780000 5337.950000
endloop
endfacet
 ・・・・・・・・・・・・・（中略）・・・・・・・・・・・・・
facet normal 0.000000 1.000000 0.000000
outer loop
vertex -15447.500000 -10854.800000 14141.900000
vertex -16488.500000 -10854.800000 14891.900000
vertex -15447.500000 -10854.800000 14891.900000
```

```
endloop
endfacet
endsolid
```

②メタファイル

1) 景観シミュレータのための外部関数として実装したコンバータ

メタファイル作成に先行して作成した、景観シミュレータの外部関数形式のコンバータ入力部分は以下の通りである。実際のデータが参考資料等通りにできているか、あるいはどのようなコマンドが実際に使用されているのかを確認するためには、充実したデバッグ環境でコンバータを作成してみることが能率的である。

リスト 4-3-7 STL 形式を入力する外部関数コンバータ

```
/*************************************************************
 STL2LSS.cpp
 An external function for sim.exe :
 to convert STL file into LSS-G format.
  Copyright(c) NILIM 2011-2012
  Created by DR. H. Kobayashi
*************************************************************/

// stl2lss.cpp : コンソールアプリケーション用のエントリポイントの定義
// デバッグ用コマンドラインの例：
//c:¥@keikan¥ksim¥temp¥110530_kimidori_ascii.stl c:¥@keikan¥kdb¥geometry¥@stlsample.geo
//c:¥@keikan¥ksim¥temp¥@@@110706.stl c:¥@keikan¥kdb¥geometry¥@stlsample2.geo
//c:¥@keikan¥kdb¥@samples¥110530STLサンプル¥110530_blue_ascii.stl c:¥@keikan¥kdb¥geometry¥@stlsample3.geo
/*
STL File の構造：
solid
[任意回数のfacet]
endsolid

facet normal %f %f %f
outer loop
vertex %f %f %f
vertex %f %f %f
vertex %f %f %f
endloop
endfacet

solid の表面を構成する面群記述する。solid は一つのみ。
構造上は、多角形や穴あきポリゴンを記述することも可能だが、
実用上は、三角形のみ、穴なしのポリゴン(outer loopのみ) に用いられている。

*/

#include "stdafx.h"
#include <stdlib.h>
#include <math.h>
#include <string.h>
int OUTFORM;//0:line 1:plane
FILE *WP,*RP;
char buf[81];

void SALAH( char *s ){
          printf( "¥n¥nERROR(%s)¥n", s );
          getchar();
}
```

```
void HABIS( int n ){
          exit( n );
}

char* l(){
          int i;
          char c;
          memset(buf,0,81);
          for(i=0;1;i++){
                    if(feof(RP)) break;
                    c = fgetc(RP);
                    if(c==10) break;
                    if(c==13) break;
                    if(80<i) break;
                    buf[i] = c;
          }
          return buf;
}

int cari(char* s){//文字列が現れるまでファイルを読む
          char c;
          for(;;){
                    if(!s){
                              if(1 == sscanf_s(buf,"%c",&c))
                                        return 1;
                    }else{
                              if(strstr(buf,s))
                                        return 1;
                    }
                    if( feof(RP) ) return 0;
                    l();
          }
}

int loop(){
          double x1,x2,x3,y1,y2,y3,z1,z2,z3;
          int hasil;
          if(!cari("vertex")) return 0;
          hasil = sscanf_s(buf,"vertex %lf %lf %lf",&x1,&y1,&z1);
          hasil += sscanf_s(l(),"vertex %lf %lf %lf",&x2,&y2,&z2);
          hasil += sscanf_s(l(),"vertex %lf %lf %lf",&x3,&y3,&z3);
          if(!cari("endloop")) return 0;
          if(hasil == 9){
                    fprintf(WP,"P1=COORD(%g,%g,%g);¥n",x1,y1,z1);
                    fprintf(WP,"P2=COORD(%g,%g,%g);¥n",x2,y2,z2);
                    fprintf(WP,"P3=COORD(%g,%g,%g);¥n",x3,y3,z3);
                    fprintf(WP,"GROUP_FACE(STL,F);¥n");
                    return 1;
          }
          return 0;
}

int facet(){
          double nx,ny,nz;
          const char *siori;
          int hasil;
          if(!cari("normal")) return 0;
          siori = strstr(buf,"normal");
          if(siori)
                    hasil = sscanf_s(siori+6,"%lf %lf %lf",  &nx,&ny,&nz);
          if(!cari("outer loop")) return 0;//一つしかない入れ子
          if(!loop()) return 0;
          if(!cari("endloop")) return 0;
```

```
	if(!cari("endfacet")) return 0;
	l();
	cari(NULL);
	return 1;
}

int skip(){
	char c;
	for(;;){
		if( feof(RP) ) return 0;
		if( 1 == sscanf_s(buf,"%c", &c )) break;
		l();
	}
	return 1;
}

int solid(){
	rewind(RP);
	l();
	if(!cari("solid")) return 0;
	//次に来るコマンドはfacetかendsolidである
	l();
	while( cari(NULL)){
		if(strstr(buf,"facet"))
			facet();
		else if(strstr(buf,"endsolid"))
			return 1;
		else
			return 0;
	}
	return 0; //eof
}

int header(){
	char *C;//,*M;
	fread(buf,80,1,RP);
	buf[80]=0;
	if(strstr(buf,"solid")==buf) return 0;
	else{
		if(C=strstr(buf,"COLOR=")){
			float R,G,B,A;
			float K = (float)(1.0/255.0);
			R = (unsigned char)C[6];
			G = (unsigned char)C[7];
			B = (unsigned char)C[8];
			A = (unsigned char)C[9];
			fprintf(WP,"C=COLOR(%f,%f,%f,%f);¥n",R*K,G*K,B*K,A*K);
			fprintf(WP,"FACE_COLOR(F,C);¥n");
		}
	}
	return 1;
}

void showAC(unsigned short AC){
	int R,G,B;
	int FLAG;
	printf("AC=%x¥n",AC);
	R=AC&31;
	AC>>=5;
	G=AC&31;
	AC>>=5;
	B=AC&31;
	AC>>=5;
```

```
	FLAG=AC&1;
	if(FLAG) printf("RGB no¥n");
	else printf("RGB=(%d,%d,%d)¥n",R,G,B);
}

int binary(){
	unsigned int ip;
	unsigned short AC;//attribute byte count
	long i,SIZ;
	float x,y,z;
	size_t hasil;
	fread(&SIZ,4,1,RP);
	printf("SIZE=%ld¥n",SIZ);
	for(i=0;i<SIZ;i++){
		hasil  = fread(&x,4,1,RP);//normal;
		hasil += fread(&y,4,1,RP);
		hasil += fread(&z,4,1,RP);
		fprintf(WP,"N=NORMAL(%g,%g,%g);¥n",x,y,z);
		printf("N(%g,%g,%g)¥n",x,y,z);
		for(ip=1;ip<4;ip++){
			hasil += fread(&x,4,1,RP);
			hasil += fread(&y,4,1,RP);
			hasil += fread(&z,4,1,RP);
			printf("(%g,%g,%g)",x,y,z);
			fprintf(WP,"P%d=COORD(%g,%g,%g);¥n",ip,x,y,z);
		}
		printf("¥n");
		hasil += fread(&AC,2,1,RP);
		if(AC != 65535)
			break;
		//printf("AC=%d¥n",AC);
		showAC(AC);
		if(hasil < 13){
			fprintf(WP,"#Unexpected EOF at %d of %d¥n",i,SIZ);
			printf("unexpected EOF at %d of %d¥n",i,SIZ);
			break;
		}
		fprintf(WP,"GROUP_FACE(STL,F);¥n");
	}
	if( i == SIZ ) return 1;
	return 0;
}

int main(int argc, char* argv[])
{
	if( argc<3){
		SALAH( "引数不足" );
		HABIS(2);
	}
	if(!argv[1]){
		SALAH( "引数：変換元処理対象案件名称未指定" );
		HABIS(2);
	}
	if( 250 < strlen(argv[1]) ){
		SALAH("入力ファイルが長すぎ");
		HABIS(2);
	}
	if(!argv[2]){
		SALAH( "出力先未指定" );
		HABIS(2);
	}
	if( 3 < argc ){
		sscanf_s(argv[3],"%d",&OUTFORM);
```

```
        }else{
                OUTFORM = 0;
        }
        int retval;
        fopen_s(&RP,argv[1],"rb");
        if(!RP){
                SALAH( "入力元ファイル開かず" );
                HABIS(4);
        }
        fopen_s(&WP,argv[2],"wt");
        if(!WP){
                SALAH( "出力先ファイル開かず" );
                HABIS(3);
        }
        /*ヘッダ部分の出力*/
        fprintf(WP,"#STL2LSS ver 0.0¥n");
        fprintf(WP,"¥tSTL = GROUP();¥n");
        fprintf(WP,"¥tN=NORMAL(0,0,1);¥n");
        fprintf(WP,"¥tP1=COORD(0,0,0);¥n");
        fprintf(WP,"¥tP2=COORD(0,0,0);¥n");
        fprintf(WP,"¥tP3=COORD(0,0,0);¥n");
        fprintf(WP,"¥tV1=VERTEX(P1);¥n");
        fprintf(WP,"¥tV2=VERTEX(P2);¥n");
        fprintf(WP,"¥tV3=VERTEX(P3);¥n");
        fprintf(WP,"¥tF=FACE(V1,V2,V3);¥n");
        /*ヘッダ部分終了*/
        if(header()) retval = binary();
        else{
                freopen_s(&RP,argv[1],"rt",RP);
                if(!RP){
                        SALAH( "入力元ファイル開かず" );
                        HABIS(4);
                }
                retval = solid();/*solid は一つのみ*/
        }
        if(RP) fclose(RP);
        if(WP) fclose(WP);
        return retval;
}
/*-----------------------------*
戻り値return またはexit
0:未定義のエラー
1:正常終了
2:引数の数が合わない
3:出力ファイルが開かない
*-----------------------------*/
```

２）仮想コンバータのためのメタファイル

メタファイルをリスト 4-3-8 に示す。１）と比較して、コンパクトに記述している。header() 関数で最初の１行を読み込み、”solid”の文字列があればテキスト形式としてデータ読み込みを開始し、無ければバイナリ形式として、ヘッダ部 80 バイトを読み込んでいる。

リスト 4-3-8 STL テキスト形式のメタファイル

```
/*stl.cmm*/

int P[3],V3[3],F3,ROOT,N3,C3;

int header(){
        int r,g,b,a;
```

```
          if(scanf("solid")) return 0;
//        logf("binary¥n");
          if(scanf("COLOR=")){
            scanf("%c",r);
            scanf("%c",g);
            scanf("%c",b);
            scanf("%c",a);
            C3 = COLOR(r,g,b,a);
          }else exit(10);
          return 1;
}

float readF(){
          float f;
          scanf("%4c",f);
          return f;
}

void bcolor(){//バイナリ形式カラー入力
          float r,g,b;
          int ac,flag;

          scanf("%2c",ac);
          r = ac % 32;//５ビット単位に分解
          ac /= 32;
          g = ac % 32;
          ac /= 32;
          b = ac % 32;
          ac /= 32;
          flag = ac % 1;
          C3 = COLOR(r,g,b);
          logf("bcolor(%d)",ac);
          logf("=(%f",r/31.0);
          logf(",%f",g/31.0);
          logf(",%f",b/31.0);
          logf(",%d)¥n",flag);
}

void face(){
          int ip,AC;
    N3 = NORMAL(readF(),readF(),readF());
    for(ip=0;ip<3;ip++){
                    P[ip]=COORD(readF(),readF(),readF());
          }
          bcolor();//中でCOLORコマンドを実行
          GROUP_FACE(ROOT,F3);
}

void template(){
          P[0] = COORD(0,0,0);
          P[1] = COORD(0,0,0);
          P[2] = COORD(0,0,0);
          V3[0] = VERTEX(P[0]);
          V3[1] = VERTEX(P[1]);
          V3[2] = VERTEX(P[2]);
          F3 = FACE(V3[0],V3[1],V3[2]);
          N3 = NORMAL(0,0,1);
          FACE_NORMAL(F3,N3);
          C3 = COLOR(1,0,0);
          FACE_COLOR(F3,C3);
          ROOT = GROUP();
}
```

```
void binary(){//バイナリ形式
    int SIZ,i;
    SEEK(80);
    template();
    scanf("%4c",SIZ);
    logf("SIZ=%d¥n",SIZ);
    for(i=0;i<SIZ;i++){
                    face();
          }
}

/* ascii */
void outerloop(){
    quat q;
    int i;
    for(i=0;i<3;i++){
      scanf(" vertex %qx %qy %qz",q);
      P[i] = COORD(q);
    }
    if(!scanf(" endloop")) logf("#no endloop¥n");
}

void facet(){
   int n;
   quat normal;
   if(!scanf(" normal %qx %qy %qz",normal)) logf("#no normal¥n");
   N3 = NORMAL(_Qx(normal),_Qy(normal),_Qz(normal));
   if(scanf(" outer loop")) outerloop();
   if(!scanf(" endfacet")) logf("#no endfacet¥n");
   GROUP_FACE(ROOT,F3);
}

void solid(){
    C3 = COLOR(1,1,1);
    while(scanf(" facet")) facet();
    if(!scanf(" endsolid")) logf("#no endsolid¥n");
}

void ascii(){//ASCIIテキスト形式
    template();
    solid();
}

int main(){
          if(header()) binary();
          else ascii();
          return 1;
}
```

STL 形式のファイルの出力方法に関しては、4-2(4)④で解説した。

（７）LandXML 形式

LandXML 形式は、ＸＭＬ形式に従って、地形、市街地、インフラ等を記述するためにタグを定義したファイル形式である。LandXML 形式は、道路や堤防等の長尺物を対象として、慣習的な設計製図法における中心線軌跡、横断図、縦断図等に描かれた同一オブジェクトの図面間の対応関係を識別できるように、属性付きでベクトルデータ化したファイルである。中心線の三次元的な軌跡を作成し、これを骨格として隣接する横断面（多くの場

合 20m 間隔）をつないでいくことにより、立体的な構造物のデータを自動的に生成し、表示確認や自動運転等に活用することが容易である。紙図面による表現法との連続性を保っている点が建築分野と異なっている。

①資料等

運営組織のサイト(Landxml.org)によれば、2000 年頃から仕様が検討され、初期のスキーマが公開されている。最新のスキーマは、2014 年の Ver.2.0 案となっている。測量された土地の形状などを記述する用途が先行している。土木系 CAD ソフト等も LandXML 形式での入出力機能を装備している。

国総研では、2004 年頃の図面の電子化(SXF 形式)を踏まえて、LandXML 形式をカスタマイズして、紙図面から二次元ＣＡＤデータに継承されている慣習的な記法を記録することを可能にした。特に、道路中心線に関しては早くから仕様が決定され、国総研資料が公開された。更に、断面形状を含む立体的に表示可能な仕様が検討され、2014 年時点ではいくつかのサンプルデータが作成された。これらに基づき「(案)に準じた LandXML1.2 拡張(案)-国総研 NILIM」が作成され、国総研のサイトから公開されている。

本節で解説するメタファイルは、このサンプルデータを可視化し確認するための限定された目的のものであり、様々な属性情報は、表示には反映していない。しかし、ジオメトリの記述に使用した座標系や長さ等の単位などについては解析に含めており、アーカイブとして意味のある属性情報の読み出し処理を今後更に追加するための参考になると考える。

②データ構造

LandXML 形式で記述された道路や河川堤防等の構造物のデータは、CG 等の目的に作成された各種の三次元データ形式とは異なり、ベクトル化された図面情報であって、紙図面により仕事が行われていた業務形態の慣習を基本的に継承している。

しかし単に図面をスキャンしたデータや、プロッタによる図面出力を目的とした CAD データとは異なる特徴として、制御点等の名称を通じて、断面図と中心線の軌跡等の異なる図葉の対応関係を自動的に認識し、立体を構成することが容易化されている点がある。

一つの路線の単位が Alignment タグにより記述されている。これは例えば交差点から交差点までの区間である。Alignments タグの中に複数の Alignment が束ねられて一つの LandXML ファイルとなっている。現在までに扱った LandXML 形式のファイルは、検討、開発、普及等を目的としたサンプル的なものであるため、Alignment を単位として解読を行うのが便利である。

一つの路線(Alignment)は、平面形(CoordGeom)、縦断線形（Profile）、および横断図（CrossSect）の集合(CrossSects)により表現されており、これに属性情報(Feature)が情報として付加されている。

平面形は、直線区間(Line)、クロソイド区間(Spiral)、円弧区間(Curve)の連鎖として表現される。それぞれの区間は起点と終点のキロ程、XY 座標に、曲率中心等のパラメータを持つ。縦断線形は、勾配が一定の区間を単位として、その開始点と終了点のキロ程と標高が

リストされている。キロ程は、平面形における起点からの累積距離であり、斜面における立体的な距離ではない。勾配が変化する点の位置は、平面形における区間分割と一致するわけではない。横断面は、多くの場合２０m間隔で、断面を構成する線分（X-Z 座標）とその属性（法面、舗装面等）が記述されている。断面は、道路の平面系における接線と垂直の面の中での二次元図形である。

2017 年に作成されたサンプルデータにおいては、この紙図面の形式に由来するデータ構造に加えて、立体的なモデルを確認するためのサーフェスモデルが追加されている。

データの本体は、<LandXML> ~</LandXML>の間に記述されており、ファイル全体に関する様々な属性（例えば設計者、使用する単位など）を除くと、本質的な各オブジェクトは、<Alignments>～</Alignments>の中に、複数の路線が、<Alignment>各路線</Alignment>を単位として記述されている。

1）線形データ<CoordGeom>

中心線軌跡の平面図は、<Alignment>タグの中の<CoordGeom>タグにより記述される。一つの CoordGeom（路線の中心線）は、一連の接続された区間から成り立っており、サンプルデータの場合には<Line>(直線区間)、<Curve>(円弧区間)、Spiral（クロソイド区間）の３種類の区間がある。

Line は、起点と終点の座標により定義される。

Curve は、起点と終点の座標に加えて、半径の中心位置の座標が定義されている。区間の長さはこれらのパラメータから計算により求める。

Spiral は、起点と終点の座標と曲率半径、および区間の長さにより定義されている。クロソイド曲線は、曲率が測地線（キロ程）に対して線形に変化するような曲線である。超越関数等により表現することはできないため、本処理系では、初期化の段階でクロソイド曲線の値を大域配列に求めておき、個別の区間の形状の算出に際してはスケールを掛けて配列の値を取得すると共に、補完計算を行っている。なお、曲率半径のパラメータは曲率がゼロの地点においては無限大となるため、”INF”の文字列で無限大が表現されている。

路線全体の中心線の軌跡は、キロ程を引数とする関数により表現する。この関数の内部では、キロ程からその地点が３種類の内のどの区間に該当するかを求め、その区間内のキロ程に応じて３種類の形状別に座標値を計算した上、起点と終点の座標から回転角を求めて具体的な座標値を計算する。

なお、Line、Curve および Spiral はそれぞれ異なる方法（パラメータ）を用いて定義されているが、幾何形状としては、Curve は、Spiral の特殊な場合（起点と終点の曲率半径が一数）であり、更に Line は Curve の特殊な場合（起点と終点の曲率半径が共に無限大）であるから、いずれも Spiral のパラメータを用いて表現することが可能である。

図 4-3- に、直線―クロソイド－円弧－クロソイド－円弧と変化する中心線の区間を返還した例を示す。

図 4-3-5 クロソイド区間を含む中心線軌跡の入力結果

2) 縦断線形データ <Profile>と<ProfAlign>

紙図面における縦断図に相当するデータが、Alignment タグ(路線)毎に、Profile タグ(縦断線形)とその下の ProfAlign(断面路線)として定義されている。平面図データと縦断図データを組み合わせることにより、立体的な中心線軌跡を完全に求めることができる。

折れ線で記述した縦断線形は PVI タグ（縦断勾配変移点）と、勾配が変化する中間点の ParaCurve タグ（縦断曲線）で記述されている。通常は、勾配が変化する中間点の前後に緩和区間を設けて、滑らかに勾配が変化するように設計される。この緩和区間の長さ（縦断曲線長）が、length 属性で記述されている。緩和区間の形状は、LandXML.org による仕様 1.2 では放物線(a parabolic vertical curve)とされているが、国総研仕様 1.2 では「縦断曲率半径」の用語があり、円弧とされている。本メタファイルでは、後者に従う。

3）横断図データ　＜CrossSects>と<CrossSect>

紙図面における横断図に相当するデータが、Alignment タグ(路線)毎に、CrossSects（横断図群）とその下の複数の CrossSect(横断面)として定義されている。一つの横断面には、その位置を定義するキロ程が属性 sta として記述され、更に個々の断面を構成する線分群が DesignCrossSectSur タグ（複数）で記述され、起点と終点の名称と座標値が記述されている。水平位置を示す座標値は 0 以上の正数であり、別途右か左が指定されている。垂直位置を示す値は、中心線の高さとは独立した標高の絶対値となっている。立体形状を生成するためには、それぞれの横断面を、平面図と縦断図から求めた立体的な中心線軌跡のキロ程の位置に、進行方向と垂直の向きに配置し、隣接する横断面の対応する点を結ぶことにより、路面や盛り土面を生成した。

③メタファイル

1)景観シミュレータのための外部関数（LandXML.cpp）

なお、LandXML 形式のサンプルデータを読み込み、表示・編集する入力用コンバータとして、景観シミュレータの外部関数 LandXML.exe が利用できるので、本処理系の上でメタファイルを用いてサンプルデータが正常に読み込まれた場合の表示状態を比較参照することができる。このソースコード（LandXML.cpp）をリスト 4-3-9 に示す。特徴として、道路中心線の軌跡を記述するために多用されているクロソイド曲線をその都度計算するのではなく、あらかじめ固定長配列に関数を変換表として格納しておき、プログラムからはテーブル参照と補間処理によって高速処理している。また、実質的な意味のある XML タグに関しては、タグと同名の関数を用いて処理している。

一方、仮想コンバータのメタファイル（リスト 4-3-10）と比較すると、出力処理において全て fprintf(wp,”・・・・)という処理を行っているため、読みにくく長いプログラムとなっている。

リスト 4-3-9 外部関数 LandXML.cpp 抜粋

```
/*********************************************************/
/* LandXML.cpp                                           */
/* external function for LSS sim.exe                     */
/*                                        */
/*  Copyright(c) National Institute for                  */
/*    Land and Infrastructure Management, 2006           */
/*  Created by Dr. KOBAYASHI, Hideyuki 2006.12.05        */
/*********************************************************/
// LandXML.cpp : コンソール アプリケーション用のエントリ ポイントの定義
（中略）
*/

#include "stdafx.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <errno.h>
```

```
#include <math.h>
#include <process.h>
#if defined (_WIN32)
#include <windows.h>
#else  /* defined (_WIN32) */
#include <fltintrn.h>
#endif  /* defined (_WIN32) */
#include <atlstr.h> /*140208*/

void SALAH(char*);
void AWAS(char*);
void HABIS(int );

/****************** STRUCT DEFINITIONS ****************/
typedef struct {
          double X;
          double Y;
}XY;

typedef struct {
          double X;
          double Y;
          double Z;
}XYZ;

typedef struct {
          double X;
          double Y;
          double Z;
          double T;
}XYZT;

typedef struct {
          double R;
          double G;
          double B;
          double A;
}RGBA;

typedef struct {
          char *name;
          char *code;
          char *desc;
          char *featureRef;
          double x;
          double y;
          double z;
}CGPOINT;

typedef struct{
          float length;
          float staStart;
          float altitude;
          int TYPE; //区間のタイプ(直線、円弧、クロソイド)
          void* p; //データブロックへのポインタ(LINE,CURVE,SPIRAL,PVI,ParaCurve)
}ROUTE;

typedef struct{
          float sta;//キロ程
          char *name;//断面名称("No.8"等)
          int np;
          XY *POINTS;
          char **names; //構成点の名称
```

```
            int nl; //線分の数
            int *head;//始点リスト：構成点の番号を参照する
            int *tail;//終点リスト
            char **facename; //面の名称リスト
}CROSS;

typedef struct{
            char *name;
            char *desc;
            float length; //区間毎の長さ
            float staStart;
            int NH; //水平区間の総数
            ROUTE*HR; //水平区間リスト
            int NV; //垂直区間の総数
            ROUTE *VR; //垂直区間リスト
            int no; //軌道の頂点数
            XYZ *orbit; //軌道を構成する頂点
            int *type;//起動を構成する頂点の種類（直線、円弧、クロソイド）
            int NC; //横断図の数
            CROSS*CR; //横断図リスト
}ALIGNMENT;//路線

/************** GLOBAL VARIABLES ******************/
//char *PATH, path[256];//入力フォルダと、フルパス入力ファイル
char *path, SHOWERROR[1000] = "exit";
FILE *LAPOR=NULL;
//int lc;
long lc;
//void * memlist[100000];
//void **memlist=NULL;
//int memcount;
int mcount; /*Malloc count*/
int NERROR = 0; //020320 DR.H.K.
//int DIRTY = 0;
int NSF = 0;
#define D20 20 //断面間隔
#define JARAK 7 //記憶間隔
//#define D20 2
//140117
int params, param1, param2;
//140208, -1:unknown, 0:shift-jis, 1:utf-8
int charcode;//ファイル先頭コードによる判定
int encoding;//<?xml タグ内部でのencoding= 記述
char *projname; //子ファイルのプレフィックスに使用
char *projpath;

CGPOINT *CGPOINTS;
int NCG;

ALIGNMENT **R;//データ中の路線リスト(水平区間と垂直区間)
int NR;//路線の数
XY* CLOTHOID;
/*   (中略)  */

XY* CreateClothoidArray(){
/*1000mの区間、１m毎にクロソイド曲線の座標を求め配列に格納する。
  個別のクロソイドは、配列を参照して求める。
  起点を中心に回転させて、実際の座標を求める。
  50m進んだところで、R=200mとする。
  1m進むと、曲率が1/10000 増える。
  50m地点で、曲率は、1/200
  X軸方向から出発して、Y軸方向に曲がる
*/
```

```
	int i;
	XY *clothoid,v1,v2;
	double t,s,c;
//	FILE *f;

	clothoid = (XY*) Calloc( 1000, sizeof(XY) );
	clothoid[0].X = 0.0;
	clothoid[0].Y = 0.0;
	clothoid[1].X = cos(0.5/10000.0);
	clothoid[1].Y = sin(0.5/10000.0);
	for(i=1;i<999;i++){
		t = ((double)(i) + 0.5)/10000.0;
		v1.X = clothoid[i].X - clothoid[i-1].X;
		v1.Y = clothoid[i].Y - clothoid[i-1].Y;
		s = sin(t);
		c = cos(t);
		v2.X = v1.X * c - v1.Y * s;
		v2.Y = v1.X * s + v1.Y * c;
		clothoid[i+1].X = clothoid[i].X + v2.X;
		clothoid[i+1].Y = clothoid[i].Y + v2.Y;
	}
	return clothoid;
}

XY GetClothoidPoint(double llocal, double length, double R, int rot){
	/*長さL,終端Rのクロソイドのllocal地点での座標を求める*/
	XY P = {0.0,0.0};
	int i;
	double d;
	double K; //スケール・ファクタ（ハンドル操作の忙しさに比例）

	if( length < 0.1 || R < 0.1 ){
		SALAH("GetClothoidPointの引数length");
		return P;
	}
	K = sqrt(10000.0 / length / R);
	i = (int) (llocal * K);
	d = llocal * K - (double)i;
	P.X = CLOTHOID[i].X * (1.0-d) + CLOTHOID[i+1].X * d;
	P.Y = CLOTHOID[i].Y * (1.0-d) + CLOTHOID[i+1].Y * d;
	P.X /= K, P.Y /= K;//相似形を保つ
	if(rot == 1) P.Y = -P.Y; //cw 時計周りなら負
	P.Y = -P.Y; //cw 時計周りなら負
	return P;
}

XY locS(void*P, float llocal){//緩和曲線
	/*Start, End, PI, rotだけから曲線形を求める*/
	XY loc, clot, v1, v2;
	SPIRAL *S;
	int dir;
	double d1, d2, t1, t2, s, c;

	S = (SPIRAL*)P;
//	index = (int)(llocal/20.0);
	if(!CLOTHOID) CLOTHOID = CreateClothoidArray();

	v1.X = S->PI.X - S->Start.X;
	v1.Y = S->PI.Y - S->Start.Y;
	v2.X = S->End.X - S->PI.X;
	v2.Y = S->End.Y - S->PI.Y;
	d1 = v1.X*v1.X + v1.Y*v1.Y;
	d2 = v2.X*v2.X + v2.Y*v2.Y;
```

```
		t1 = getangle(v1);
		t2 = getangle(v2);
		if(d1<d2){
			dir = 1;//カーブの出口
			clot = GetClothoidPoint(S->length-llocal,S->length,S->radiusStart,S->rot);
			//この点を、終点の周りに回転
			s = sin(t2);
			c = cos(t2);
			loc.X = S->End.X - clot.X * c - clot.Y * s;
			loc.Y = S->End.Y - clot.X * s + clot.Y * c;
		}else{
			dir = 0;//カーブの入口
			clot = GetClothoidPoint(llocal,S->length,S->radiusEnd,S->rot);
			//この点を始点の周りに回転
			s = sin(t1);
			c = cos(t1);
			loc.X = S->Start.X + clot.X * c - clot.Y * s;
			loc.Y = S->Start.Y + clot.X * s + clot.Y * c;
		}
		return loc;
}

XY loc(int TYPE, void*P, float llocal){//個別区間の中の位置を取得
		XY loc;
		loc.X = loc.Y = 0.0f;
		switch(TYPE){//区間の種類毎に振り分け
			case 'L': return locL(P, llocal);//直線
			case 'C': return locC(P, llocal);//円弧
			case 'S': return locS(P, llocal);//クロソイド
			default: return loc;
		}
}

/*   中略 */

PC *ParaCurve(char *S, FILE *rp, FILE *wp, int count){//PARACURVE構造体をメモリブロックで返す
		float length;
		float x,y;
		if(!findfloatoperand(S, "length", &length)) length = 0;
		PC *p;

		p = (PC*) Malloc(sizeof(PC));
		fscanf(rp, "%f %f", &x, &y);
		fprintf(wp, "PC%d=COORD(%f, 0, %f);¥n", count, x, y);
		fprintf(wp, "V%d=VERTEX(PC%d);¥n", count, count);
		FindTagName(rp, "/ParaCurve");
		/*VR配列に中間点を追加*/
		p->alongwiselocation = x;
		p->altitude = y;
		p->lengthofverticalsmoothing = length;
		return p;
}

void CalcAlignCurve(int NV, ROUTE *VR){
		int i;
		PC *P;
		double r;

		XY vec1, vec2, vec0;//前後の区画の垂直面内の方向
		for(i=0;i<NV;i++){
			if(VR[i].TYPE != 'C') continue;
			if(i<1) continue;
			if(NV-2<i) continue;
```

```
                        //ここで、円弧の中心座標、半径、等を計算し、CRのパラメータに追加する
                        P = (PC*)VR[i].p;
                        vec1.X = VR[i].staStart - VR[i-1].staStart;
                        vec1.Y = VR[i].altitude - VR[i-1].altitude;
                        vec2.X = VR[i+1].staStart - VR[i].staStart;
                        vec2.Y = VR[i+1].altitude - VR[i].altitude;
                        r = sqrt(vec1.X*vec1.X + vec1.Y*vec1.Y);
                        vec1.X /= r;
                        vec1.Y /= r;
                        r = sqrt(vec2.X*vec2.X + vec2.Y*vec2.Y);
                        vec2.X /= r;
                        vec2.Y /= r;
                        P->Theta = (float) asin(vec1.X*vec2.Y - vec1.Y*vec2.X);
                        P->R = P->lengthofverticalsmoothing / P->Theta;
                        vec0.X = vec2.X - vec1.X;
                        vec0.Y = vec2.Y - vec1.Y;
                        r = sqrt(vec0.X*vec0.X + vec0.Y*vec0.Y);
                        vec0.X /= r;
                        vec0.Y /= r;
                        vec0.X *= fabs(P->R);
                        vec0.Y *= fabs(P->R);
                        P->0.X = P->alongwiselocation + vec0.X;
                        P->0.Y = P->altitude + vec0.Y;
                        P->Rhead = (float) (P->0.X + P->R * vec1.Y);
                        P->Rtail = (float) (P->0.X + P->R * vec2.Y);
                        r = P->Rtail - P->Rhead - P->lengthofverticalsmoothing;
                }
}

void ProfAlign(char *S, FILE *rp, FILE *wp, ALIGNMENT *A){
                char *s, *name, nf[1000];
                const char *sjname; /*Shift-jis*/
                FILE *prof;
                int i,count;
                int NV;
                ROUTE *VR; //縦断路線
                name = findoperand(S, "name");
                /*140208 utf-8への対応*/
                if(charcode == 1){
                        sjname = UTF8_SJIS1(name).GetString();
                }else if(encoding == 1){
                        sjname = UTF8_SJIS1(name).GetString();
                        if(*sjname=='?')
                                sjname = A->name; //140221 苦肉の策
                }else{
                        sjname = name;
                }
                /*確認用ファイルの作成*/
                sprintf(nf, "%s-PROF-%s.geo", projname, sjname);
//              prof = fopen(nf, "wt");
                prof = wp; //現況と計画を同じファイルに、色を変えて入れる
                fprintf(prof,"#ProfAlign [%s]¥n",sjname);
                fprintf(prof,"PROF=GROUP();¥n");
                /*配列の作成*/
                NV = A->NV;
                VR = A->VR;
                /*タグ内部の処理*/
                for(count=0;;){
                        if(!VR){
                                NV = 0;
                                VR = (ROUTE*) Malloc(sizeof(ROUTE));
                        }else{
                                VR = (ROUTE*) Realloc(VR, (NV+1) * sizeof(ROUTE));
```

```
                    }
                    s = GetTag( rp );
                    if(headmatch(s, "/ProfAlign")) break;
                    else if(headmatch(s, "PVI")){
                              PVI *p;
                              p = pvi(rp, prof, count++);
                              VR[NV].staStart = p->alongwiselocation;
                              VR[NV].TYPE = 'P';
                              VR[NV].p = p; //高さ情報を含む
                              VR[NV].length = 0.0;
                              VR[NV].altitude = p->altitude;
                              NV++;
                    }else if(headmatch(s, "ParaCurve")){
                              PC *p;
//                            p = ParaCurve(s, rp, prof, count++,A);
                              p = ParaCurve(s, rp, prof, count++);
                              VR[NV].staStart = p->alongwiselocation;
                              VR[NV].TYPE = 'C';
                              VR[NV].p = p; //高さ情報を含む
                              VR[NV].length = p->lengthofverticalsmoothing;
                              VR[NV].altitude = p->altitude;
                              NV++;
                    }else SALAH(s);
          }
          /*終了処理*/
          fprintf(prof,"L=LINE(V0");
          for(i=1;i<count;i++){
                    fprintf(prof,",V%d", i);
          }
          fprintf(prof,");¥n");
          fprintf(prof,"Red=COLOR(1,0,0);¥n");
          fprintf(prof,"LINE_COLOR(L,Red);¥n");
          fprintf(prof,"GROUP_LINE(PROF,L);¥n");
//        fclose(prof);
          CalcAlignCurve(NV,VR);//円弧区間の高さ計算に必要なパラメータを算出する
          A->VR = VR;
          A->NV = NV;
          Free(name);
}

void PntList2D(FILE *rp, FILE *wp);

void ProfSurf(char *S, FILE *rp, FILE *wp){//縦断地形
          char *s, *name;
          const char *sjname;
          name = findoperand(S, "name");
//140208
          if(charcode == 1){
                    sjname = UTF8_SJIS1(name).GetString();
          }else if(encoding == 1){
                    sjname = UTF8_SJIS1(name).GetString();
                    if(*sjname=='?')
                              sjname = name; //140221 苦肉の策
          }else{
                    sjname = name;
          }
          fprintf(wp,"#PntList2D [%s]¥n",sjname);
          Free(name);
          /*140208 dsb.*/
          for(;;){
                    s = GetTag( rp );
                    if(headmatch(s, "/ProfSurf"))  break;
                    else if(headmatch(s, "PntList2D")) PntList2D(rp, wp);
```

```
                    else SALAH(s);
          }
}

void Profile(char *S, FILE *rp, FILE *wp, ALIGNMENT *A){
          char *s, *name, *staStart, nf[1000];
          const char *sjname;
          FILE *pf;

          name = findoperand(S, "name");//メモリブロック
//140208
          if(charcode == 1){
                    sjname = UTF8_SJIS1(name).GetString();
          }else if(encoding == 1){
                    sjname = UTF8_SJIS1(name).GetString();
                    if(*sjname=='?')
                              sjname = A->name; //140221 苦肉の策
          }else{
                    sjname = name;
          }

          staStart = findoperand(S, "staStart");
          sprintf(nf, "%s¥¥%s-PROF-%s.geo", projpath,projname, sjname);
          pf = fopen( nf, "wt");
          fprintf(pf,"SECT=GROUP();¥n");
          /*タグ内部の処理*/
          for(;;){
                    s = GetTag( rp );
                    if(headmatch(s, "/Profile"))    break;
                    else if(headmatch(s, "ProfAlign")) ProfAlign(s, rp, pf, A);//計画縦断：A->VRIにデータを
格納
                    else if(headmatch(s, "ProfSurf")) ProfSurf(s, rp, pf);//現況地形縦断
                    else SALAH(s);
          }
          /*終了処理*/
          Free(name);
          Free(staStart);
          fclose( pf );
}
void GetLandXml(FILE *rp, FILE *wp){
          char *s;
#if 1
          do{
                    s = GetTag(rp);
          }while ( !headmatch(s,"LandXML"));
#else
          do{
                    s = readquotedstring( rp );
          }while (!strcmp( s+1, "LandXML" ));
#endif
          for(;;){//本体の解析
                    s = GetTag( rp );
                    if(headmatch(s, "/LandXML")) break;
//                  else if(headmatch(s, "Author")) Author(s,rp,wp);
                    else if(headmatch(s, "CgPoints")) CgPoints(rp,wp);
                    else if(headmatch(s, "Project")) Project( s, rp, wp );
                    else if(headmatch(s, "Units")) Units( rp, wp );
                    else if(headmatch(s, "CoordinateSystem")) CoordinateSystem( s, wp );
                    else if(headmatch(s, "Application")) Application(s, rp, wp );
                    else if(headmatch(s, "Surfaces")) Surfaces( rp, wp );
                    else if(headmatch(s, "Alignments")) Alignments(s, rp, wp );
                    else if(headmatch(s, "Roadways")) Roadways( rp, wp );
                    else SALAH(s);
```

```
        }
//      return NULL;
}

/* 中略 */
void conv(FILE *rp, FILE *wp){
        version( rp, wp );
        Template(wp);
        GetLandXml( rp, wp );
//      return NULL;
}

int main(int argc, char* argv[])
{
        FILE *rp,*wp;
//      char* p;

        LAPOR = NULL; //020320 DR.H.K.

        if(argc<4){
                SALAH("引数不十分");
                HABIS(2);
        }
        if(!argv[1]){
                SALAH( "引数LandXML名称未指定" );
                HABIS(2);
        }
        rp = fopen(argv[1],"rt");
        if(!rp){
                SALAH( "指定LANDXML無之" );
                HABIS( 3280 );
        }else{/*140208 文字コード判定を追加*/
                charcode = getcharcode(rp);
                path = strrchr( argv[1],'¥¥');
                if(!path) path = argv[1];
                else path++; //¥¥を省く
        }
//      strcpy( path, argv[1] );
//      p = strrchr( path, '¥¥' );
        params = atoi(argv[2]);
        param1 = params & 1;
        param2 = (params & 2)/2;

        if(!argv[3]){
                SALAH("引数：出力先名未指定");
                HABIS( 20041022 );
        }else{
                projname = getprojname(argv[3]);
                projpath = getprojpath(argv[3]);
        }
        wp = fopen(argv[3],"wt");
        if(!wp){
                char s[256];
                sprintf(s,"出力ファイル[%s]開かず",argv[2]);
                SALAH(s);
                HABIS(20130726);
        }
        fprintf(wp,"#LandXML Converter for LSSG, Ver.2.0(長尺物系)¥n");
        fprintf(wp,"# LANDXML  SOURCE:[%s]¥n",argv[1]);
        memload();
        conv( rp, wp );
        fclose( rp );
        fclose( wp );
```

```
        beres();//140208 おそうじ

        if(NERROR || (0 < mcount)){//報告事項有之
                if(!LAPOR) SALAH("メモリーリーク以外のエラーなし");
                fprintf(LAPOR,"----------------------¥n");
                fprintf(LAPOR,"TOTAL ERROR = %d¥n", NERROR);
                fprintf(LAPOR,"メモリリーク数 = %d¥n",mcount);
                if(0<mcount) fprintf(LAPOR,"最初のリークブロック番号 = %d¥n",errpnt);
                fclose(LAPOR);
                system( SHOWERROR );
//              ShellExecute( NULL, _T("open"), path, NULL, path, SW_SHOW );
        }
        memsave();
        return 1;//正常終了値１
}
```

入力結果の表示例を図 4-3-6 に示す。

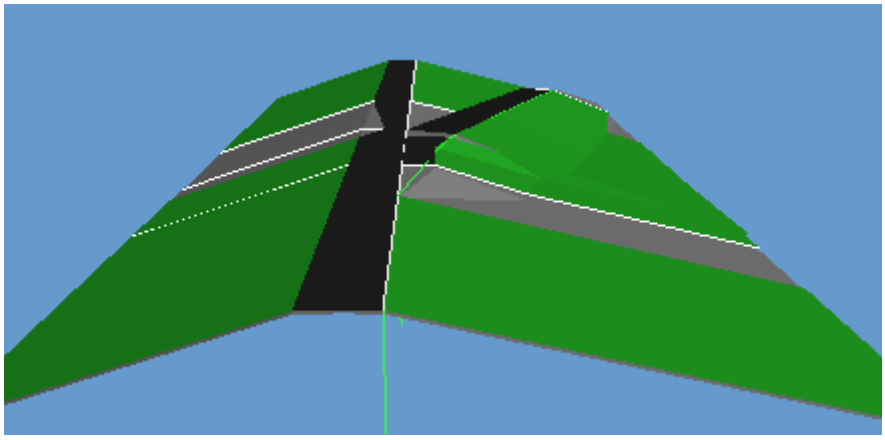


図 4-3-6 LandXML 変換結果表示例（堤防に斜路がついた図形）

2)仮想コンバータのためのメタファイル（LandXML.cmm）

解析にあたっては、Alignment 毎に平面形、縦断線形、断面図の情報を累積的に配列に取り込む。解析結果の表示方法は任意であるが、上記の景観シミュレータのための外部関数 LandXML.cpp によるコンバータ LandXML.exe と同じ方法で表現する。表示のために中心線の三次元的な軌跡と、空間的に配置した断面図、および隣接する断面図の間の対応する線分を結んだ面（法面、舗装面等）を表示する。中心線軌跡を表示するために、断面図が定義されている区間においては、断面図が定義されている位置の中心線上の点を線分で結んだ形で表現している。また、高さが明確にわかるように、この点から標高ゼロの面まで下した垂線も表示している。

外部関数 LandXML.exe は、C 言語でコーディングしたので、構造体とメモリブロックを使用しており、様々の種類・規模の LandXML ファイルを扱うことを目的としている。一方、仮想コンバータのメタファイルはよりシンプルに、解読対象とする特定のデータに関して、テストにより絞り込んだ十分な長さの固定長配列のみを用いて処理を行っている。

例えば、Alignment を 30 含むデータに関しては、グローバル変数として、

Alignment.name[30];

等を定義しておき、Alignment タグ内部の解析に際して、取得したパラメータをリスト 4-3-10 のように、パラメータ毎の配列に Alignment 番号を添字として格納する。

リスト 4-3-10 取得したパラメータの配列への格納

```
Alignment.name[NAlignment] = param6;
Alignment.length[NAlignment] = param22;
Alignment.staStart[NAlignment] = param23;
```

また、Alignmentタグ内部の、複数回繰り返されるサブタグの解析結果は別途配列に蓄積しておき、その先頭アドレスを親タグの配列に格納しておく(リスト4-3-11)。

リスト 4-3-11 サブタグ解析結果の配列への格納

```
Alignment.CoordGeom[NAlignment] = NCoordGeom;
CoordGeom[NCoordGeom++] = (解析結果)
CoordGeom[NCoordGeom++] = (解析結果)
  ・・・
Alignment.CrossSects[NAlignment] = NCrossSects;
CrossSects[NCrossSects] = (解析結果)
CrossSects[NCrossSects] = (解析結果)
CrossSects[NCrossSects] = (解析結果)
  ・・・
```

親タグ配列の差分に対応する、

「Alignment.CrossSects[N]以上、Alignment.CrossSects[N+1]未満」

のCrossSects配列の成分が、Alignment[N]に属するCrossSectsである。

<Curve>タグの内部を解析するCurve関数の例をリスト4-3-12に示す。

リスト 4-3-12 <Curve>タグの処理

```
void Curve(){
        int param22, prot, pradius, childflag;
        float length, radius;
        logf("#Curve(lc:%d)¥n",LC());
        while((childflag=c1())<0){
                if(scanf(" length=")) param22=SIORU();
                else if(scanf(" rot=")){
                        prot = 0;
                        if(scanf("¥"cw¥""))    prot=1;
                        else if(scanf("¥"ccw¥"")) prot = 2;
                }else if(scanf(" radius=")) pradius = SIORU();
                else abort();
        }
        if(childflag==0) return;
        //process
        length = _f(param22); printf("##length=%f¥n", length);
        radius = _f(pradius);    printf("##radius = %f¥n", radius);
        Curve.length[NCurve] = length;
        Curve.radius[NCurve] = radius;
        Curve.rot[NCurve] = prot;
        //サブ参照
```

```
		for(;;){
			c2();
			if(scanf("</Curve>")) break;
			else if(scanf("<Start")){
				Curve.Start[NCurve] = NPoint;
				Start();
			}else if(scanf("<Center")){
				Curve.Center[NCurve] = NPoint;
				Center();
			}else if(scanf("<End")){
				Curve.End[NCurve] = NPoint;
				End();
			}else abort();
		}//endfor
		NCurve++;
}
```

同様の関数をメタファイル内で必要なタグに関して作成することにより、意味のある情報を読み出す。後述のように、任意の XML ファイルを解析する XML.cmm を用いることにより、ある特定の LandXML ファイルに用いられているタグ構成を解析するメタファイルのテンプレートを自動生成することができる。このテンプレートを用いて、必要なデータ構築の処理を追記することにより、メタファイルのコーディングの手間を省いている。

2017 時点で最新の 8 サンプルデータを共通に変換するメタファイルである。

リスト 4-3-13 メタファイル LandXML.cmm

```
# cmm 生成
int dummy;
void LandXML();
void Project();
void Feature();
void Property();
void Application();
void Author();
void CoordinateSystem();
void Units();
void Metric();
void Alignments();
void Alignment();
void CoordGeom();
void Line();
void Curve();
void Start();
void End();
void Profile();
void ProfAlign();
void PVI();
void CrossSects();
void CrossSect();
void DesignCrossSectSurf();
void CrossSectPnt();
void Roadways();
void Roadway();
void Speeds();
void DesignSpeed();
void Surfaces();
void Surface();
void Definition();
void Pnts();
void P();
void Faces();
void F0();
float _f(int i);
```

```
quat P0[1000];
int P1,P2,P3,V1,V2,V3,F1,G1;

int Alignments.h[100], NAlignments;

int NAlignment;
int Alignment.name[100];
int Alignment.length[100];
int Alignment.staStart[100];
int Alignment.CoordGeom[100];
int Alignment.Feature[100];
int Alignment.Profile[100];
int Alignment.CrossSects[100];

int NPoint;//Start,End 等
int Point.name[100];
quat Point.q[100];

int NLine;
//int Line.length[100];
float Line.length[100];
int Line.Start[100];
int Line.End[100];
int NCurve;
float Curve.length[100];
int Curve.rot[100];
float Curve.radius[100];
int Curve.Start[100];
//quat Curve.Center[100];
int Curve.Center[100];
int Curve.End[100];

int NCoordGeom;
int CoordGeom.type[100];//Line|Curve|Spiral
int CoordGeom.ref[100];

int CrossSects.h[100], NCrossSects;

int NCrossSect;
int CrossSect.name[100];
int CrossSect.sta[100];//記述内容は浮動だがとりあえずアドレスを取得
int CrossSect.DesignCrossSectSurf[100]; //head

int NDesignCrossSectSurf;
int DesignCrossSectSurf.name[100];
int DesignCrossSectSurf.side[100];
int DesignCrossSectSurf.desc[100];
int DesignCrossSectSurf.CrossSectPnt[100]; //NCrossSectPnt

int NCrossSectPnt;
int CrossSectPnt.code[1000];
float CrossSectPnt.x[1000];
float CrossSectPnt.y[1000];

int Property.label[100], Property.value[100], NProperty;

float Sta[100];
quat Orbit[100];//中心線軌跡
int Nsta;

int LC(){//現在の SIORI 位置の行番号
   int i,lc,siori;
   siori   = SIORI();
    SEEK(0);
    for(lc=i=0;i<siori;i++){
        c = GETC();
        if(c == '¥n') lc++;
```

```
    }
    return lc;
}

void abort(){//厠
    int i,c,siori,lc;
    siori    = SIORI();
     SEEK(0);
     for(lc=i=0;i<siori;i++){
          c = GETC();
          if(c == '¥n') lc++;
    }
    printf("xxxxxxxxxxxxxxxxxx abort at line %d xxxxxxxxxxxxxxxxxx¥n{", lc+1 );
    SEEK(siori-10);
    for(i=0;i<20;i++){
          if(i==10) printf("?");
          c = GETC();
          if(c < 32) printf("<%x>",c);
          else printf("%c", c);
    }
    printf("}¥n");
    exit(3);
}

void summary(){//配列の必要長さを知る
                printf("/**SUMMARY**************¥n");
                printf("NAlignments=%d¥n", NAlignments);
                printf("NAlignment=%d¥n", NAlignment);
                printf("NPoint=%d¥n", NPoint);
                printf("NLine=%d¥n", NLine);
                printf("NCurve=%d¥n", NCurve);
                printf("NCoordGeom=%d¥n", NCoordGeom);
                printf("NCrossSect=%d¥n", NCrossSect);
                printf("NDesignCrossSectSurf=%d¥n", NDesignCrossSectSurf);
                printf("NCrossSectPnt=%d¥n", NCrossSectPnt);
                printf("NProperty=%d¥n", NProperty);
                printf("Nsta=%d¥n", Nsta);
                printf("**SUMMARY**************/¥n");
}

/**SUMMARY**************
NAlignments=2
NAlignment=2
NPoint=4
NLine=2
NCurve=0
NCoordGeom=2
NCrossSect=11
NDesignCrossSectSurf=42
NCrossSectPnt=84
NProperty=21
Nsta=3
**SUMMARY**************/

//二重引用符で囲まれた文字列をパースし先頭アドレスを返す
int SIORU(){
    int adress,c,count;
    c = GETC();    if(c!='"') exit(logf("[%c]しおる¥n",c));
    adress = SIORI();
count = 0;
    while(GETC()!='"'){
       if(100<count++) abort();
    }
    return adress;
}

int c1(){//パラ終了検出
```

```
   if(scanf(" />")) return 0;
   if(scanf(" >")) return 1;
   return -1;
}

void c2(){
        scanf("%*[^<]");
}

/*
void c3(){ // c 2 ハングる？
   int c;
   printf("[");
   while(c = GETC()){
      if(c < 32) printf("<%x>",c);
      else printf("%c",c);
      if(c == '<') break;
   }
   SEEK(SIORI()-1);
}
*/

//個別タグ関数
void F0(){
        int childflag,p1,p2,p3;
        logf("#F(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
    scanf(" %d", p1);
    scanf(" %d", p2);
        scanf(" %d", p3);
        P1 = COORD(P0[p1]);
        P2 = COORD(P0[p2]);
        P3 = COORD(P0[p3]);
        GROUP_FACE(G1,F1);
        //サブ参照
        for(;;){
                c2();
                if(scanf("</F>")) return;
        }//endfor
}

void Faces(){
        int childflag;
        logf("#Faces(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</Faces>")) return;
                else if(scanf("<F")) F0();
        }//endfor
}

void P(){
        int param30, childflag;
        quat Qp;
        logf("#P(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" id=")){ //param30=SIORU();
                        scanf("¥"%d¥"",param30);
                        printf("#id=%d¥n",param30);
                }
        }
```

```
		if(childflag==0) return;
		//サブ参照
		scanf("%qx %qy %qz", Qp);
		P0[param30] = Qp;
		for(;;){
			c2();//'<'探す
			if(scanf("</P>")) return;
		}//endfor
}

void Pnts(){
		int childflag;
		logf("#Pnts(%d)¥n",SIORI());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Pnts>")) return;
			else if(scanf("<P")) P();
		}//endfor
}

void Definition(){
		int param29, childflag;
		logf("#Definition(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" surfType=")) param29=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Definition>")) return;
			else if(scanf("<Pnts")) Pnts();
			else if(scanf("<Faces")) Faces();
		}//endfor
}

void Surface(){
		int param6, param7, childflag;
		logf("#Surface(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
			else if(scanf(" desc=")) param7=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Surface>")) return;
			else if(scanf("<Definition")) Definition();
		}//endfor
}

void Surfaces(){
		int childflag;
		logf("#Surfaces(%d)¥n",SIORI());
		while((childflag=c1())<0){
if(scanf(" name")) printf("#Surfaces（%d）が腐っている¥n", LC());
else abort();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
```

```
                        if(scanf("</Surfaces>")) return;
                        else if(scanf("<Surface")) Surface();
        }//endfor
}

void DesignSpeed(){
        int param28, childflag;
        logf("#DesignSpeed(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" speed=")) param28=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</DesignSpeed>")) return;
        }//endfor
}

void Speeds(){
        int childflag;
        logf("#Speeds(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Speeds>")) return;
                        else if(scanf("<DesignSpeed")) DesignSpeed();
        }//endfor
}

void Roadway(){
        int param6, param27, childflag;
        logf("#Roadway(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" name=")) param6=SIORU();
                        else if(scanf(" alignmentRefs=")) param27=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Roadway>")) return;
                        else if(scanf("<Speeds")) Speeds();
        }//endfor
}

void Roadways(){
        int childflag;
        logf("#Roadways(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Roadways>")) return;
                        else if(scanf("<Roadway")) Roadway();
        }//endfor
}
void _o(int adr);

void CrossSectPnt(){
        int param26, childflag;
        float x,y;
```

```
		param26 = 0;
		logf("#CrossSectPnt(lc:%d)",LC());
		while((childflag=c1())<0){
			if(scanf(" code=")) param26=SIORU();
		}
		if(0<param26) _o(param26);
		printf("¥n");
		if(childflag==0) return;
		//process
		CrossSectPnt.code[NCrossSectPnt] = param26;
		scanf(" %f", x);
		scanf(" %f", y);
		CrossSectPnt.x[NCrossSectPnt] = x;
		CrossSectPnt.y[NCrossSectPnt] = y;
		printf("# CrossSectPnt[%d]=", NCrossSectPnt);
		logf(" (%f,", x);
		logf("%f)¥n", y);
		NCrossSectPnt++;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</CrossSectPnt>")) return;
		}//endfor
}

void DesignCrossSectSurf(){
		int param6, param25, param7, childflag;
		int pmaterial, ptypicalThickness,pclosedArea;
		logf("#DesignCrossSectSurf(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
			else if(scanf(" side=")) param25=SIORU();
			else if(scanf(" desc=")) param7=SIORU();
			else if(scanf(" material=")) pmaterial=SIORU();
			else if(scanf(" typicalThickness=")) ptypicalThickness=SIORU();
			else if(scanf(" closedArea=")) pclosedArea=SIORU();
else abort();
		}
		if(childflag==0) return;
		//shori
		DesignCrossSectSurf.name[NDesignCrossSectSurf] = param6;
		DesignCrossSectSurf.side[NDesignCrossSectSurf] = param25;
		DesignCrossSectSurf.desc[NDesignCrossSectSurf] = param7;
		DesignCrossSectSurf.CrossSectPnt[NDesignCrossSectSurf] = NCrossSectPnt;
		//サブ参照
		for(;;){//通常は、二つの CrossSectPnt を有する
			c2();
			if(scanf("</DesignCrossSectSurf>")) break;
			else if(scanf("<CrossSectPnt")) CrossSectPnt();
			else if(scanf("<Feature")) Feature();
else abort();
		}//endfor
		//ループ終了後の NCrossSectPnt は、
		//DesignCrossSectSurf.CrossSectPnt[NDesignCrossSectSurf+1]に記録される
		printf("# NDesignCrossSectSurf=%d¥n", NDesignCrossSectSurf);
		NDesignCrossSectSurf++;
}

void CrossSect(){
		int param6, param24, childflag;
		int pangleSkew;
		logf("#CrossSect(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
			else if(scanf(" sta=")) param24=SIORU();
			else if(scanf(" angleSkew=")) pangleSkew=SIORU();
else abort();
```

```
		}
		if(childflag==0) return;
		//isi
		CrossSect.name[NCrossSect] = param6;
		CrossSect.sta[NCrossSect] = param24;
		CrossSect.DesignCrossSectSurf[NCrossSect] = NDesignCrossSectSurf;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</CrossSect>")) break;
			else if(scanf("<DesignCrossSectSurf")) DesignCrossSectSurf();
else abort();
		}//endfor
		NCrossSect++;
}

void CrossSects(){
		int childflag;
		logf("#CrossSects(%d)¥n",SIORI());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//サブ参照
		CrossSects.h[NCrossSects] = NCrossSect;
		for(;;){
			c2();
			if(scanf("</CrossSects>")) break;
			else if(scanf("<CrossSect")) CrossSect();
			else if(scanf("<Feature")) Feature();
		}//endfor
		NCrossSects++;
		printf("#/CrossSects h=%d,", CrossSects.h[NCrossSects-1]);
		printf(" t=%d¥n", NCrossSect - 1);
}

void PVI(){
		int childflag;
		logf("#PVI(lc=%d)¥n",LC());
		while((childflag=c1())<0){
abort();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</PVI>")) return;
else abort();
		}//endfor
}

void ParaCurve(){
		int childflag;
		int plength;
		logf("#ParaCurve(lc=%d)¥n",LC());
		while((childflag=c1())<0){
			if(scanf(" length=")) plength=SIORU();
else abort();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</ParaCurve>")) return;
else abort();
		}//endfor
}
```

```
void ProfAlign(){
		int param6, childflag;
		logf("#ProfAlign(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</ProfAlign>")) return;
			else if(scanf("<PVI")) PVI();
			else if(scanf("<ParaCurve")) ParaCurve();
else abort();
		}//endfor
}

void Profile(){
		int childflag;
		logf("#Profile(%d)¥n",SIORI());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Profile>")) return;
			else if(scanf("<ProfAlign")) ProfAlign();
		}//endfor
}

void Center(){
		int param6, childflag;
		quat q;
		logf("#Center(lc=%d)¥n",LC());
		while((childflag=c1())<0){
//			if(scanf(" name=")) param6=SIORU();
		}
		if(childflag==0) return;
		//内容取得
		scanf("%qx %qy", q);
		printf("# Center=%q¥n",q);
		Point.name[NPoint] = param6;
		Point.q[NPoint] = q;
		NPoint++;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Center>")) return;
			else abort();
		}//endfor
}

void End(){
		int param6, childflag;
		quat q;
		logf("#End(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
		}
		if(childflag==0) return;
		//内容取得
		scanf("%qx %qy %qz", q);
		printf("# End=%q¥n",q);
		Point.name[NPoint] = param6;
		Point.q[NPoint] = q;
```

```
		NPoint++;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</End>")) return;
		}//endfor
}

void Start(){
		int param6, childflag;
		quat q;
		logf("#Start(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();//name 常に BP
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある
		scanf("%qx %qy %qz", q);
		printf("# Start=%q¥n",q);
		Point.name[NPoint] = param6;
		Point.q[NPoint] = q;
		NPoint++;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Start>")) return;
		}//endfor
}

void Line(){
		int param22, childflag;
		logf("#Line(lc:%d)¥n",LC());
//abort();
		while((childflag=c1())<0){
			if(scanf(" length=")) param22=SIORU();
else abort();
		}
		if(childflag==0) return;
		//process
		Line.length[NLine] = _f(param22);
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Line>")) break;
			else if(scanf("<Start")){
				Line.Start[NLine] = NPoint;
				Start();
			}else if(scanf("<End")){
				Line.End[NLine] = NPoint;
				End();
//abort();
			}else abort();
		}//endfor
		logf("#/Line(%d)¥n",SIORI());
		NLine++;
}

void Curve(){//ハンドアセンブル
		int param22, prot, pradius, childflag;
		float length, radius;
		logf("#Curve(lc:%d)¥n",LC());
		while((childflag=c1())<0){
			if(scanf(" length=")) param22=SIORU();
			else if(scanf(" rot=")){
				prot = 0;
				if(scanf("¥"cw¥"")) prot=1;
				else if(scanf("¥"ccw¥"")) prot = 2;
```

```
//printf("prot=%d¥n", prot);
//exit(0);
                        }else if(scanf(" radius=")) pradius = SIORU();
                        else abort();
        }
        if(childflag==0) return;
        //process
        length = _f(param22); printf("##length=%f¥n", length);
        radius = _f(pradius);   printf("##radius = %f¥n", radius);
        Curve.length[NCurve] = length;
        Curve.radius[NCurve] = radius;
        Curve.rot[NCurve] = prot;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Curve>")) break;
                        else if(scanf("<Start")){
                                Curve.Start[NCurve] = NPoint;
                                Start();
                        }else if(scanf("<Center")){
                                Curve.Center[NCurve] = NPoint;
                                Center();
                        }else if(scanf("<End")){
                                Curve.End[NCurve] = NPoint;
                                End();
                        }else abort();
        }//endfor
        NCurve++;
}

void CoordGeom(){//この中に line, curve, spiral が
        int childflag;
        logf("#CoordGeom(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //syori
//      CoordGeom.Line[NCoordGeom] = NLine;
//      CoordGeom.Curve[NCoordGeom] = NCurve;
//      CoordGeom.Spiral[NCoordGeom] = NSpiral;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</CoordGeom>")) break;
                        else if(scanf("<Line")){
                                CoordGeom.type[NCoordGeom] = 1;//Line
                                CoordGeom.ref[NCoordGeom++] = NLine;
                                Line();
                        }else if(scanf("<Curve")){
                                CoordGeom.type[NCoordGeom] = 2;//Curve
                                CoordGeom.ref[NCoordGeom++] = NCurve;
                                Curve();
                        }else abort();
        }//endfor
        printf("# NCoordGeom=%d¥n", NCoordGeom);
}

void tagPI(){//ハンドアセンブル
        int name, childflag;
        quat q;
        logf("#PI(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" name=")) name=SIORU();
        }
        if(childflag==0) return;
        //内容取得:ある場合とない場合（定義済名称参照のみ）がある
```

```
		//サブ参照
		for(;;){
			c2();
			if(scanf("</PI>")) return;
			else abort();
		}//endfor
}

void AlignPI(){//ハンドアセンブル
		int childflag;
		quat q;
		logf("#AlignPI(%d)¥n",SIORI());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある

		//サブ参照
		for(;;){
			c2();
			if(scanf("</AlignPI>")) return;
			else if(scanf("<PI")) tagPI(); //PI は予約語
			else abort();
		}//endfor
}

void AlignPIs(){//ハンドアセンブル
		int childflag;
		quat q;
		logf("#AlignPIs(%d)¥n",SIORI());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある

		//サブ参照
		for(;;){
			c2();
			if(scanf("</AlignPIs>")) return;
			else if(scanf("<AlignPI")) AlignPI();
			else abort();
		}//endfor
}

void AdverseSE(){//ハンドアセンブル
		int pstaStart, pstaEnd, childflag;
		quat q;
		logf("#AdverseSE(lc:%d)¥n",LC());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある
		scanf("non-adverse");
		//サブ参照
		for(;;){
			c2();
			if(scanf("</AdverseSE>")) return;
			else abort();
		}//endfor
}

void FullSuperelev(){//ハンドアセンブル
		int pstaStart, pstaEnd, childflag;
		quat q;
		logf("#FullSuperelev(lc:%d)¥n",LC());
		while((childflag=c1())<0){
		}
```

```
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある
		scanf("%*f");
		//サブ参照
		for(;;){
			c2();
			if(scanf("</FullSuperelev>")) return;
			else abort();
		}//endfor
}

void RunoffSta(){//ハンドアセンブル
		int pstaStart, pstaEnd, childflag;
		quat q;
		logf("#RunoffSta(lc:%d)¥n",LC());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある
		scanf("%*f");
		//サブ参照
		for(;;){
			c2();
			if(scanf("</RunoffSta>")) return;
			else abort();
		}//endfor
}

void EndofRunoutSta(){//ハンドアセンブル
		int pstaStart, pstaEnd, childflag;
		quat q;
		logf("#EndofRunoutSta(lc:%d)¥n",LC());
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある
		scanf("%*f");
		//サブ参照
		for(;;){
			c2();
			if(scanf("</EndofRunoutSta>")) return;
			else abort();
		}//endfor
}

void Superelevation(){//ハンドアセンブル
		int pstaStart, pstaEnd, childflag;
		quat q;
		logf("#SuperElevation(lc:%d)¥n",LC());
		while((childflag=c1())<0){
			if(scanf(" staStart=")) pstaStart=SIORU();
			else if(scanf(" staEnd=")) pstaEnd=SIORU();
		}
		if(childflag==0) return;
		//内容取得:ある場合とない場合（定義済名称参照のみ）がある

		//サブ参照
		for(;;){
			c2();
			if(scanf("</Superelevation>")) return;
			else if(scanf("<AdverseSE")) AdverseSE();
			else if(scanf("<FullSuperelev")) FullSuperelev();
			else if(scanf("<RunoffSta")) RunoffSta();
			else if(scanf("<EndofRunoutSta")) EndofRunoutSta();
			else abort();
		}//endfor
}
```

```
void Alignment(){
	int param6, param22, param23, pdesc, childflag;
	logf("#Alignment(%d)¥n",SIORI());
//abort();
	while((childflag=c1())<0){
		if(scanf(" name=")) param6=SIORU();
		else if(scanf(" length=")) param22=SIORU();
		else if(scanf(" staStart=")) param23=SIORU();
		else if(scanf(" desc=")) pdesc=SIORU();
		else if(scanf(" desk=")){
			printf("#desk ラベルは、desc ラベルの誤記では？¥n");
			pdesc=SIORU();
		}else abort();
	}
//abort();
	Alignment.name[NAlignment] = param6;
	Alignment.length[NAlignment] = param22;
	Alignment.staStart[NAlignment] = param23;
	Alignment.CoordGeom[NAlignment] = NCoordGeom;
	Alignment.CrossSects[NAlignment] = NCrossSects;//構造体配列の先頭
	if(childflag==0) return;
	//サブ参照
//abort();
	for(;;){
		c2();
		if(scanf("</Alignment>")) break;
		else if(scanf("<CoordGeom")) CoordGeom();
		else if(scanf("<Feature")) Feature();
		else if(scanf("<Profile")) Profile();
		else if(scanf("<CrossSects")) CrossSects();
		else if(scanf("<AlignPIs")) AlignPIs();
		else if(scanf("<Superelevation")) Superelevation();
else abort();
	}//endfor
	NAlignment++;
}

void Alignments(){
	int childflag;
	logf("#Alignments(%d)¥n",SIORI());
	while((childflag=c1())<0){
	}
	if(childflag==0) return;
	//サブ参照
//abort();ここまでは来る
	Alignments.h[NAlignments] = NAlignment;
	for(;;){
		c2();//c3();//c2();
		if(scanf("</Alignments>")) break;
		else if(scanf("<Alignment")) Alignment();//この中で NAlignment++
		else if(scanf("<Feature")) Feature();
	}//endfor
	NAlignments++;
	printf("#/Alignments h=%d,", Alignments.h[NAlignments-1]);
	printf("t=%d¥n", NAlignment-1);
}

void Metric(){
	int param15, param16, param17, param18, param19, param20, param21, childflag;
	logf("#Metric(%d)¥n",SIORI());
	while((childflag=c1())<0){
		if(scanf(" areaUnit=")) param15=SIORU();
		else if(scanf(" linearUnit=")) param16=SIORU();
		else if(scanf(" volumeUnit=")) param17=SIORU();
		else if(scanf(" temperatureUnit=")) param18=SIORU();
		else if(scanf(" pressureUnit=")) param19=SIORU();
```

```
                        else if(scanf(" angularUnit=")) param20=SIORU();
                        else if(scanf(" directionUnit=")) param21=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Metric>")) return;
        }//endfor
}

void Units(){
        int childflag;
        logf("#Units(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Units>")) return;
                        else if(scanf("<Metric")) Metric();
        }//endfor
}

void CoordinateSystem(){
        int param6, param12, param13, param14, param7, childflag;
        logf("#CoordinateSystem(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" name=")) param6=SIORU();
                        else if(scanf(" horizontalDatum=")) param12=SIORU();
                        else if(scanf(" verticalDatum=")) param13=SIORU();
                        else if(scanf(" horizontalCoordinateSystemName=")) param14=SIORU();
                        else if(scanf(" desc=")) param7=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</CoordinateSystem>")) return;
                        else if(scanf("<Feature")) Feature();
        }//endfor
}

void Author(){
        int param10, param11, childflag;
        logf("#Author(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" createdBy=")) param10=SIORU();
                        else if(scanf(" company=")) param11=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                        c2();
                        if(scanf("</Author>")) return;
        }//endfor
}

void Application(){
        int param6, childflag;
        logf("#Application(%d)¥n",SIORI());
        while((childflag=c1())<0){
                        if(scanf(" name=")) param6=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
```

```
		for(;;){
			c2();
			if(scanf("</Application>")) return;
			else if(scanf("<Author")) Author();
		}//endfor
}

void Property(){
		int param8, param9, childflag;
		logf("#Property(%d)¥n",SIORI());
		while((childflag=c1())<0){
//abort();
			if(scanf(" label=")){ param8=SIORU();}
			else if(scanf(" value=")){ param9=SIORU(); }
else abort();
		}
		Property.label[NProperty] = param8;
		Property.value[NProperty] = param9;
		NProperty++;
//if(1 < NProperty) abort();
		printf("# NProperty=%d¥n", NProperty);
printf("###childflag = %d¥n", childflag);
//abort();
//if(1 < NProperty) abort();
		if(childflag==0) return;
abort();
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Property>")) return;
		}//endfor
}

void Feature(){
		int param6, childflag;
		logf("#Feature(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
//c3();
			if(scanf("</Feature>")) return;
			else if(scanf("<Property")) Property();
else abort();
		}//endfor
abort();
}

void Project(){
		int param6, param7, childflag;
		logf("#Project(%d)¥n",SIORI());
//abort();
		while((childflag=c1())<0){
			if(scanf(" name=")) param6=SIORU();
			else if(scanf(" desc=")) param7=SIORU();
else abort();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Project>")) return;
			else if(scanf("<Feature")) Feature();
else abort();
		}//endfor
```

```
}

void LandXML(){
	int param0, param1, param2, param3, param4, param5, childflag;
	logf("#LandXML(%d)¥n",SIORI());
	while((childflag=c1())<0){
		if(scanf(" date=")) param0=SIORU();
		else if(scanf(" time=")) param1=SIORU();
		else if(scanf(" version=")) param2=SIORU();
		else if(scanf(" xsi:schemaLocation=")) param3=SIORU();
		else if(scanf(" xmlns=")) param4=SIORU();
		else if(scanf(" xmlns:xsi=")) param5=SIORU();
else abort();
	}
	if(childflag==0) return;
	//サブ参照
	for(;;){
		c2();
		if(scanf("</LandXML>")) return;
		else if(scanf("<Project")) Project();
		else if(scanf("<Application")) Application();
		else if(scanf("<CoordinateSystem")) CoordinateSystem();
		else if(scanf("<Units")) Units();
		else if(scanf("<Alignments")) Alignments();
		else if(scanf("<Roadways")) Roadways();
		else if(scanf("<Surfaces")) Surfaces();
else abort();
	}//endfor
}

void template(){
	P1 = COORD(0,0,0);
	P2 = COORD(0,0,1);
	P3 = COORD(0,1,0);
	V1 = VERTEX(P1);
	V2 = VERTEX(P2);
	V3 = VERTEX(P3);
	F1 = FACE(V1,V2,V3);
	G1 = GROUP();
}

/*
int Alignment.name[100];
int Alignment.length[100];
int Alignment.staStart[100];
int Alignment.CoordGeom[100];
int Alignment.Feature[100];
int Alignment.Profile[100];
int Alignment.CrossSects[100];
*/
void outputCrossSectsAll(){
	int i;
	for(i=0;i<NCrossSects;i++){
		printf("# CrossSects[%d]=", i);
		printf("%d¥n", CrossSects.h[i]);
	}
	printf("# NCrossSect=%d¥n", NCrossSect);
}

void PUTC(int c){
	printf("%c",c);
}

void q(){
	PUTC("");
}

void _o(int i){
```

```
	int c,siori;
	siori=SIORI();
	SEEK(i);
	q();//printf("¥"");
	for(c=GETC();c!='"';c=GETC()){
		printf("%c",c);
	}
	q();//printf("¥"");
	SEEK(siori);
}

float _f(int i){
	int c,siori;
	float f;
	siori=SIORI();
	SEEK(i);
	scanf("%f",f);
	SEEK(siori);
	return f;
}

void outputCrossSectPnt(int i){
	printf("#     CrossSectPnt[%d]={",i);
	printf("code="); _o(CrossSectPnt.code[i]);
	printf(",x=%f",CrossSectPnt.x[i]);
	printf(",y=%f}¥n",CrossSectPnt.y[i]);
}

void outputDesignCrossSectSurf(int head, int tail){//横断面を構成する各辺
	int i;
	printf("# outputDesignCrossSectSurf(%d-", head);
	printf("%d)¥n", tail);
	for(i=head;i<tail;i++){
		printf("#    DesignCrossSectSurf[%d]=",i);
		printf("{name="); _o(DesignCrossSectSurf.name[i]);
		printf(",side="); _o(DesignCrossSectSurf.side[i]);
		printf(",desc=");_o(DesignCrossSectSurf.desc[i]);
		if(0<DesignCrossSectSurf.CrossSectPnt[i+1]){
			printf(",CrossSectPnt=[%d-",DesignCrossSectSurf.CrossSectPnt[i]);
			printf("%d]}¥n",DesignCrossSectSurf.CrossSectPnt[i+1]-1);
			outputCrossSectPnt(DesignCrossSectSurf.CrossSectPnt[i]);
			outputCrossSectPnt(DesignCrossSectSurf.CrossSectPnt[i+1]-1);
		}else{
			printf(",CrossSectPnt=[%d-",DesignCrossSectSurf.CrossSectPnt[i]);
			printf("%d]}¥n",NCrossSectPnt-1);
			outputCrossSectPnt(DesignCrossSectSurf.CrossSectPnt[i]);
			outputCrossSectPnt(NCrossSectPnt-1);
		}
	}
}

void outputCrossSects(int n){
	int i,h,t;
	float f;
	h = CrossSects.h[n];
	if(n == NCrossSects -1) t = NCrossSect;
	else t = CrossSects.h[n+1];
	Nsta = 0;
	for(i=h;i<t;i++){
		printf("# CrossSect[%d]={", i);
		printf("name:");_o(CrossSect.name[i]);
		printf(",sta:"); f=_f(CrossSect.sta[i]);
		Sta[Nsta++] = f;
		printf(",DesignCrossSectSurf:%d}¥n", CrossSect.DesignCrossSectSurf[i]);
	}
}

void outPoint(int n){
```

```
		printf("<Point name="); _o(Point.name[n]);
		printf("%q />¥n", Point.q[n]);
}

void outputLine(int n){
		printf("#////////////////////////////////Line(%d)////////////////////////////////¥n", n);
//		printf("#Length=");
//		_o(Line.length[n]);
//		printf("¥n");
		printf("#Length=%f¥n",Line.length[n]);
		printf("#Start=%d ", Line.Start[n]); outPoint(Line.Start[n]);
		printf("#End=%d", Line.End[n]); outPoint(Line.End[n]);
}

void outputCoordGeom(int h, int t){
		int i;
		if(t == 0) t = NCoordGeom;
		printf("# CoordGeom(%d", h);
		printf("-%d)¥n",t);
		for(i=h;i<t;i++){
			switch(CoordGeom.type[i]){
			case 1: outputLine(CoordGeom.ref[i]); break;
			default: logf("## unknown CoordGeom.type：%d¥n", CoordGeom.type[i]);
			}
		}
}

quat calcQsta(quat qC, quat qH, float r, int rot, float sta){
//考え方：sta/r で角度はわかる。中心の周りに起点を回転させれば位置が出る
//起点の向きを直角にすれば現在の向きが出る
		quat rotq, posq,q;
		if(rot == 1) rotq = _Q( cos(sta/r/2.0), 0.0, 0.0, sin(sta/r/2.0));
		else if(rot == 2) rotq = _Q( cos(sta/r/2.0), 0.0, 0.0, -sin(sta/r/2.0));
		return qC + rotq*(qH-qC)/rotq;
}

quat qxy(quat q, float r){
		return _Q(0.0, _Qy(q)/r,-_Qx(q)/r, 0.0);
}

quat calcQarah(quat qC, quat qH, float r, int rot, float sta){
		quat rotq, posq,q;
		if(rot == 1){
			rotq = _Q( cos(sta/r/2.0), 0.0, 0.0, sin(sta/r/2.0));
			q = -rotq*qxy(qH-qC,r)/rotq;
		}else if(rot == 2){
			rotq = _Q( cos(sta/r/2.0), 0.0, 0.0, -sin(sta/r/2.0));
			q = rotq*qxy(qH-qC,r)/rotq;
		}else{
			printf("calcQarah 関数に渡された rot 変数が不適¥n");
			exit(0);
		}
		return q;
}

quat getpos(int ialignment, float sta){
		quat qStart,qEnd,qCenter,qpos,qdir;
		int i,j,t;
		float fh,ft;
//		printf("sta を範囲として含む直線、円弧または螺旋を見つける¥n");
		if(ialignment+1 < NAlignment) t = Alignment.CoordGeom[ialignment+1];
		else t = NCoordGeom;
		fh = ft = _f(Alignment.staStart[ialignment]);
		for(i = Alignment.CoordGeom[ialignment]; i < t; i++){
			j = CoordGeom.ref[i];
			switch(CoordGeom.type[i]){
			case 1: //line
```

```
                                        qStart = Point.q[Line.Start[j]];
                                        qEnd = Point.q[Line.End[j]];
                                        ft += Line.length[j];
                                        break;
                                case 2: //Curve
                                        qStart = Point.q[Curve.Start[j]];
                                        qEnd = Point.q[Curve.End[j]];
                                        qCenter = Point.q[Curve.Center[j]];
//                                      fh = _f(Curve.Start[j]);
//                                      ft = _f(Curve.End[j]);
                                        ft += Curve.length[j];
                                        break;
                                case 3: //Spiral
                                default:
                                        printf("###直線でも円弧でもない！¥n");
                                        continue;
                                }
                                if(sta < fh){
                                        printf("###範囲外の外¥n") ;//失敗
                                        continue;
                                }
                                if(ft < sta){
                                        printf("###区間前  fh=%f,", fh);
                                        printf("    ft=%f¥n", ft);
                                        fh = ft;
                                        continue;//次の区間に期待
                                }
                                switch(CoordGeom.type[i]){
                                case 1: //line
                                        qpos = qStart + (qEnd-qStart) * ((sta-fh)/(ft-fh));
                                        qdir = (qEnd - qStart) / (ft-fh);
                                        printf("###qdir=%q¥n", qdir);
                                        break;
                                case 2: //curve
printf("###sta=%f,",sta);
printf("fh=%f,",fh);
printf("sta-fh=%f¥n", sta-fh);
//exit(0);
                                        qpos = calcQsta(qCenter,qStart,Curve.radius[j],Curve.rot[j], sta-fh);
                                        qdir =    calcQarah(qCenter,qStart,Curve.radius[j],Curve.rot[j], sta-fh);
                                        printf("###qdir=%q¥n", qdir);
                                        break;
                                case 3: //spiral
                                default:
                                        qpos = _Q(0.0,0.0,0.0,0.0);
                                }
                                break;
                }
                return qpos;
}//getpos

//sta の位置座標と方向を同時に求め、配列の指定アドレスに格納する
quat Orbit.pos[100], Orbit.arah[100], Orbit.rotq[100];

void getposarah(int ialignment, float sta, int n){
                quat qStart,qEnd,qCenter,qpos,qdir;
                int i,j,t;
                float fh,ft;
float tsta; //sta  の、中心に対する角度
//              printf("sta  を範囲として含む直線、円弧または螺旋を見つける¥n");
                if(ialignment+1 < NAlignment) t = Alignment.CoordGeom[ialignment+1];
                else t = NCoordGeom;
                fh = ft = _f(Alignment.staStart[ialignment]);
                for(i = Alignment.CoordGeom[ialignment]; i < t; i++){
                                j = CoordGeom.ref[i];
                                switch(CoordGeom.type[i]){
                                case 1: //line
```

```
                                        qStart = Point.q[Line.Start[j]];
                                        qEnd = Point.q[Line.End[j]];
                                        ft += Line.length[j];
                                        break;
                                case 2: //Curve
                                        qStart = Point.q[Curve.Start[j]];
                                        qEnd = Point.q[Curve.End[j]];
                                        qCenter = Point.q[Curve.Center[j]];
                                        ft += Curve.length[j];
printf("###Curve.length=%f¥n", Curve.length[j]);
printf("###ft=%f¥n", ft);
//exit(0);
                                        break;
                                case 3: //Spiral
                                default:
                                        printf("###直線でも円弧でもない！¥n");
                                        continue;
                                }
printf("####ft=%f¥n", ft);
                                if(sta < fh){
                                        printf("###範囲外の外¥n") ;//失敗
                                        continue;
                                }
                                if(ft < sta){
                                        printf("###区間前  fh=%f,", fh);
                                        printf("    ft=%f¥n", ft);
                                        fh = ft;
                                        continue;//次の区間に期待
                                }
                                switch(CoordGeom.type[i]){
                                case 1: //line
                                        Orbit.pos[n] = qStart + (qEnd-qStart) * ((sta-fh)/(ft-fh));
                                        Orbit.arah[n] = qdir = (qEnd - qStart) / (ft-fh);
                                        if(_Qy(qdir) < 1.0)
                                           Orbit.rotq[n] = _Q( -_Qx(qdir)/sqrt(2.0*(1.0-_Qy(qdir))), 0.0,0.0, sqrt( (1.0 -
_Qy(qdir))*0.5));
                                        else{
                                           Orbit.rotq[n] = _Q(1.0,0.0,0.0,0.0);//無回転
                                        }
                                        break;
                                case 2: //curve
                                        printf("#sta に対応する位置 pos を求める¥n");
                                        Orbit.pos[n] = calcQsta(qCenter,qStart,Curve.radius[j],Curve.rot[j], sta-fh);
//printf("#Orbit.pos[n]=%q¥n", Orbit.pos[n]);
                                        Orbit.arah[n] = qdir = calcQarah(qCenter,qStart,Curve.radius[j],Curve.rot[j],
sta-fh);
                                        if(_Qy(qdir) < 1.0){
                                           Orbit.rotq[n] = _Q( -_Qx(qdir)/sqrt(2.0*(1.0-_Qy(qdir))), 0.0,0.0, sqrt( (1.0 -
_Qy(qdir))*0.5));
//                                         Orbit.rotq[n] = _Q( -_Qx(qdir)/sqrt(2.0*(1.0-_Qy(qdir))), 0.0,0.0, -sqrt( (1.0 -
_Qy(qdir))*0.5));
printf("###qdir = %q, ", qdir);
printf(" cos t = %f, ", -_Qx(qdir)/sqrt(2.0*(1.0-_Qy(qdir))) );
printf(" sin t = %f¥n", sqrt( (1.0 - _Qy(qdir))*0.5));
//exit(0);
//                                         Orbit.rotq[n] = _Q(1.0,0.0,0.0,0.0);//無回転
                                        }else{
                                           Orbit.rotq[n] = _Q(1.0,0.0,0.0,0.0);//無回転
                                        }
                                        break;
                                case 3: //spiral
                                default:
                                        qpos = _Q(0.0,0.0,0.0,0.0);
                                }
                                break;
                }
}
```

```
int sudah,GSTA,PB,PS,VB,VS,LSTA;
quat qbottom(quat q){
	return _Q(0.0, _Qx(q), _Qy(q), 0.0);
}

void drawSta(){
	int i;
	if(!sudah){
		GSTA = GROUP();
		PB = COORD(0,0,0);
		PS = COORD(0,0,1);
		VB = VERTEX(PB);
		VS = VERTEX(PS);
		LSTA = LINE(VB,VS);
		sudah = 1;
	}
	for(i=0;i<Nsta;i++){
logf("#SSSSSSSSSSSSSSS スタ%d SSSSSSSSSSSSSSSSSS¥n", i);
		PS = COORD(Orbit[i]);
		PB = COORD(qbottom(Orbit[i]));
		GROUP_LINE(GSTA,LSTA);
	}
}

int cudah,Gcr,Ph,Pt,Vh,Vt,Lcr;

void drawCrossSectPnt(int i, int j, int ista){//通常は二つの頂点を結ぶ。ista 情報をもとに回転移動
	quat q, q1,q2;
	printf("### drawCrossSectPnt[%d-",i);
	printf("%d]",j);
	printf("(%f",CrossSectPnt.x[i]);
	printf(",%f)-",CrossSectPnt.y[i]);
	printf("(%f",CrossSectPnt.x[j]);
	printf(",%f)",CrossSectPnt.y[j]);
	printf(" ISTA=%d¥n", ista);
	q1 = _Q(0.0, -CrossSectPnt.x[i], 0.0, CrossSectPnt.y[i]);
	q = Orbit.rotq[ista];
	q1 = Orbit.pos[ista] + q*q1/q;
	Ph = COORD(q1);
	q2 = _Q(0.0, -CrossSectPnt.x[j], 0.0, CrossSectPnt.y[j]);
	q2 = Orbit.pos[ista] + q*q2/q;
	Pt = COORD(q2);
	GROUP_LINE(Gcr,Lcr);
}

//一つの断面を描く
void drawCross(int head, int tail, int ista){//head,tail は、CrossSects.h[i]～CrossSects.h[i+1]
	int i;
	printf("# drawCrossSectSurf(%d-", head);
	printf("%d)¥n", tail);
	for(i=head;i<tail;i++){
		printf("#   DesignCrossSectSurf[%d]=",i);
		printf("{name="); _o(DesignCrossSectSurf.name[i]);
		printf(",side="); _o(DesignCrossSectSurf.side[i]);
		printf(",desc="); _o(DesignCrossSectSurf.desc[i]);
		if(0<DesignCrossSectSurf.CrossSectPnt[i+1]){
			printf(",CrossSectPnt=[%d-",DesignCrossSectSurf.CrossSectPnt[i]);
			printf("%d]}¥n",DesignCrossSectSurf.CrossSectPnt[i+1]-1);
			drawCrossSectPnt(DesignCrossSectSurf.CrossSectPnt[i],
				DesignCrossSectSurf.CrossSectPnt[i+1]-1, ista);
		}else{
			printf(",CrossSectPnt=[%d-",DesignCrossSectSurf.CrossSectPnt[i]);
			printf("%d]}¥n",NCrossSectPnt-1);
			drawCrossSectPnt(DesignCrossSectSurf.CrossSectPnt[i],
				NCrossSectPnt-1, ista);
		}
```

```
		}
}

void outputSta(int ialignment){
		int i;
		printf("#****************Sta[%f]*****************¥n",// Nsta);
		   _f(Alignment.staStart[ialignment]) );
		for(i=0;i<Nsta;i++){
				Orbit[i] = getpos(ialignment, Sta[i]);
				getposarah(ialignment, Sta[i], i);
				printf("#Sta[%d] = ", i);
				printf("%f", Sta[i]);
				printf(" %q¥n", Orbit[i]);
		}
		printf("#****************/Sta[%f]*****************¥n",// Nsta);
		   _f(Alignment.staStart[ialignment])+_f(Alignment.length[ialignment]) );
}

void drawPntcode(int i){
		printf("adr=%d,", CrossSectPnt.code[i]);
		printf("code="); _o(CrossSectPnt.code[i]);
		printf("¥n");
}

int endcode(int c){/*文字列の終端文字の判定を行う*/
		if(',' == c) c = ','; //名称の中に　.　があってもよい
		if(c == ' ') return 1;
		if(c == '/') return 1;
		if(c == '>') return 1;
		if(c == '=') return 1;
		return 0;
}

int strcmp(int str1, int str2){/*文字列の比較*/
		int pos,i,c1,c2;
		pos = SIORI();
		for(i=0;;i++){
				SEEK(str1+i);
				c1 = GETC();
				SEEK(str2+i);
				c2 = GETC();
				if(endcode(c1)) break;
				if(endcode(c2)) break;
				if(c1<c2) break;
				else if(c1>c2) break;
		}
		SEEK(pos); /*ポインタを元の位置に戻す*/
		if(endcode(c1)){
				if(endcode(c2)) return 0;
				else return -1; //str2 が長い
		}else if(endcode(c2)){
				return 1; //str1 が長い
		}
		if(c1 < c2) return -1;
		if(c1 > c2) return 1;
		return 0;
}

int codecmp(int p1, int p2){
		return strcmp(CrossSectPnt.code[p1], CrossSectPnt.code[p2]);
}

int qe(quat q1, quat q2){
		quat q;
		q = q1 - q2;
		if(_Qt(q) !=0.0) return 0;
		if(_Qx(q) !=0.0) return 0;
```

```
		if(_Qy(q) !=0.0) return 0;
		if(_Qz(q) !=0.0) return 0;
		return 1;
}

void drawSweep(int s1, int s2, int s3, int ista){
//辺群 s1-s2　と辺群 s2-s3 の間のペアを探す
		int i,j,h1,h2,t1,t2,P11,P12,P21,P22,V11,V12,V21,V22,Fsweep,dirflag;
		quat q11,q12,q21,q22, rotq, posq;
		printf("#.....Sweep¥n");
		for(i=s1; i<s2; i++){
			h1 = DesignCrossSectSurf.CrossSectPnt[i];
			h2 = DesignCrossSectSurf.CrossSectPnt[i+1]-1;
			for(j=s2; j<s3; j++){
				t1 = DesignCrossSectSurf.CrossSectPnt[j];
				if(j+1 == NDesignCrossSectSurf) t2 = NCrossSectPnt - 1;
				else t2 = DesignCrossSectSurf.CrossSectPnt[j+1]-1;
				if(codecmp(h1,t1)) continue;
				if(codecmp(h2,t2)) continue;
				//ペアが成立=> 面を出力
				rotq = Orbit.rotq[ista];
				posq = Orbit.pos[ista];
				if(CrossSectPnt.x[h1]<CrossSectPnt.x[h2]) dirflag = -1;
				else dirflag = 1;
				q11 = _Q(0.0, -CrossSectPnt.x[h1], 0.0, CrossSectPnt.y[h1]);
				q11 = posq + rotq*q11/rotq;
				q12 = _Q(0.0, -CrossSectPnt.x[h2], 0.0, CrossSectPnt.y[h2]);
				q12 = posq + rotq*q12/rotq;
if(qe(q11,q12)){
printf("####q11==q12¥n");
continue;
}
				rotq = Orbit.rotq[ista+1];
				posq = Orbit.pos[ista+1];
				q21 = _Q(0.0, -CrossSectPnt.x[t1], 0.0, CrossSectPnt.y[t1]);
				q21 = posq + rotq*q21/rotq;
				q22 = _Q(0.0, -CrossSectPnt.x[t2], 0.0, CrossSectPnt.y[t2]);
				q22 = posq + rotq*q22/rotq;
if(qe(q21,q22)){
printf("####q21==q22¥n");
continue;
}
if(qe(q11,q21)){
printf("####q11==q21¥n");
continue;
}
if(qe(q12,q22)){
printf("####q12==q22¥n");
continue;
}
if(qe(q12,q21)){
printf("####q12==q21¥n");
continue;
}
				P11 = COORD(q11);
				P12 = COORD(q12);
				P21 = COORD(q21);
				P22 = COORD(q22);
if(P11==P12){
printf("####P11==P12¥n");
exit(0);
continue;
}
if(P12==P21){
printf("####P12==P21¥n");
exit(0);
continue;
}
```

```
if(P12==P22){
printf("#### overlap P12==P22¥n");
//exit(0);
continue;
}
if(P11==P21){
printf("#### overlap P11==P21¥n");
//exit(0);
continue;
}
                                        V11 = VERTEX(P11);
                                        V12 = VERTEX(P12);
                                        V21 = VERTEX(P21);
                                        V22 = VERTEX(P22);
                                        if(0<dirflag) Fsweep = FACE(V11,V12,V22,V21);
                                        else Fsweep = FACE(V12,V11,V21,V22);
                                        GROUP_FACE(Gcr,Fsweep);
                        }
                }
                printf("#...../Sweep¥n");
}

void drawCrossSects(int n){
                int i,h,t;
                int s1,s2,s3;
                float f;
                h = CrossSects.h[n];
                if(n == NCrossSects -1) t = NCrossSect;
                else t = CrossSects.h[n+1];
                printf("#::::::::::::::::::::::::::::::::drawCrossSects<%d-",h);
                printf("%d>¥n", t);
                Gcr = GROUP();
                Ph = COORD(0,0,0);
                Vh = VERTEX(Ph);
                Pt = COORD(0,0,1);
                Vt = VERTEX(Pt);
                Lcr = LINE(Vh,Vt);

                Nsta = 0;
                for(i=h;i<t;i++){//各断面を得る
                        printf("# CrossSect[%d]={", i);
                        printf("name:");_o(CrossSect.name[i]);
                        printf(",sta:%f", f=_f(CrossSect.sta[i]));
                        Sta[Nsta++] = f;
                        printf(",DesignCrossSectSurf:%d}¥n", CrossSect.DesignCrossSectSurf[i]);

                        if(0<CrossSect.DesignCrossSectSurf[i+1]){
                                drawCross(CrossSect.DesignCrossSectSurf[i],
                                        CrossSect.DesignCrossSectSurf[i+1], i-h);
                        }else{
                                drawCross(CrossSect.DesignCrossSectSurf[i],
                                        NDesignCrossSectSurf, i-h);
                        }

                }
                for(i=h;i<t-1;i++){ //横断面 i と横断面 i+1 の間の頂点比較を行う
                        if(CrossSect.sta[i] == CrossSect.sta[i+1]){
                                printf("#sweep しようとする二つの横断面の sta が同じ¥n");
                                continue;
                        }
                        s1   = CrossSect.DesignCrossSectSurf[i];
                        s2 = CrossSect.DesignCrossSectSurf[i+1];
                        s3 =    CrossSect.DesignCrossSectSurf[i+2];
                        if(s3 < 1) s3 = NDesignCrossSectSurf;
                        drawSweep(s1,s2,s3,i-h);
                }
                printf("#::::::::::::::::::::::::::::::::/drawCrossSects¥n");
}
```

```
void outputAlignment(int i){
		printf("#---------<Alignment[%d]>----------¥n", i);
		printf("# Alignment.CrossSects[%d]=", i);
		printf("%d¥n", Alignment.CrossSects[i]);
		printf("# Alignment.CrossSects[%d]=", i+1);
		printf("%d¥n", Alignment.CrossSects[i+1]);
		outputCrossSects(Alignment.CrossSects[i]);
		printf("# Alignment.CoordGeom[%d]=",i);
		printf("%d¥n", Alignment.CoordGeom[i]);
		outputCoordGeom( Alignment.CoordGeom[i], Alignment.CoordGeom[i+1]);
		printf("#---------</Alignment[%d]>----------¥n", i);
		outputSta(i);//この中で中心線軌跡を作成
		drawSta();//sta の位置から標高ゼロまで垂線の足を下す
		drawCrossSects(Alignment.CrossSects[i]);
}

void output(){//解読結果の出力：Alignment
		int i;
		printf("############################################################¥n");
		  printf("#NAlignments=%d¥n", NAlignments);
		  for(i=0;i<NAlignments;i++){
			printf("#Alignments.h[%d]=", i);
			printf("%d¥n", Alignments.h[i]);
		}
		printf("#NAlignment=%d¥n",NAlignment);
		for(i=0;i<NAlignment;i++){
			printf("#Alignment[%d]¥n",i);
			outputAlignment(i);
		}
		outputCrossSectsAll();
}

int bom(){//EF BB BF
		if(GETC() == 14*16+15)
		if(GETC() == 11*16+11)
		if(GETC()== 11*16+15)
		return 1;
		SEEK(0);
		return 0;
}

/*
int bom(){//EF BB BF
int c;
//printf("abc¥x42¥xefde¥n");
if(scanf("¥xef")) printf("[%%ef]");
scanf("¥xEF");//あまり効果がない（要デバッグ)
scanf("¥xef");//あまり効果がない（要デバッグ)
c = GETC();
if(c==16*14+15) printf("[EF]");
c = GETC();
c = GETC();
printf("0x%x",c);
//printf("[%x]",c);
//		return scanf("¥xEF¥xBB¥xBF");うまくいかない（要デバッグ)
return 0;
}
*/

int main(){
		bom();
		if(scanf(" <?")) while(GETC() != '>');
		template();
		if(scanf(" <LandXML")) LandXML();
else logf("# <LandXML がない>¥n");
		output();
```

```
        summary();
}
#----------実行終了----------
#戻り値：0
```

④補助的なメタファイル

一般的な XML 形式のファイルを解読するメタファイル(XML.cmm)を作成した。この XML.cmm は、LandXML に限らず任意の XML ファイルを解析して、専用の解読用メタファイルを自動生成するメタファイルである。

これにより、保存データの中で実際に使用されている XML タグの階層に即した解読メタファイルのテンプレートを得ることができる。これに、④でタグ間の記述内容を解読する処理を追加して個別の LandXML ファイルの解読用メタファイルを完成させる。

XML.cmm は、当該 XML ファイルの中で使用されるすべての XML タグとパラメータの構成を解析し、当該 XML ファイルを解析するメタファイルのテンプレートを出力する。

出力されたメタファイルを用いて、仮想コンバータ上で当該 XML ファイルを解読すると、解析だけを行い、参照されたタグの一覧だけを出力する。意味のある処理を行わせるためには、XML が出力したメタファイルをテンプレートとして、これに形態的要素を抽出する利活用処理を追加する。

XML ファイルのタグは、以下のような構成である。

<タグ パラ１=”xxxx” パラ２=”yyyy” />

<タグ パラ１=”xxxx” パラ２=”yyyy”＞ （中身） </タグ>

解析においては、全てのタグとパラメータの辞書を作成する。更に、あるタグに使用されるパラメータとサブタグのリンク辞書を作成する。これらを用いて、トップレベルのタグ（LandXML）から下に向かって階層的に、出現するタグとパラメータを読み出す処理を作成する。

なお、XML.cmm では、記号表機能を使用せず、タグ名称、パラメータ名称のリストを、初出アドレスで管理する。この方法により、整数配列だけで辞書を管理している。

また、仮想コンバータでは用意していない文字列比較のためのライブラリ関数を、メタファイルの中で定義する関数として提供している。引数は、通常の C 言語では文字列のアドレスであるが、本処理系においては、データファイルの中のアドレスである。

strcmp(adr1, adr2)

は、adr1 から始まる文字列と adr2 から始まる文字列の比較を行う。いずれかの文字列が終端文字に到達すると、比較は終了する。終端文字は、引用符、句読点、改行コードである。

c:¥@keikan¥kdb¥geometry¥C--サンプル¥入出力¥LandXML¥XML.cmm

c:¥@keikan¥kdb¥geometry¥LandXML¥LandXML_Test20SJIS.xml

main 関数は、ヘッダを解読した後に、トップレベルのタグを tag 関数で処理する。tag 関数は、その中に階層的に組み込まれた下位のタグを再帰的に処理する(リスト 4-3-12(1))。

リスト 4-3-14 XML.cmm メイン関数

```
int main(){
		int c,count,siori;
		NLINK = NLINKPARAM = 1;
		logf("¥n#count=%d¥n",count);
		logf("#LEN=%d¥n",LEN());/*データファイルの長さ*/
		NDIC = 0; /*辞書クリア*/
		headline();
		tag();
		logf("#----------------SUMMARY--------------------------¥n");
		logf("#NDIC=%d¥n",NDIC);
		if(ERROR){
			logf("#ERROR(%d)¥n",ERROR);
			return 140503;
		}else{
			createcmm();
			return 0;
		}
}
```

あるタグのパラメータと、サブタグを解析する tag 関数をリスト 4-3-9 (2)に示す。タグの名称とパラメータの名称を辞書に登録するとともに、このタグの内部に定義されるサブタグを再帰的に解析し、親タグと子タグのリンクリストを作成している。

リスト 4-3-15 XML.cmm タグ関数

```
int tag(){
		int siori,c,head;
		int label,param,tagID,subtagID,paramID;

		seeklt();
		head = SIORI();
		tagID = adddic(head+1);
		logf("#(++%d)",++LEVEL);/*再帰レベル*/
		printtag(); //' 'または'>'で終了する
		while(siori=seeklabel()){
			paramID = addparamdic(siori);//この中で改行される場合がある
			linkparam(tagID,paramID);//未リンクならリンク
			loglabel();
			param = findparam();//パラメータの代入値をログ出力
			logf("=[");
			printparam(param);
			logf("]¥n");
		}
		if(scanf(">")){ //パラメータ部が/>の/で終了
			logf("#(%d--)/>¥n",LEVEL);
			LEVEL--;
			return tagID;//サブタグなし
		}
		printcontent();
		for(;;){
			seeklt();
			siori = SIORI();
			if(scanf("</")){
				break;
			}else{
				subtagID = tag();
```

```
                                        link(tagID, subtagID);
                        }
                        if(0<ERROR) break;
                }
                logf("#(%d--)</",LEVEL--);
                logtagname(SIORI());
                seekgt();
                logf(">¥n");
                return tagID;
}
```

最後に、解析結果を格納した辞書を用いて、この XML ファイルを読み込む処理を記述したメタファイルを出力する createcmm 関数をリスト 4-3-9 (3) に示す。

リスト 4-3-16 XML.cmm タグ構成解析結果の出力関数

```
void createcmm(){
                int i,j,c;
                printf("#cmm 生成¥n");
                printf("int dummy;¥n");
                //最初に、全てのタグ関数をプロトタイプ宣言する。
                for(i=0; i<NDIC; i++){
                        printf("void ");
                        printtagname(DICTIONARY[i]);
                        printf("();¥n");
                }
                //次に、教養関数を出力する
                printf("¥nint SIORU(){¥n");
                printf("    int adress,c;¥n");
                printf("    c = GETC();");
                printf("    if(c!='¥"') exit(logf(¥"[%%c]しおる¥¥n¥",c));¥n");
                printf("    adress = SIORI();¥n");
                printf("    while(GETC()!='¥");¥n");
                printf("    return adress;¥n");
                printf("}¥n");
                printf("¥n");
                printf("int c1(){//パラ終了検出¥n");
                printf("    if(scanf(¥" />¥")) return 0;¥n");
                printf("    if(scanf(¥" >¥")) return 1;¥n");
                printf("    return -1;¥n");
                printf("}¥n");
                printf("¥n");
                printf("void c2(){¥n");
                printf("        scanf(¥"%%*[^<]¥");¥n");
                printf("}¥n");
                printf("¥n");
                printf("//個別タグ関数¥n");
                for(i=NDIC-1; 0<=i; i--){
                        printf("void ");//関数のタイトル部
                        printtagname(DICTIONARY[i]);
                        printf("(){¥n");
                        printf("¥tint ");
                        for(j=PARAMS[i];j;j=PARAMNEXT[j]){
                                printf("param%d, ", PARAMREF[j]);
                        }
                        printf("childflag;¥n");
                        printf("logf(¥"");
                        printtagname(DICTIONARY[i]);
                        printf("(%%d)¥¥n¥",SIORI());¥n");
                        printf("¥twhile((childflag=c1())<0){¥n");
                        for(j=PARAMS[i];j;j=PARAMNEXT[j]){
                                if(j==PARAMS[i]) printf("¥t¥tif(scanf(¥" ");
                                else printf("¥t¥telse if(scanf(¥" ");
                                SEEK(PARAMDIC[PARAMREF[j]]);
                                printlabel();
                                printf("=¥")) ");
                                printf("param%d", PARAMREF[j]);
```

```
                        printf("=SIORU();¥n");
                }
                printf("¥t}¥n");
                printf("¥tif(childflag==0) return;¥n");
                printf("¥t//サブ参照¥n");
                printf("¥tfor(;;){¥n");
                printf("¥t¥tc2();¥n");
                printf("¥t¥tif(scanf(¥"</");
                printtagname(DICTIONARY[i]);
                printf(">¥")) return;¥n");
                for(j=CHILDREN[i];j;j=CHILDNEXT[j]){
                        printf("¥t¥telse if(scanf(¥"<");
                        printtagname(DICTIONARY[CHILDREF[j]]);
                        printf("¥")) ");
                        printtagname(DICTIONARY[CHILDREF[j]]);
                        printf("();¥n");
                }
                printf("¥t}//endfor¥n");
                printf("}¥n¥n");
        }
        //main 関数の生成
        printf("int main(){¥n");
        printf("¥tif(scanf(¥" <?¥")) while(GETC() != '>');¥n");
        printf("¥tif(scanf(¥" <LandXML¥")) LandXML();¥n");
        printf("}¥n");
}
```

従来のファイルコンバータ等では、ある形式で記述された不特定のファイルを変換する必要があるため、オブジェクト定義やメモリ管理を行う処理系（言語）が一般的であるが、本処理系においては、処理系自体をシンプルなものとすることを目的とするため、メモリブロックやバッファも使用していない。データファイル中に使用されるタグ名称のリストは、初出位置のアドレス（整数値）として登録し、出力が必要であれば、これを用いる。更に、文字列比較の基礎的な関数である strcmp や strchr 関数も、システムの組込関数としては用意していないため、メタファイル中で関数定義している。これらは、データファイル中の初出アドレスを引数としている。

XML.cmm が出力したメタファイル・テンプレートの例をリスト 4-3-15 に示す。前掲のリスト 4-3-11 は、このテンプレートを出発点として、解析したデータ構成を配列に格納し、三次元データを合成して表示や出力の処理を行うメタファイルに仕立てたものである。

リスト 4-3-17 メタファイル・テンプレートの生成例

```
# cmm 生成
int dummy;
void LandXML();
void Units();
void Metric();
void Project();
void Application();
void Author();
void CoordinateSystem();
void CgPoints();
void CgPoint();
void Alignments();
void Alignment();
void CoordGeom();
void Line();
void Start();
void End();
void Curve();
```

```
void Center();
void CrossSects();
void CrossSect();
void DesignCrossSectSurf();
void CrossSectPnt();
void Profile();
void ProfAlign();
void PVI();

int SIORU(){
   int adress,c;
   c = GETC();   if(c!="") exit(logf("[%c]しおる¥n",c));
   adress = SIORI();
   while(GETC()!="");
   return adress;
}

int c1(){//パラ終了検出
   if(scanf(" />")) return 0;
   if(scanf(" >")) return 1;
   return -1;
}

void c2(){
        scanf("%*[^<]");
}

//個別タグ関数
void PVI(){
        int childflag;
        logf("PVI(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</PVI>")) return;
        }//endfor
}

void ProfAlign(){
        int param13, childflag;
        logf("ProfAlign(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</ProfAlign>")) return;
                else if(scanf("<PVI")) PVI();
        }//endfor
}

void Profile(){
        int param21, childflag;
        logf("Profile(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" staStart=")) param21=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</Profile>")) return;
```

```
                        else if(scanf("<ProfAlign")) ProfAlign();
        }//endfor
}

void CrossSectPnt(){
        int param27, childflag;
        logf("CrossSectPnt(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" code=")) param27=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</CrossSectPnt>")) return;
        }//endfor
}

void DesignCrossSectSurf(){
        int param13, param26, childflag;
        logf("DesignCrossSectSurf(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
                else if(scanf(" side=")) param26=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</DesignCrossSectSurf>")) return;
                else if(scanf("<CrossSectPnt")) CrossSectPnt();
        }//endfor
}

void CrossSect(){
        int param13, param25, childflag;
        logf("CrossSect(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
                else if(scanf(" sta=")) param25=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</CrossSect>")) return;
                else if(scanf("<DesignCrossSectSurf")) DesignCrossSectSurf();
        }//endfor
}

void CrossSects(){
        int childflag;
        logf("CrossSects(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</CrossSects>")) return;
                else if(scanf("<CrossSect")) CrossSect();
        }//endfor
}

void Center(){
        int childflag;
        logf("Center(%d)¥n",SIORI());
```

```
		while((childflag=c1())<0){
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Center>")) return;
		}//endfor
}

void Curve(){
		int param23, param24, childflag;
		logf("Curve(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" rot=")) param23=SIORU();
			else if(scanf(" radius=")) param24=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Curve>")) return;
			else if(scanf("<Start")) Start();
			else if(scanf("<Center")) Center();
			else if(scanf("<End")) End();
		}//endfor
}

void End(){
		int param22, childflag;
		logf("End(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" pntRef=")) param22=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</End>")) return;
		}//endfor
}

void Start(){
		int param22, childflag;
		logf("Start(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" pntRef=")) param22=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Start>")) return;
		}//endfor
}

void Line(){
		int param20, childflag;
		logf("Line(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" length=")) param20=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Line>")) return;
```

```
                        else if(scanf("<Start")) Start();
                        else if(scanf("<End")) End();
        }//endfor
}

void CoordGeom(){
        int childflag;
        logf("CoordGeom(%d)¥n",SIORI());
        while((childflag=c1())<0){
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</CoordGeom>")) return;
                else if(scanf("<Line")) Line();
                else if(scanf("<Curve")) Curve();
        }//endfor
}

void Alignment(){
        int param13, param20, param21, param14, childflag;
        logf("Alignment(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
                else if(scanf(" length=")) param20=SIORU();
                else if(scanf(" staStart=")) param21=SIORU();
                else if(scanf(" desc=")) param14=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</Alignment>")) return;
                else if(scanf("<CoordGeom")) CoordGeom();
                else if(scanf("<CrossSects")) CrossSects();
                else if(scanf("<Profile")) Profile();
        }//endfor
}

void Alignments(){
        int param13, param14, childflag;
        logf("Alignments(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
                else if(scanf(" desc=")) param14=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</Alignments>")) return;
                else if(scanf("<Alignment")) Alignment();
        }//endfor
}

void CgPoint(){
        int param13, childflag;
        logf("CgPoint(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param13=SIORU();
        }
        if(childflag==0) return;
        //サブ参照
        for(;;){
                c2();
                if(scanf("</CgPoint>")) return;
```

```
		}//endfor
}

void CgPoints(){
		int param13, childflag;
		logf("CgPoints(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param13=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</CgPoints>")) return;
			else if(scanf("<CgPoint")) CgPoint();
		}//endfor
}

void CoordinateSystem(){
		int param13, param17, param18, param19, param14, childflag;
		logf("CoordinateSystem(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param13=SIORU();
			else if(scanf(" horizontalDatum=")) param17=SIORU();
			else if(scanf(" verticalDatum=")) param18=SIORU();
			else if(scanf(" horizontalCoordinateSystemName=")) param19=SIORU();
			else if(scanf(" desc=")) param14=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</CoordinateSystem>")) return;
		}//endfor
}

void Author(){
		int param15, param16, childflag;
		logf("Author(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" createdBy=")) param15=SIORU();
			else if(scanf(" company=")) param16=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Author>")) return;
		}//endfor
}

void Application(){
		int param13, childflag;
		logf("Application(%d)¥n",SIORI());
		while((childflag=c1())<0){
			if(scanf(" name=")) param13=SIORU();
		}
		if(childflag==0) return;
		//サブ参照
		for(;;){
			c2();
			if(scanf("</Application>")) return;
			else if(scanf("<Author")) Author();
		}//endfor
}

void Project(){
```

```
	int param13, param14, childflag;
	logf("Project(%d)¥n",SIORI());
	while((childflag=c1())<0){
		if(scanf(" name=")) param13=SIORU();
		else if(scanf(" desc=")) param14=SIORU();
	}
	if(childflag==0) return;
	//サブ参照
	for(;;){
		c2();
		if(scanf("</Project>")) return;
	}//endfor
}

void Metric(){
	int param6, param7, param8, param9, param10, param11, param12, childflag;
	logf("Metric(%d)¥n",SIORI());
	while((childflag=c1())<0){
		if(scanf(" areaUnit=")) param6=SIORU();
		else if(scanf(" linearUnit=")) param7=SIORU();
		else if(scanf(" volumeUnit=")) param8=SIORU();
		else if(scanf(" temperatureUnit=")) param9=SIORU();
		else if(scanf(" pressureUnit=")) param10=SIORU();
		else if(scanf(" angularUnit=")) param11=SIORU();
		else if(scanf(" directionUnit=")) param12=SIORU();
	}
	if(childflag==0) return;
	//サブ参照
	for(;;){
		c2();
		if(scanf("</Metric>")) return;
	}//endfor
}

void Units(){
	int childflag;
	logf("Units(%d)¥n",SIORI());
	while((childflag=c1())<0){
	}
	if(childflag==0) return;
	//サブ参照
	for(;;){
		c2();
		if(scanf("</Units>")) return;
		else if(scanf("<Metric")) Metric();
	}//endfor
}

void LandXML(){
	int param0, param1, param2, param3, param4, param5, childflag;
	logf("LandXML(%d)¥n",SIORI());
	while((childflag=c1())<0){
		if(scanf(" xmlns=")) param0=SIORU();
		else if(scanf(" xmlns:xsi=")) param1=SIORU();
		else if(scanf(" date=")) param2=SIORU();
		else if(scanf(" time=")) param3=SIORU();
		else if(scanf(" version=")) param4=SIORU();
		else if(scanf(" xsi:schemaLocation=")) param5=SIORU();
	}
	if(childflag==0) return;
	//サブ参照
	for(;;){
		c2();
		if(scanf("</LandXML>")) return;
		else if(scanf("<Units")) Units();
		else if(scanf("<Project")) Project();
		else if(scanf("<Application")) Application();
```

```
                        else if(scanf("<CoordinateSystem")) CoordinateSystem();
                        else if(scanf("<CgPoints")) CgPoints();
                        else if(scanf("<Alignments")) Alignments();
        }//endfor
}

int main(){
        if(scanf(" <?")) while(GETC() != '>');
        if(scanf(" <LandXML")) LandXML();
}
```

XML.cmm を、LandXML-1.xml に適用して出力されてテンプレートを基に、形状定義にとって意味のあるタグの内部に記述されている座標値などのデータを抽出し、断面と軌跡から三次元的な形状生成の処理を行うためのメタファイル LandXML1.cmm を作成する。これを、別のデータファイル LandXML-2.xml に対して適用したときに、LandXML-1.xml には用いられていなかったパラメータやタグが使用されていると、エンドレスループとなる。これを防ぐために、for(;;)ループの最後に、`else abort();`という 1 行を加えるとともに、abort()関数を冒頭に用意し、エラーが生じた行の周辺を表示するとともに、強制終了すると、プログラムの修正するために便利である（リスト 4-3-16)。このようにして 8 のサンプルに適用して拡張した共通メタファイルがリスト 4-3-11 である。

リスト 4-3-18 拡張とデバッグのために追加した abort() 関数

```
void abort(){
   int i,c,siori,lc;
   siori   = SIORI();
    SEEK(0);
    for(lc=i=0;i<siori;i++){
        c = GETC();
        if(c == '¥n') lc++;
   }
   printf("xxxxxxxxxxxxxxxxx abort at line %d xxxxxxxxxxxxxxxxxxx¥n{", lc+1 );
   SEEK(siori-10);
   for(i=0;i<20;i++){
        if(i==10) printf("?");
        c = GETC();
        if(c < 32) printf("<%x>",c);
        else printf("%c", c);
   }
   printf("}¥n");
   exit(3);
}

/* 以下は、デバッグのために abort()関数呼び出しを追記した関数の例
void DesignCrossSectSurf(){
        int param6, param25, param7, childflag;
        int pmaterial, ptypicalThickness,pclosedArea;
        logf("#DesignCrossSectSurf(%d)¥n",SIORI());
        while((childflag=c1())<0){
                if(scanf(" name=")) param6=SIORU();
                else if(scanf(" side=")) param25=SIORU();
                else if(scanf(" desc=")) param7=SIORU();
                else if(scanf(" material=")) pmaterial=SIORU();
                else if(scanf(" typicalThickness=")) ptypicalThickness=SIORU();
                else if(scanf(" closedArea=")) pclosedArea=SIORU();
                else abort();
        }
        if(childflag==0) return;
        //shori
        DesignCrossSectSurf.name[NDesignCrossSectSurf] = param6;
        DesignCrossSectSurf.side[NDesignCrossSectSurf] = param25;
        DesignCrossSectSurf.desc[NDesignCrossSectSurf] = param7;
```

```
		DesignCrossSectSurf.CrossSectPnt[NDesignCrossSectSurf] = NCrossSectPnt;
		//サブ参照
		for(;;){//通常は、二つの CrossSectPnt を有する
			c2();
			if(scanf("</DesignCrossSectSurf>")) break;
			else if(scanf("<CrossSectPnt")) CrossSectPnt();
			else if(scanf("<Feature")) Feature();
			else abort();
		}//endfor
		//ループ終了後の NCrossSectPnt は、
		//DesignCrossSectSurf.CrossSectPnt[NDesignCrossSectSurf+1]に記録される
		printf("# NDesignCrossSectSurf=%d¥n", NDesignCrossSectSurf);
		NDesignCrossSectSurf++;
}
```

（８）その他のファイル形式

三次元データ形式には様々なものがあり、（７）までに記した各種ファイル形式と同様の方法でメタファイルを作成することが可能である。少し性格の異なる形式が存在するため若干補足しておく。

①コンテナ形式

特定のフォーマットによる三次元データではなく、各種形式のファイルを束ねてアーカイブしたファイル形式であり、代表的なものに pdf 形式や zip 形式がある。

通常はこのようなデータファイルを処理する際には、梱包を解いて個別のファイルに展開した後に解読処理を行うが、本処理系においてはそのようなバッファリングを用意していない。よって、本処理系を用いて保存すべきデータとしては適当ではない。

②圧縮形式

データファイルを、あるアルゴリズムに従って圧縮したファイル形式である。①の ZIP 形式の場合には、Deflate 法式(RFC1951)に従って圧縮を掛けたものが多い。

通常はこのようなデータファイルを処理する際には、解凍した後に解読処理を行うが、本処理系においてはそのようなバッファリングを用意していない。よって、本処理系を用いて保存すべきデータとしては適当ではない。

③暗号化形式

データファイルを、ある鍵がなければ解読できない形に暗号化した形式である。

⑤コメント埋め込み形式

ある著名なファイル形式において、コメントとして形式やサイズが束縛されない、通常の処理系では無視される領域に意図的なデータを埋め込んだ形式である。本処理系では、このようなデータを解読するのに適している。

４－４．空間座標と携帯端末

（１）座標系とその変換の表現

長期保存のために作成した三次元データによる記録の将来における利活用を例示する一形態として本研究の中で国総研が作成したＶＣ－３Ｍの処理系においては、携帯端末の位置・姿勢検出及びメタファイルにおけるシーングラフ型データ構造における局所的な位置座標の問題に関して、旧来の景観シミュレーション・システムにはなかった処理を行っているため、旧システムのデータ構造との関係を含めて、幾何学的な解説を行う。

① 携帯端末の姿勢の表現

携帯端末を剛体とみなすと、位置に関して３、姿勢に関して３の、合計６の自由度を有している。

一方、背景と三次元データを合成描画する OpenGL ライブラリによる景観シミュレーション表示処理に用いられる CAM 構造体を、本処理系では最終的な描画処理に使用している。

リスト 4-4-1 CAM 構造体の定義

```
typedef struct _s3Camera {
        char    *name;
        double  eye[3];
        double  center[3];
        double  up[3];
        double  fovy, aspect, zNear, zFar;
} s3Camera;
```

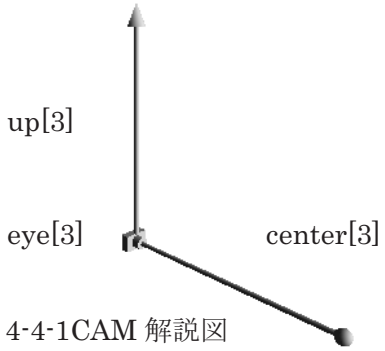


図 4-4-1CAM 解説図

ここで、eye[3]は、視点の位置を、モデルと同じ座標系で記述した X,Y,Z の座標値を示し、通常はメートルを単位として、東を X 軸、北を Y 軸、上方を Z 軸としている。背面カメラが撮影した画像と正しく位置合わせするためには、携帯端末の背面カメラのレンズ中心の座標値を用いる。

center[3]は、注視点つまり、カメラが向いている方向で、上記の背面カメラの中心から正面に向かった延長線（半直線）上の１点の座標値である。この点が半直線上のどこにあっても、表示は等しい。注視点を中心として回転するような視点移動を行うような場合には、描画ではなく、視点移動における移動先を決定するために注視点までの距離に意味がある。

up[3]は、携帯端末の画面における上方を示すベクトルである。

視点と注視点を定めただけでは、携帯端末の姿勢は一意に定まらず、カメラの中心軸の周りに回転するような自由度が残る。この上方ベクトルを定めることにより、携帯端末の姿勢は一意に決定する（図 4-4-1）。

位置の自由度３は、eye[3]と一致している。

姿勢の自由度３は、center[3]と up[3]の６変数で記述されており、３の冗長性がある。３の余分な自由度は、center[3]-eye[3]の絶対値を１とし、up[3]の絶対値を１とし、up[3]を、center[3]-eye[3]と直交させることにより、解消することができる。

ここで、カメラに固定したローカルな座標系を考えると、center、up およびこれらの外積となる center×up は､カメラに固定した座標軸と見ることができ､これらのベクトルは、カメラの座標軸を土地の座標系で記述したものであり、更にこれらのベクトルを並べて３×３の行列を作成すると、回転行列によりカメラ姿勢を表現できる。

各種内蔵センサで計測した携帯端末の位置・姿勢を用いて画面表示を行う VC-3M の場合には、センサが取得した緯度、経度、標高および端末の姿勢、即ち携帯端末固有の座標系で記述した地球の位置と姿勢から、敷地や建物を表示している座標系で表示した携帯端末の位置と姿勢を算出し、CAM 形式のデータとして表示処理系に渡している。

② ローカル座標を用いた階層的な空間記述における移動・回転・スケール

全ての位置と形状を同じグローバル座標を用いて団地や集落の空間を記述する方法は GIS 等のシステムでよく行われる。しかしこの場合、ミリ単位の精度で記述する小さな家具の形状にも数十ｋｍのオーダーの座標値を用いることになり、メモリ使用効率やデータのハンドリングの効率が下がる。

空間を段階的に構成することにより、この問題を回避することができる。建物の形状を、CAD でよく行われるように平面図の通り芯左下の地表の点を原点とする座標系で記述することにより、建物のローカルな座標系で見通し良く作成することができる。これを敷地や団地の中に配置する際には、敷地や団地の座標系（世界測地系を用いた平面直角座標系がよく用いられる）で表現した、建物の配置位置と姿勢（方位）を指定することにより、正しい位置に表示させることができる。

更に、住宅内部に配置される家具等についても、家具の製造工程で使用される直角座標系で記述された形状ファイルが使用できるならばこれを用いて、上記の建物の座標系の中に原点位置と回転を指定して配置することにより、正しい位置に表示させることができる。

同形の家具等を複数個配置する場合に、一つの家具の形状データだけを用意しておき、これを必要な個数、異なる場所と向きに配置することにより複数個を表示することができ、データを軽くし、部品の形状を統一することができる。

このような相対的な配置の指定に際しては、以下の３つの要素を指定することができる。

1）移動（TRANSLATE）

配置する位置を指定するために、下位（部分）を記述する座標系の原点を、上位（全体）を記述する座標系３値(T_x, T_y, T_z)で表記する。

$x' = T_x + x$

$y' = T_y + y$

$z' = T_z + z$

2）回転（ROTATE）

配置する向きを指定するために、下位（部分）を記述する座標系を、上位（全体）を記述する座標系に変換するパラメータで表記する。回転に関しては、④で示すように、いくつかの表現方法がある。各方法の間での相互変換に関して、⑥で解説する。

座標系の回転、言い換えると剛体の回転の自由度は３であるが、広く用いられている回転行列では、９の数値が用いられており、冗長な表記方法となっている。

$V' = M V$

M[3][3]の各列は、部分を記述する座標系の各軸の単位ベクトルを、全体を記述する座標系の各軸成分に分解して表現したものであり、ベクトルとしての長さは１である。異なる列は互いに直交する。M の逆行列 M^{-1} は M の転置行列 ${}^{t}M$ である。言い換えると Mij と $M^{-1}ji$ は、共に部分座標系の j 軸と、全体座標系の i 軸の単位ベクトルの内積である。

3）スケール（SCALE）

配置する際に、拡大縮小を掛ける場合がある。また、部品の形状を記述している尺度の単位（例えばミリメートル）が、全体の形状を記述している尺度の単位（例えばメートル）と異なっているような場合もある。

スケールは、３軸に関して独立して指定することができるため、３の自由度がある。

$x' = S_x * x$

$y' = S_y * y$

$z' = S_z * z$

一般に滑り変形など、物体の変形（アフィン変換）は歪と同様、その変形における拡大・縮小の主軸が座標軸と一致するとは限らず、回転と拡大・縮小を組み合わせて表現する必要がある。

③ 同次行列

本処理系を含め、三次元 CG においては、相対的な位置関係を記述するために、三次元座標に定数項１を加えた同次座標(homogeneous coordinates)と、４×４の同次行列(homogeneous matrix)が広く用いられる。その理由の一つに、４×４の行列の数値計算が、描画のためのライブラリの中に用意されており、最近では GPU の普及に伴い高速に計算できるようになってきたことがある。機械の関節運動の表現等にも用いられている。

座標値に定数１を加えた４元のベクトル V を用意しておき、同次行列をかけることにより、移動・回転・スケールを一つの行列計算で処理することができる。更に、座標変換を累積的に行うためには、行列の積を計算するだけでよい。

これにより、例えば

集落→住宅→建物の部分や家具

という段階的な構成として居住空間を記述する場合に、建物の部分や家具は、それぞれの人体に近いスケールにおける物体の座標系を用いた記述を行い、次にこれを住宅の各部分に配置する際に、配置する位置と向きを示す同次行列をかけて、住宅の座標系を用いた記述に変換し、更に複数の住宅を集落の座標系の中に配置するための相対位置関係を記述した同次行列をかけて、集落全体の空間を構成することができる。

この方法の長所は、処理系内部における計算処理をコンパクトに記述できることにあるが、建築物の場合には、平行移動と Z 軸に回転するだけの配置も頻繁に行われるため配置

の定義スクリプトや数値計算に際して１６の数値を全て使用することは冗長な場合がある。

行列を用いた移動・回転・スケールの表現は、数式表現の見通しが良く、また行列演算パッケージがグラフィクスライブラリである OpenGL や、最近では高速演算処理専用 IC チップ GPU の中にも用意されており、プログラミングに便利である。

しかしその一方で、移動（自由度３）、回転（自由度３）、スケール（自由度３）を常に４×４の行列として表現するために、冗長である。このため、スケールの変化を伴わない回転を多数回繰り返し行う行列の乗算処理の後に、計算誤差がスケール成分として累積する。

そこで、⑥で、行列から、回転成分、拡大縮小成分を抽出する方法について解説し、⑦においては、本処理系において新たに導入した代替的な方法として、四元数による姿勢の表現方法について解説する。

二次元座標系の場合、３×３の同次行列を用いて移動回転スケールを集約する。

$$\begin{bmatrix} x' \\ y' \\ m' \end{bmatrix} = Tr \cdot Tm \cdot \begin{bmatrix} x \\ y \\ m \end{bmatrix}$$ (m, m'は単位長さで、通常は 1。この場合、平行移動してから回転)

$$Tr = \left[\begin{array}{cc:c} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ \hdashline 0 & 0 & 1 \end{array}\right]$$

（Tr の各行は、回転後の各座標軸単位ベクトルを、回転前の座標系で表示したものである。）

$$Tm = \left[\begin{array}{cc:c} 1 & 0 & Dx \\ 0 & 1 & Dy \\ \hdashline 0 & 0 & 1 \end{array}\right]$$

$x' = x\cos\theta - y\sin\theta + Dx$

$y' = x\sin\theta + y\cos\theta + Dy$

三次元の場合、位置座標をの X,Y,Z に定数 1 を加えた四元ベクトルを、

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

と置き、移動・回転・スケールの変換を４×４の同次行列として表現する。

$$\left[\begin{array}{c:c} M[3][3] & D \\ \hdashline 0 & 1 \end{array}\right]$$

と置くことで、同様の処理を行う。

本処理系では、４×４の同次行列を、長さ 16 の倍精度浮動小数の一次元配列として表現している。この時、配列 M[16]（添え字は 0~15）と表記すると、本処理系では次の配置としている。

$$\begin{bmatrix} M[0] & M[4] & M[8] & M[12] \\ M[1] & M[5] & M[9] & M[13] \\ M[2] & M[6] & M[10] & M[14] \\ M[3] & M[7] & M[11] & M[15] \end{bmatrix}$$

ここで、M[3],M[7],M[11]は、座標値のベクトルに対する座標変換に際しては意味を持たない。また M[15]は常に１であってよい。座標変換を重ねる際には、M[3],M[7],m[11]に意味のない誤差などの数値があると、結果の M[15]が 1.0 とはならない。

景観シミュレータ(1996)の dml ライブラリ関数 d3TransformPointd(double*p1, double*p2, d3Matrix m)において同次行列[4×4]とベクトル[3 と定数１]の積を行う際には、第４行の成分であるベクトル(M[3],M[7],M[11])と変換前のベクトル p2 との内積に M[15]を加えた計算値 w で、変換後のベクトルの x,y,z 各値を除している。これは、空間→第４次元の変換を考えた時のベクトル(M[3],M[7],M[11])を軸とした変形（潰れ、ローレンツ収縮のような効果）に相当し、特別な場合としてベクトルが有限値、M[15]がゼロの場合には立体が一平面（描画面）上に潰れるため、この軸を視点－注視点ベクトルとした透視投影変換（カメラ軸に垂直な面へのパース描画）を行うために利用されている。多くの場合、M[11]のみ１とし、M[15]を０として、XY 平面への描画を行う。

景観シミュレータにおいては、親グループ（例えば団地や集落）に対して、子グループ（例えば個々の住宅建物）が配置されてその間にリンクが定義されると、移動や回転の有無に関わりなく、内部的にはリンク構造体に、この１６の倍精度実数からなる配列としてリンク・行列を設定してきた（初期値は、単位行列)。

一度移動・回転・縮尺を行った物体を、更に移動・回転する場合、既に設定されているリンク・行列に対する積を求める。この時、行列を掛ける順序により結果が異なる。例えば、平行移動行列 M_t と回転の行列 M_r を掛け合わせた M_tM_r は先に回転を行ってから平行移動を行う効果がある。この場合回転は部分の座標系に即して行われる。これに対して、M_rM_t は先に平行移動してから回転する。この場合、回転は全体の座標系の原点を中心として行われる。よって、部分が全体の中に配置される位置が変化する。行列の積を前から行うか、後から行うかを、LINK_XFORM コマンドの第二引数である PRE または POST が指定している。

同様に拡大縮小に関しても、M_tM_a は、部分の座標系の原点を中心とした拡大縮小であるが、M_aM_t は、全体の座標系の原点を中心とした拡大縮小となるため、結果が異なる。

回転を二回行う場合、$M_{r1}M_{r2}$ と、$M_{r2}M_{r1}$ は一般に異なる結果となる。

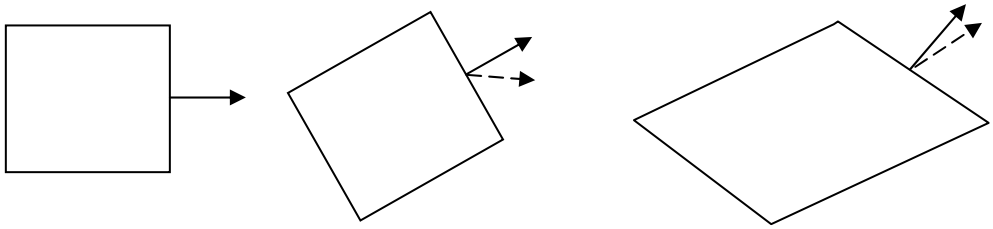


図 4-4-2　法線ベクトルの変換（破線：変換前、実線：変換後）

リンク・行列は、主に座標値の変換に使用するが、データに「法線ベクトル」が設定されている場合、座標値の変換と同時に法線ベクトルも変換して、正しい表示を得る。平行移動では、法線ベクトルを変化させない。回転は、座標変換と同じ変換を行う。スケール変換では、法線ベクトルの各軸成分に、座標軸のスケールの逆数を乗じた上で長さ１に正規化する。

位置座標の変換に関して回転とスケールを合成した行列 M から、法線ベクトルの変換のための行列を求めるためには、逆行列の転置を行う。

[簡単な証明]
面を構成する任意のベクトル V と法線ベクトル N は常に直交する。V と N を３行１列の行列として、また V を転置した１列３行の行列を ${}^{t}V$ と表現すると、

$${}^{t}V\,N \equiv (0)$$ （ベクトルの内積の行列表現、(０)は１×１の行列で唯一の成分が０）

の関係が常に成立する。

位置座標の変換行列を M、逆行列の転置を ${}^{t}M^{-1}$ と表すと、変換後の V と N はそれぞれ、MV、${}^{t}M^{-1}\,N$ と表せ、この内積は、

$${}^{t}(MV)\,({}^{t}M^{-1}N) = {}^{t}V\,{}^{t}M\;\;{}^{t}M^{-1}N = {}^{t}V\,({}^{t}M\;\;{}^{t}M^{-1})N = {}^{t}V\,{}^{t}(M^{-1}M)N = {}^{t}V\,{}^{t}I\,N$$

$$= {}^{t}V\,I\,N = {}^{t}V\,N \equiv (0)$$

よって、V と N が直交する関係は、変換後も保たれる。

④ LINK_XFORM コマンドによる表現

景観シミュレータのための LSS-G 形式においては、上位の座標系で記述された領域（敷地や団地）の中に下位の座標系で記述された要素（建物等）を配置する定義を LINK コマンドで記述し、その際の位置関係（移動・回転・スケール）を記述するために、LINK_XFORM コマンドを使用した。メタファイルの記述においても、この二つのコマンドをライブラリ関数として用意している。

上位の要素の中に下位の要素を関係づける配置操作は、

```
リンク名称＝LINK(上位の要素,下位の要素);
```

というコマンド形式で定義する。

次に LINK_XFORM コマンドの第一引数にこのリンク名称を用いて、次の形式で移動・回転・スケールの関係を記述する。

```
LINK_XFORM(リンク名称, 適用方法, 記述方法, 引数群);
```

この LINK_XFORM は、同じリンク名称に対して累積的に適用することができ、まず回転してから移動する、というように複数回組み合わせた定義を行うことができる。

LSS-G 形式においては、引数群は浮動小数の固定値を用いたが、メタファイルにおいては、変数や数式を用いることができ、x,y,z の３値の変わりに一つの四元数 q に集約して記述することもできる。

適用方法は、LOAD、PRE、POST の３種類のキーワードから選択し、LOAD では、既存の同次行列への上書きを行う。PRE では、既存の同次行列の後に新たな同次行列を乗じる。POST では、既存の同次行列の前に新たな同次行列を乗じる。例えば、既に回転が定義されているリンクに対して、移動を LOAD で適用すると回転はキャンセルされ、PRE で適用すると「まず移動してから回転」、POST で適用すると、「まず回転してから移動」という

結果が得られる。LINK_XFORM ライブラリ関数を実行する内部処理においては、同次行列の乗算を行っている。

全体に対する部分の位置と姿勢、つまり座標変換の記述方法として、上記の１６の成分を直接記述する MATRIX の代わりに、平行移動、回転、スケールを単独で定義しリンク・行列に累加するコマンドとして、

1) IDENTITY (単位行列を定義し、引数はない。再度初期化する場合などに使用する。)
2) TRANSLATE(平行移動を定義し、引数は XYZ の各座標の３個の浮動小数値である)
3) ROTATE_X(X 軸周りの回転で引数は度(0-360)で表記した回転角度を表す浮動小数値)
   ROTATE_Y(同上)
   ROTATE_Z(同上)
4) ROTATE_A(任意の軸周りの回転で、引数は軸を表す３個と回転角１個の浮動小数値)
5) SCALE(XYZ の各座標軸に関する倍率を表す引数として浮動小数値３個)
6) MATRIX（同次行列の要素として浮動小数値１６個))

を使用することができる。

リンク情報が含まれるシーンを景観シミュレータにおいて LSS-G 形式で保存すると、単純に、MATRIX 形式でリンク情報が出力される。但し、デフォルト値に等しい単位行列に等しい場合にのみ省略される。

リスト 4-4-2 MATRIX 形式によるリンク情報の出力

```
GRP23 = FILE("wall-nb1.geo");
K38 = LINK(GRP23.org, GRP23);
LINK_XFORM(K38, LOAD, MATRIX, 1, 0, 0, 0,
                0, 1, 0, 0, 0, 0,
                1, 0, 12, 0, 7.14, 1);
```

一方、建築物により構成された空間の場合には、平行移動(TRANSLATE)と Z 軸まわりの回転(ROTATE_Z)だけを適用することで足りる場合が多い。例えば、景観シミュレータのサンプルデータとして古くからセットアップに含めている landmark.geo は、Attract というソフトウェアでモデリングしたデータをコンバータで変換したものであり、コンバータが自動生成・出力する LINK_XFORM コマンドでは、ROTATE_X、TRANSLATE を用いて回転と並進を定義している。このように、ファイル・コンバータが出力する LSSG ファイルの場合には、変換元のファイル形式に応じて、使いやすい並進・回転の指定方法が用いられてきた。

リスト 4-4-3 ROTATE と TRANSLATE によるリンク情報の出力

```
L000001 = LINK(ROOT, G000001);
LINK_XFORM(L000001, POST, ROTATE_X, -90.000000);
LINK_XFORM(L000001, POST, TRANSLATE, 0.000000, 0.000000, -55.000000);
LINK_XFORM(L000001, POST, ROTATE_X, 90.000000);
```

記述方法別のパラメータの与え方は以下の通りである。

1) IDENTITY

単位行列（移動回転スケールなし）を指定する。適用方法が PRE、POST ならば何も変化はなく、LOAD ならば既に定義されている同次行列が単位行列に初期化される。

内部処理においては、LINK コマンドが実行された段階で、同次行列が定義され、単位行列に初期化される。従って、この IDENTITY を記述方法とした LINK_XFORM コマンドを常に最初に実行する必要はない。

2) TRANSLATE

平行移動を、XYZ の３軸のオフセットを、浮動小数３個の引数で記述する。

3) ROTATE_X, ROTATE_Y, ROTATE_Z

座標軸を回転軸とする回転であり、角度（度）を示す浮動小数１個を引数とする。

航空機等の姿勢を示す表現では、ロールは機体の軸まわりの回転（傾き）、ピッチは上昇下降に関わる回転、ヨー（アジマス）は、進行方向を示す鉛直軸まわりの回転である。

この概念は、携帯端末の姿勢を示す API 関数においても用いられている。携帯端末に固有の座標軸を、画面上方を X 軸、画面右方を Y 軸、画面に垂直な軸を Z 軸とすると、端末を水平に、画面上方を北に向けて置いた姿勢を出発点として、まずヨー角だけ方位を変え（水平回転し）、ピッチ角だけ俯角仰角を変え（縦回転し）、最後にロール角だけ画面を回転させることにより最終的な姿勢が得られる。順序が異なると、最終的な姿勢は異なる。

座標軸の一つを回転軸とする回転は、二次元の回転と同等の計算で処理可能である。よって、パラメータはそれぞれ一次元である。仮想コンバータのコマンドにおいては、

LINK_XFORM(リンク名称, 適用方法, ROTATE_X, 30.0);

が、名称で指定したリンクに対して、X 軸に関して 30° の回転を与えることを示す。この角度は、軸を上から見た時に反時計回りの向きの回転である。

携帯端末のセンサに基づいて OS 側でロール、ピッチ、ヨーを計算した値を簡便に取り出すライブラリ関数が開発環境側の API として用意されており、初期の開発用機種が固定されていた段階で使用した（Android 上で java 言語が使用する SensorEvent ライブラリクラスの values メンバとして取得し、OrientationSensor の values[0]がヨー、values[1]がピッチ、values[2]がロール）。しかしながら、端末の角度による上方軸の自動切り替えの機能等が適用されて複雑な動作を示し、またそのような動作の機種による互換性が低いことから、本処理系においては、加速度センサ(AccSensor)および磁気センサ(MagSensor)からの計測値を直接取得し、アプリ内部で上方と北を使用する方法に改めている。

4) ROTATE_Z

任意の三次元回転（自由度３）は、回転軸（自由度２）と回転角（自由度１）によって表せることから、

LINK_XFORM(リンク名称, 適用方法, ROTATE_A, Ax, Ay, Az, t);

と表せる。Ax,Ay,Az は、回転の軸を表す単位ベクトルの各成分を表し、t は回転角を度で表す。回転軸の先から根元を見た時に、反時計まわり（右螺子）の角度である。

ROTATE_A, 0, 0, 1, 30

は、

ROTATE_Z, 30

と同等の効果がある。

この表現方法は、３.で解説する四元数の乗算として回転を表現する方法とほぼ同等である。

5) SCALE

LINK_XFORM(リンク名称, 適用方法, SCALE, S_x,S_y, S_z);

S_x, S_y, S_z は座標軸毎のスケール（倍率）を記述する。**（２）**で後述するように、座標軸とは一致しない斜めの主軸に関してスケールを掛ける場合には、回転とスケールを組み合わせて適用する必要がある。

6) MATRIX

LINK_XFORM(リンク名称, 適用方法, MATRIX, m[0],m[1],・・・,m[15]);

同次行列を構成する１６の成分を全て列挙して定義する方法である。

⑤ 同次行列の合成

移動、回転、スケールの個別の定義からリンク・行列を求める方法は以下の通り。

1) 単位行列(IDENTITY)

対角成分(添字 0,5,10,15）のみ 1.0 とし、それ以外を 0.0 とする。

2) 平行移動(TRANSLATE)

同次行列全体を、単位行列に初期化する（回転はなくスケールは３軸とも 1.0)。平行移動を(Dx,Dy,Dz)とすると、以下の成分にこれを代入する。

M[13] = T_x

M[14] = T_y

M[15] = T_z

次のコマンド表現と同等である。

LINK_XFORM(L,LOAD,TRANSLATE, T_x, T_y, T_z);,

3) 各軸周りの回転(ROTATE_X, ROTATE_Y, ROTATE_Z)

XYZ 各軸まわりの回転は、二次元回転として処理可能である。

回転角を θ （度）とすると、X 軸周りの回転を行うと、Y－Z 平面での回転が生じる。

$y' = y \cos\theta - z \sin\theta$

$z' = y \sin\theta + z \cos\theta$

$M[5]=M[10]=\cos\theta$

$M[6]=\sin\theta$, $M[9]= -\sin\theta$

これは次のコマンド表現と同等である。

LINK_XFORM(L,LOAD,ROTATE_X, θ);

同様に、Y 軸周り、Z 軸周りの回転は次のように記述できる。

LINK_XFORM(L,LOAD,ROTATE_Y, θ);

LINK_XFORM(L,LOAD,ROTATE_Z, θ);

4) 任意の回転軸周りの回転(ROTATE_A)

LINK_XFORM(L,LOAD,ROTATE_A, Ax, Ay, Az, θ);

回転軸を単位ベクトル Ax, Ay, Az で表現し、$C = \cos\theta$, $S = \sin\theta$ としたとき、リンク・行列は以下のようになる。

$$\begin{bmatrix} C+Ax^2*(1-C); & Ax*Ay*(1-C)+Az*S & Az*Ax*(1-C)-Ay*S & 0 \\ Ax*Ay*(1-C)-Az*S & C+Ay^2*(1-C); & Ay*Az*(1-C)+Ax*S & 0 \\ Az*Ax*(1-C)+Ay*s & Ay*Az*(1-C)-Ax*S & C + Az^2*(1-C); & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5) スケール(SCALE)

行列の対角成分以外を 0 に初期化した上で、各軸のスケールを Sx,Sy,Sz とすると、

M[0] = S_x

M[5] = S_y

M[10] = S_z

これは、次のコマンド表現と同等である。

LINK_XFORM(L,LOAD,SCALE, S_x, S_y, S_z);

特殊な場合として、ある軸に関するスケールが 0 である場合、次元の縮小が生じ、空間的な図形が平面的な図形に変換される。Z 軸に関してスケール 0 を掛けることにより、平面図のような図形が得られる。

また、スケールを -1 とすることにより、鏡像変換した図形が得られる。二つの軸に関して -1 のスケールを掛けると、残りの軸に関して 180° 回転したものと同じ図形となる（即ち、結果的に同じリンク・行列が算出される)。また、三軸に関して -1 のスケールを掛けると、点対称の図形となり、この時元の図形に対しては、鏡像の関係となる。

6)行列(MATRIX)

LINK_XFORM(L,LOAD,MATRIX, m0, m1, ・・・,m15);

引数として与えられた 16 の数値をそのまま行列に代入する。

⑥ リンク行列からの要素抽出

移動・回転・スケールを常にリンク行列で表現する方法は、各処理系における内部処理プログラムを単純なものとするが、建物の CAD データ等の場合には、回転・スケールを伴わない移動だけの座標変換も多く出現する。また、スケール変換を伴わない回転や、Z 軸まわりの回転だけの配置が行われる場合も多い。従って、保存ファイル等に常にリンク・行列の全要素を出力すると、ファイルが冗長なものとなる。そこで、リンク・行列が単位行列である場合のリンク・行列の出力の省略や、回転だけの場合の表現ができると、同一内容で、より簡潔な保存データを作成することができる。

リンク・行列から、移動、回転、スケールの要素を取り出すためには、以下のようにする。

1) 移動

移動は、リンク・行列表現において移動は、４列の１～３行の成分として表現されてい

る。位置座標を４×１行列で表現した第四成分（常に１）に乗じて、変換後の位置座標の各成分に加算される。よって、リンク行列 M を、移動のみを表現する Mt と回転・スケールを表現する Mrs の積に分解することができ、M＝Mt Mrs である。

2) スケール

回転とスケールは、リンク行列４×４の内、第１～３行、第１～３列の正方形部分で表現されている。これは、回転を表す二つの行列 P,R と、スケールを表す行列 D の積 PDR として分解される。

回転だけを表す P,R は、$P^{-1} = {}^tP$, $R^{-1} = {}^tR$ という性質（列は転置に等しい）を有する。

また、スケールを表現した D は、対角成分以外はゼロで、対角成分が各軸の拡大率である。

スケールを求めるためには、３行３列の左上部分を M とした時に、

M と転置行列の積の固有値を計算する。

$${}^tM\,M = {}^t(PDR)\,PDR = {}^tR\,{}^tD\,{}^tP\,PDR = {}^tR\,{}^tDD\,R = {}^tRD^2R$$

この固有値の算出は、固有方程式

$$|M - \lambda I| = 0$$

の解を求める問題であって、一般には固有値λに関する三次方程式となる。Cardano の方法により、再帰的計算を行わなくとも、解析的に解くことができ、立方根（x^(1.0/3.0)を用いた数値計算に帰着させることができる。

この方法により求めた対角行列D^2からDを求めるには、単に対角行列の各成分の平方根を求めればよい。体積が負値となるような同次行列に関して正値（sqrt）を用いた場合、これを前提とした回転行列が鏡像反転を含むものとなる。スケールの３値の内、１または３の値が負である場合、このような状況となる。このような場合は、Mの行列式（すなわち体積倍率）が負値であることで判定できるから、D^2からDを求める計算において、一つまたは全ての値を負値とする必要がある。

3) 回転

スケールの３パラメータを求めることにより、スケールを表す対角行列D^2を求めることができる。これを用いて、回転を表す行列の一つである R を求めることができ、更にもう一つの行列 P を求めることができる。

これにより、任意の同次行列 M は、

$$M=M_TM_PM_DM_R$$

（但し、M_Tは移動、M_Pは第二の回転、M_Dはスケール、M_Rは第一の回転）

の形に分解される。

この分解における第一の回転は、スケールの３軸をそれぞれどの座標軸と一致させるかに関して、順列６通りの選択が可能であり、更に３座標軸の向きで８通り、相乗４８通りの選択が可能である。この内半数の２４は立方体の回転群、残りはその鏡像である。従って、計算を行うためには、適切な座標軸の選択を行う実用的な手順を工夫する必要がある。

上記 2)と 3)の具体的な計算方法とプログラムについては（２）で解説する。

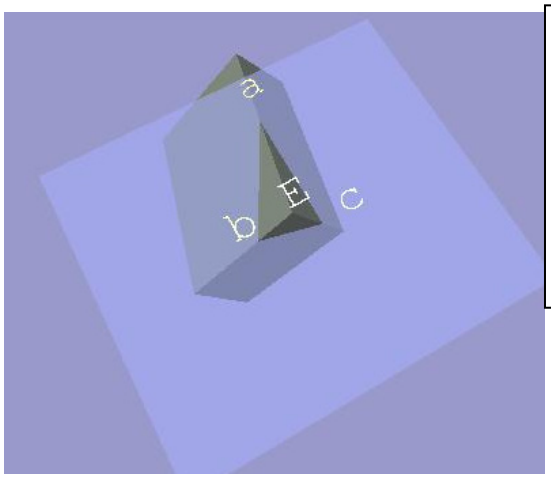
スケールがない場合（M_D が単位行列の場合）には、同次行列の左上の３×３の部分は純粋な三次元回転を表す行列である。

⑦ 逆転写としての古写真からの立体的形状の復原について

OpenGL 等のライブラリを用いた描画は、基本的には三次元形状に関するデータと、視点情報を用いた透視図（パース）の作成を行う射影変換処理である。一方、立体を撮影した写真は、カメラを用いて三次元形状を記述したデータであり、写真から視点情報と立体の形状に関する情報を取り出すことができる。

自由形状の地形等については、一般にはステレオ・ペアの２枚の画像において、対応する地物を抽出することにより解析が行われる（解析図化）。

多く直線的で直交する要素により構成された、建築物の写真の場合には、１枚の写真から形状を抽出することができる。この処理手順を組み込んで、写真を用いる建築物の復原的モデリングを行うソフトウェアは多数存在する。国土交通省版景観シミュレーション・システムにおいても、このようなソフトウェアの一つである「Real Modeler」を用いて写真から福島市、千葉市幕張の現況町並を計測するために使用した（1996 年、2-3(1)②、4-3(1)①参照）。また本研究では、奥尻島の 1993 年被災前の古写真から復原した被災前の住宅等の建物から、町並を再現した。詳細は省略するが、平行する直線群が交わる焦点を最低２点抽出することに成功すれば、これに基づいて水平・垂直の要素に関する位置を割り出していく。更に、ドア・窓サッシュや交通標識など、寸法を容易に推定できる要素から、写真の焦点距離（撮影時のズーム、引き伸ばし倍率）を求めて、建物部分等の絶対寸法を割り出す。このことを図 4-4-3 で簡単に解説した。



i’

①直方体の平行な辺の画面上の延長線の交点から消点 A,B 等を求め、これに基づいて、画像中心上の視点Ｅの撮像面からの高さα（焦点距離）を求める。
②直方体の原点 O を撮像面上とまず仮定し、ここから例えばＥ-A に平行な辺の終点 Xi の三次元座標を、Ｅ-A に平行な半直線 O-X_i と半直線 E-X_i’の延長線との交点として求める。同様の作業を繰り返す。
③最後に、実寸法が確認できる被写体部分から、復原モデルの倍率を計算する。

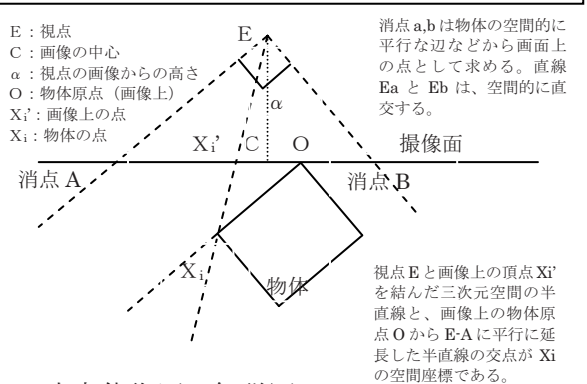


図 4-4-3　透視図からの直方体復原の解説図

より精密に形状を復原するためには、撮影に使用したレンズの収差等を補正（キャリブレーション）する必要がある。一般には、古写真に関しては、撮影に使用したカメラに関する情報は失われている。上記のソフトウェアにおいては、直線的な被写体部分の曲がりや歪みからレンズ収差を推定するアルゴリズムを内蔵していた。

（２）同次行列の要素への分解手順

一般にリンクに対して定義されている同次行列は、移動・回転・スケールが合成されたものであり、それぞれの要素を分解して取り出すことにより、見通しの良いデータを作成する、あるいは誤差を検出する、といった処理を行うことができる。例えば、スケール（変形）を伴わない同次行列から抽出されるスケール成分は、誤差である。また、敷地の中に方位（Z 軸まわりの回転）だけが定義されている建物の配置に関するリンクから抽出されるX 軸回り、Y 軸回りの回転は、誤差である。

①　歪テンソルと同次行列のスケール要素との関係

歪の実体は、３軸の伸縮である。

任意の歪テンソル A は、

$$A=QU=VQ$$

の形式で表現される。ここに、U、V は正値対称テンソル、Q は直交テンソルである（極分解）[章末文献１,p21]。直交テンソル Q は剛体回転を表し、正値対称テンソル U,V は変形を表す。この極分解は一意的である。正値対称テンソルは、３の固有値と固有ベクトルを有し、歪の主軸と各軸の伸縮率を示す。

更に、テンソルは直交テンソル P と R を用いて A=PDR の形に分解でき、D は、U、V の標準形式（対角行列，${}^{t}D=D$）である。よって、$U={}^{t}R\,DR$、$V=PD\,{}^{t}P$ の形に表現でき、

$A=Q\,{}^{t}R\,DR=PD\,{}^{t}P\,Q=QU=VQ$ の形に表現できる。

但し、$Q\,{}^{t}R=P$、${}^{t}P\,Q=R$　つまり Q=PR

${}^{t}A={}^{t}R\ {}^{t}D\ {}^{t}P={}^{t}R\,D\ {}^{t}P$　であるから、

${}^{t}A\,A={}^{t}R\,D\ {}^{t}P\,PDR={}^{t}R\,DDR={}^{t}R\,D^{2}R$（対称行列）

よって、D は、${}^{t}A\,A$ の固有値の平方根を計算することにより求められる。

行列の積を繰り返すことにより回転を表現する操作を多数回繰り返すと、計算誤差により物体の形状が変化する。これは、スケール（歪）の要素混入することを意味する。この好ましくない歪の発生により回転行列に混入した誤差を検査するためには、スケール要素Ｄを抽出計算し、Ｄの単位行列からの差分として評価を行う。

以上を要約すると、景観シミュレーションにおける配置の記述に用いられるリンク・行列の内、回転を記述する３×３の部分は、二つの回転（自由度３のユニタリ行列が二つ）と一つのスケール（自由度３の対角行列）の積であり、合計した自由度９は、行列の構成

要素の総数９と一致している。力学で用いられている歪テンソルは、この内、回転の一つを除いたものと同等であり、６の自由度を有する。

リンク・行列で移動・回転・スケールを一体的に表現している４×４の同次行列全体は、四次元空間における二つの回転（自由度６のユニタリ行列が二つ）とスケール（自由度４の対角行列）であり、仮に第 4 の次元を時間とした時に、時空を表すベクトルの時間成分を１に固定し、速度を表す第 4 列だけを用いている。これは、四次元空間における回転を表す直交行列で、第 4 の次元と空間座標の間の回転を示す成分（仮に第 4 の次元が時間であれば、移動速度を意味する）を、単位時間が経過した後の位置の変化を形式的に座標の平行移動を記述するために用いたものである（（６）参照)。

よって、リンク・行列が保有する情報を幅１６の浮動小数の配列として出力するのではなく、要素に分解して取り出し、保存するためには、１の並進（TRANSLATE,自由度３)、１のスケール（SCALE,自由度３)、２の軸回転（ROTATE_A、自由度３×２）の合計１２のパラメータを取り出せば十分であり、残りの４(=16-12)の自由度は、第 4 の次元を空間座標に変換するパラメータであって、本処理系では使用されない冗長部分である。

② 数値解法

1) 二次元の回転における円の写像を用いた解法

二次元の場合、円は最初の回転 R で円に写像され、D で楕円に変形し、P で最終的な姿勢に回転する。2×2 の行列 M を、

$$M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

と置き、円上の点を$(\cos\theta, \sin\theta)$と表すと、この点は M により、

$(a\cos\theta + b\sin\theta,\ c\cos\theta + d\sin\theta)$　に写像される。

θ を変化させたとき、楕円の長軸と短軸に相当する位置では原点からの距離が停留するので、その角度 θ_i を求めれば、長軸と短軸の長さがわかる。θ が$-\pi \sim \pi$の範囲を検討すると、この範囲に４の停留点があり、長軸と短軸に２回ずつ遭遇する。よって、$-0.5\pi \sim 0.5\pi$の範囲を検討すればよい。

更に、長軸と短軸は直角であることから、$-0.25\pi \sim 0.25\pi$の範囲にある停留点がわかればよい。

原点からの距離が停留的であれば、距離の自乗も停留するから、

$$r^2 = (a\cos\theta + b\sin\theta)^2 + (c\cos\theta + d\sin\theta)^2$$

の増減を調べることにより、長軸と短軸を求めることができる。

r^2 は、$(a^2+b^2+c^2+d^2)/2 + ((a^2-b^2+c^2-d^2)/2)\cos 2\theta + (ab+cd)\sin 2\theta$

と変形できる。

θ で r を微分した勾配がゼロとなる位置では、2θ で r^2 を微分した勾配もゼロであることから、

$$-(a^2-b^2+c^2-d^2)\sin 2\theta + 2(ab+cd)\cos 2\theta = 0$$

$\tan(2\theta) = \sin 2\theta / \cos 2\theta$ を K と置くと、停留点において

$K = 2(ab+cd)/(a^2-b^2+c^2-d^2)$

である（解）。

更にこの K を用いて２ θ 、 θ を逆算すると、

C2 = cos 2θ = -1.0f / (float)sqrt(1.0 + K²); //但し$(a^2-b^2+c^2-d^2 < 0)$の場合

= 1.0f / (float)sqrt(1.0 + K²); //但し$(0 < a^2-b^2+c^2-d^2)$の場合

このC2を用いて、

C = cos θ = (float)sqrt(0.5 + 0.5*C2);//写像前の円周上の点

S = sin θ = (float)sqrt(0.5 - 0.5*C2); //但し、(K < 0)の場合

= (float)-sqrt(0.5 - 0.5*C2); //但し、(0 <= K)の場合

元の円周上の点(cos θ , sin θ)が回転行列 m[2][2]により移動した後の位置と原点を結ぶ線分が停留点即ち楕円の長軸と短軸である。そこで、第一の伸縮係数を、

```
Dx = m[0][0] * C + m[0][1] * S;//写像後の楕円の軸
Dy = m[1][0] * C + m[1][1] * S;
D = (float)sqrt(Dx*Dx + Dy*Dy);//軸の長さ（の半分）
```

により求め、更に第二の伸縮係数を

```
//元の単位玉最初の軸と直角方向：
Dx = m[0][0] * (-S) + m[0][1] * C;//写像後の楕円の軸(C,S)->(-S,C)
Dy = m[1][0] * (-S) + m[1][1] * C;
D = (float)sqrt(Dx*Dx + Dy*Dy);//軸の長さ（の半分）
```

により求める。

最初の回転 R は、長軸短軸となる θ を、スケールを掛ける座標軸に一致させる回転であり、 $\begin{bmatrix} C & S \\ -S & C \end{bmatrix}$ により、また最後の回転 P は、座標軸を長軸短軸とする楕円を最終的な姿勢に帰着させる回転であり

$$\begin{bmatrix} Dx/D & -Dy/D \\ Dy/D & Dx/D \end{bmatrix}$$

により求める。

［例１］正方形{(0,0),(1,0),(1,1),(0,1)}を{(0,0)-(1,0)-(2,1)-(1,1)}に変形させる。

二次元の行列ないしテンソル A は以下の通りである。

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

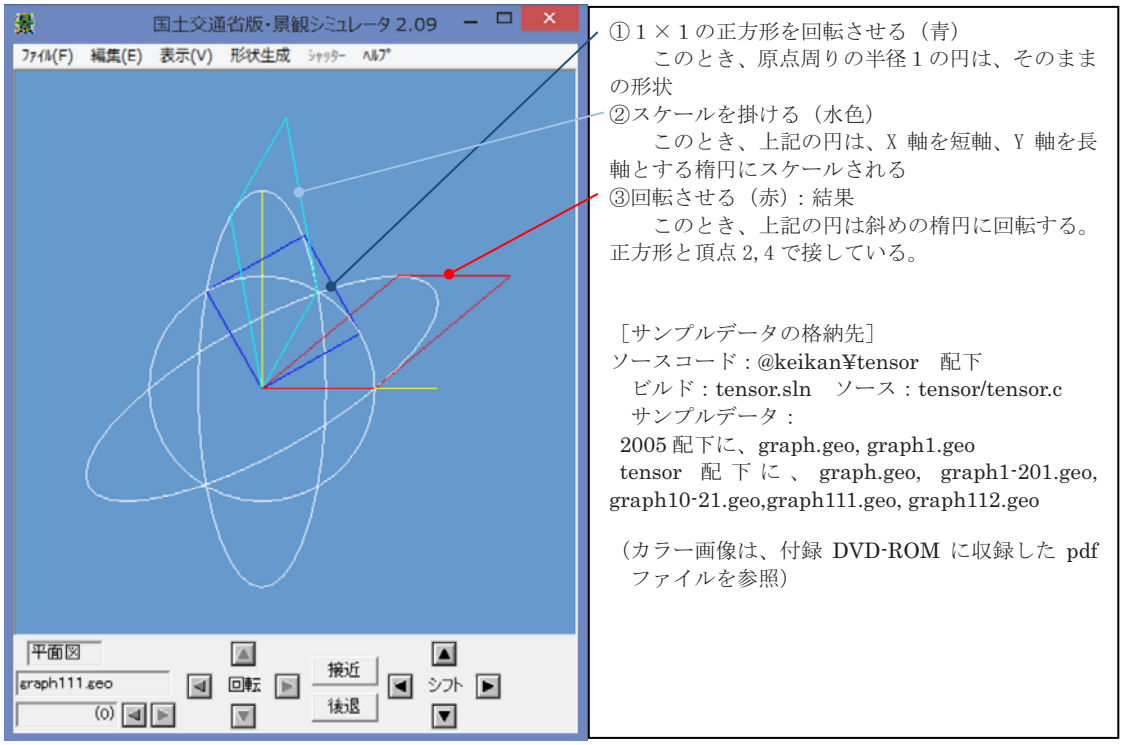


図 4-4-4 座標軸に沿った正方形の平行四辺形へのすべり変形[例１]

この A を PDR の形に分解する方法として、二次元の場合には、上記の②のように、角度をパラメータとして、三角関数を用いて解を求めたものを、図 4-4-4 に示す。

第一の回転：31. 7° 第二の回転：-58. 3°

$$P=\begin{bmatrix} 0.52 & 0.85 \\ -0.85 & 0.52 \end{bmatrix} \quad D=\begin{bmatrix} 1.62 & 0.00 \\ 0.00 & 0.62 \end{bmatrix} \quad R=\begin{bmatrix} 0.85 & -0.52 \\ 0.52 & 0.95 \end{bmatrix}$$

［例２］原点付近の XY 平面上の正方形{(0,0)-(1,0)-(1,1)-(0,1)}を、同じ Y 軸高さを持つ平行四辺形{(0,0)-(1,0)-(3,1)-(2,1)}に変形させる。

二次元の行列ないしテンソル A は以下の通りである。

$$A=\begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}$$

この A を PDR の形に分解する。P と R は、回転を表す直交行列、D はスケールないしストレッチを表す対角行列である。この結果を図 4-4-5 に示した。

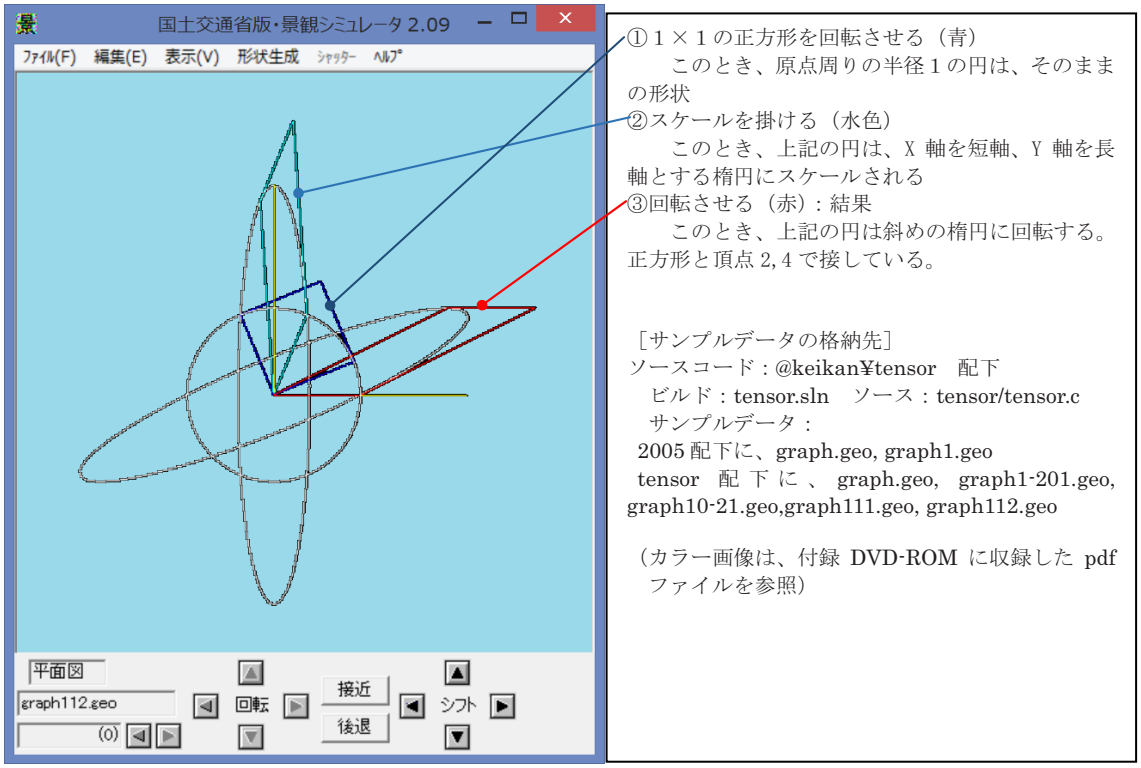


図 4-4-5 例１よりも更に大きなすべり変形［例２］

第一の回転：22.5° 第二の回転：-67.5°

$$P=\begin{bmatrix} 0.38 & 0.92 \\ -0.92 & 0.38 \end{bmatrix} \quad D=\begin{bmatrix} 2.41 & 0.00 \\ 0.00 & 0.41 \end{bmatrix} \quad R=\begin{bmatrix} 0.92 & -0.38 \\ 0.38 & 0.92 \end{bmatrix}$$

2) 行列の固有方程式を用いた解法

${}^{t}M M$ は正値対称行列であることから固有値を求めることができ、固有ベクトルが求められる。固有値は、３軸の伸縮を示し、行列を標準形式で表した時の対角成分である。

各固有値 λ_i に対応する固有ベクトルを V_i とすると、$AV_i=\lambda_i V_i$、よって

$(A-\lambda_i I)V_i=0$ となる（I は単位行列）。

この時、行列 $A-\lambda_i I$ は、全てのベクトルを、V_i に直交する平面上に写像する。つまり、$A-\lambda_i I$ の各行は、V_i と直交する一つの平面上にある。このことから、V_i は、$A-\lambda_i I$ の任意の二つの行をベクトルとしたときのその外積に平行である。

二つの固有値が等しい場合には、この二つと異なるユニークな固有値に対応する固有ベクトルを V_1 とした時に、残りの二つのベクトル V_2、V_3 は V_1 と直交し、かつ互いに直交する二つのベクトルである。このことから、例えば V_1 の三成分の内絶対値が最小となる成分に対応する座標軸方向の単位ベクトルと V_1 の外積として、V_2 を求め、更に V_1 と V_2 の外積として V_3 を求めればよい。

三つの固有値が全て等しい場合、固有ベクトルは、３の自由度を有する。この場合には、任意の互いに直交する３のベクトルを固有ベクトルとして使用し、回転行列を作成するこ

とができる。

３の固有ベクトルから作成した正規直交行列（変形を表す３の主軸を座標軸とする座標系への回転を表す）を用いて、行列を対角化することできる。

まず二次元的な問題を固有方程式により解く例を示す。

［例３］固原点付近の正方形{(0,0)-(1,0)-(1,1)-(0,1)}を、菱形{(0,0)-(1,0)-(1.6,0.8)-(0.6,0.8)}に変形させる。二次元の行列ないしテンソル A は以下の通りである。

$$A=\begin{bmatrix} 1 & 0.6 \\ 0 & 0.8 \end{bmatrix}$$

この例では A を固有値の計算を用いて PDR の形に分解する。P と R は、回転を表す直交行列、D はスケールないしストレッチを表す対角行列である。

det(A)は、であるから、このテンソルは A の体積ないし底面積を 0.8 に縮小する。

A ^ A＝PDD^P の対称行列の固有値は、D の対角成分の二乗である。

$$|A^A-\lambda I| = \begin{vmatrix} 1.36-\lambda & 0.48 \\ 0.48 & 0.64-\lambda \end{vmatrix} =0$$

を解いて、λ＝｛1.6,0.4｝を得る。

これから、

$$D=\begin{bmatrix} \sqrt{1.6} & 0 \\ 0 & \sqrt{0.4} \end{bmatrix}$$

更に、

$$P=\begin{bmatrix} \sqrt{0.8} & -\sqrt{0.2} \\ \sqrt{0.2} & \sqrt{0.8} \end{bmatrix}$$

$$R=\begin{bmatrix} \sqrt{0.5} & \sqrt{0.5} \\ \sqrt{0.5} & \sqrt{0.5} \end{bmatrix}$$

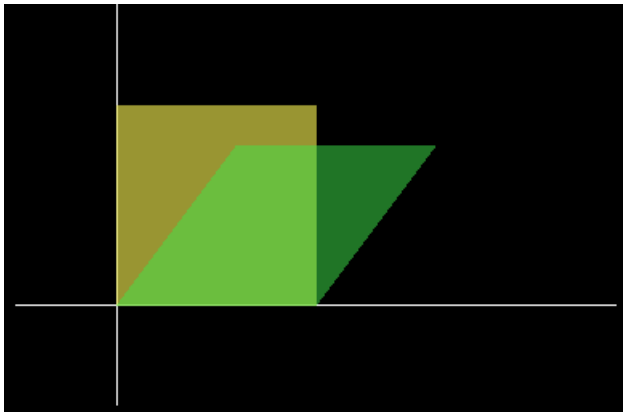


図 4-4-6　A による回転＋変形＋回転

この過程を PDR に分解して図示すると、正方形をまず R により４５度回転する。

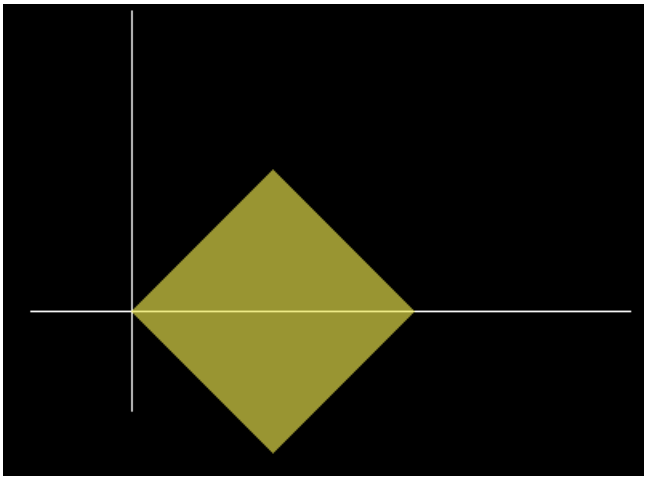


図 4-4-7 R による回転

次に、D により X 軸方向に√1.6 倍、Y 軸方向に√0.4 倍スケールを掛ける。

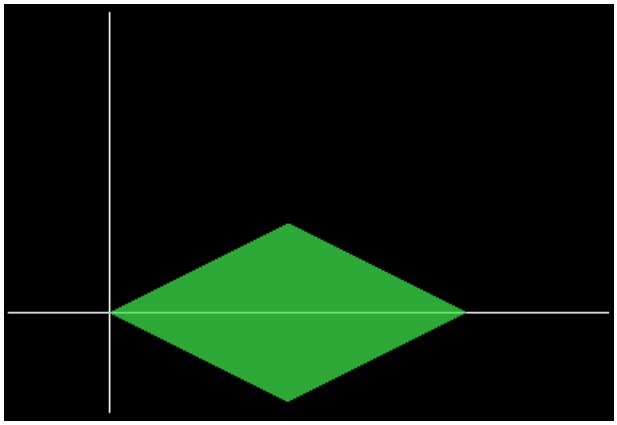


図 4-4-8　D によるスケールまたはストレッチ

最後に、P を適用して、最終的な位置に回転する。

図 4-4-9　P による回転

このケースでは、二次元の回転を扱っているが、固有値を計算し、固有ベクトルからスケール／ストレッチの前後の回転を求める手順は、三次元以上の場合でも同様である。

三次元の場合、固有方程式は、３次方程式となる。４次方程式までは代数的に解くことができ、３次方程式の解法は、カルダーノ(Gerolamo Cardano, 1501-1575)の Ars Magna により初めて解説された。

Cardano の解法の概要：
任意の三次方程式は、変数変換により、
$x^3 + px + q = 0$
の形に変形できる。ここで、$x = u - p/3u$ と置くと、
$u^3 + q + p^3/27u^3 = 0$ つまり、
$u^6 + qu^3 + (p/3)^3 = 0$ となる。これは u^3 に関する二次方程式として解くことができ、
$u^3 = -q/2 \pm \sqrt{((q/2)^2 + (p/3)^3)}$
三乗根として求めた二つの u を用いて、x を計算する。
このとき、３の最終的な実数解がある場合であっても、
$(q/2)^2+(p/3)^3 < 0$ となるため、途中経過的に複素数として三乗根 u を計算し、
最終的に実数解として x を得る。

３の固有値が存在する場合、それぞれに対応する３の固有ベクトルが直交し、正規直交行列は、回転に相当する座標変換を表現する。よって、これら３の固有ベクトルを正規化したものを行とする３×３の行列を作成すれば、これが回転を表す行列となる。

［例４］三次元の回転行列を、二つの回転と、一つのスケールに分解する。

$$\begin{bmatrix} 0.00 & 1.20 & 0.00 \\ -0.87 & 0.00 & -0.50 \\ -0.40 & 0.00 & 0.69 \end{bmatrix} = M = PDR$$

転置行列をかけて、 正値対称行列（歪テンソルに相当）を作成し、固有方程式を立てる。

$$\begin{bmatrix} 0.92 & 0.00 & 0.16 \\ 0.00 & 1.44 & 0.00 \\ 0.16 & 0.00 & 0.73 \end{bmatrix} = {}^tMM = RDDR$$

D^2の固有方程式は、

$$u^3 - 3.083u^2 + 3.0064\ u + -0.92229 = 0;$$

この解は３あって、大きい順に 1.44, 1.01, 0.64　ゆえにＤの固有値は、{1.2, 1.0, 0.8}である。これを用いて、行列は次のように分解される。

この場合は、Mの行列式（すなわち体積倍率）DET=0.96 で正値であることから、D^2からＤを求める計算において、負値(-sqrt)を用いる必要はないと判定できる。

解析結果　無回転← スケール(1.2, 1, 0.8)←軸(0.25,0.25,-0.935)周りの 93.8° 回転

$$\begin{bmatrix} 0.00 & 1.20 & 0.00 \\ -0.87 & 0.00 & -0.50 \\ -0.40 & 0.00 & 0.69 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1.2 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.8 \end{bmatrix} \begin{bmatrix} 0.00 & 1.0 & 0.00 \\ -0.87 & 0.0 & -0.49 \\ -0.50 & 0.0 & 0.87 \end{bmatrix}$$

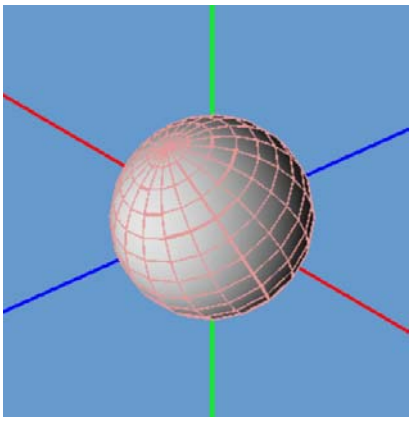


図4-4-10 第一の回転

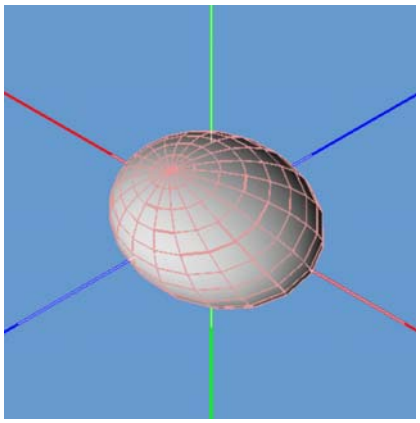


図4-4-11 スケール（最終状態と同じ）

なお、この行列データを作成した際の、Y軸30° 回転→スケール(1.0, 1.2, 0.8)→Z軸90° 回転の過程を球の変形として図4-4-12～15に図示する。

データ作成過程　　Z軸90° 回転←　スケール(1, 1.2, 0.8)　←Y軸30° 回転

$$\begin{bmatrix} 0.00 & 1.20 & 0.00 \\ -0.87 & 0.00 & -0.50 \\ -0.40 & 0.00 & 0.69 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.2 & 0.0 \\ 0.0 & 0.0 & 0.8 \end{bmatrix} \begin{bmatrix} 0.87 & 0.0 & 0.5 \\ 0.0 & 1.0 & 0.0 \\ -0.5 & 0.0 & 0.87 \end{bmatrix}$$

これは、解析の結果として分解された回転→スケール→回転の行列とは表現が異なっている。この例で、スケールにおける倍率を1.2倍とする軸を作成過程ではY軸に一致させているが、解析結果ではX軸に一致させている。1.2倍とする軸をどの軸に一致させるように回転するかは次項で解説するように複数の解が存在し、いずれも同じ回転行列を与える。本処理系では、解析の過程でスケールを表す対角行列の３の対角成分（固有値）を左上から右下に向けて大きい順に並べているためである。

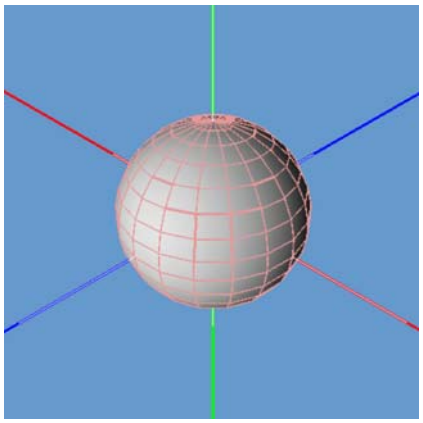


図 4-4-12 最初の状態

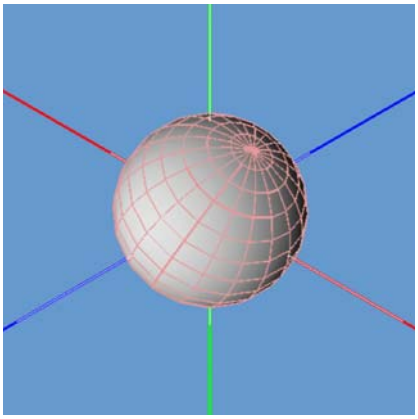


図 4-4-13 第一の回転後（Y：30°）

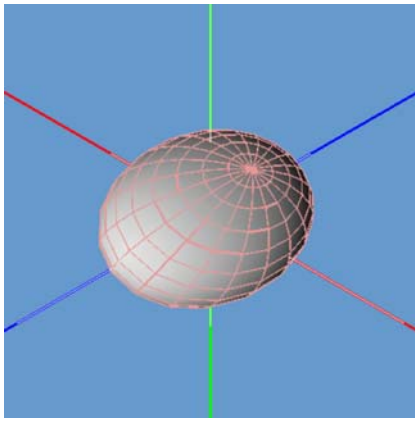


図 4-4-14 スケール(X1.0 Y1.2,Z0.8)

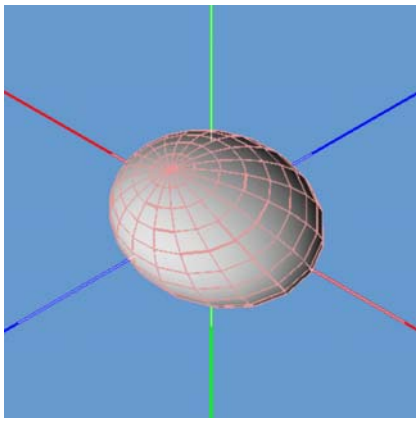


図 4-4-15 第二の回転(Z：-90°)

3）プログラム実装のための、座標軸に関する自由度削減の方法

幾何学的に「任意」であるという条件にプログラムで対応するためには、実装上の工夫が必要である。

XYZ 軸のいずれかに関して、二次元的な回転しかない場合は、行列のある対角成分が１.０であることにより　識別できる。これにより、１）の方法での解析が可能である。

例えばリンク・行列が以下のような形をしていた場合、Z 軸（高さ）に関してはスケール 1.3 だけが抽出され、X 軸と Y 軸の間で Z 軸を回転軸とする回転と、XY 空間におけるスケールの主軸が２以下抽出されることとなる。従って、残りの X 軸と Y 軸に関して、二次元の問題として解析すれば良い。あらかじめ問題の次元を三次元から二次元に削減することができれば、計算回数と計算時間は大幅に縮減できる。

$$\begin{bmatrix} M[0] & M[4] & M[8]=0.0 & M[12] \\ M[1] & M[5] & M[9]=0.0 & M[13] \\ M[2]=0.0 & M[6]=0.0 & M[10]=1.3 & M[14] \\ M[3] & M[7] & M[11] & M[15] \end{bmatrix}$$

このような次元に関する自由度削減ができず、３軸のスケール（ストレッチ）に分解す

る場合であっても、各固有値をどの座標軸に対応させるかは任意となる。例えば、３の異なるスケール 1.1，1.0，0.9 が固有値である場合、これに対応する回転行列は、６通り存在しうる。最大の 1.1 を X 軸、Y 軸、あるいは Z 軸とするいずれの分解も可能である。プログラム実装上は、大きな固有値から順にソートすることにより、スケールに関する行列を一意的に定めることができる。例えば、最大の固有値に対応するスケールの主軸を X 軸、二番目を Y 軸、最小の固有値を Z 軸とするようにプログラムすることができる。

更に、選択した各軸をどの向きにするかで各軸２通り、３軸では８通りの選択が可能である。本処理系においては、３軸のそれぞれが回転した後の軸と成す角度が、90 度以下となるように、各軸の向きを選択し回転行列を決定している。具体的には、回転行列の対角成分が正となるように列毎の符号反転を行う。これにより、座標軸の対応に関して生じる、鏡像となる回転も含めた６×８＝４８の解からの選択を絞り込んでいる。

３の固有値の一部が一致している場合（固有方程式重根の場合）には、回転行列の軸の設定が任意となる。例えば、上記の二次元的な行列の問題（Z 軸のスケールが 1.3）で、XY 平面でのスケールが共に 0.9 である場合（等方的である場合）には、スケールの主軸は任意となり、Z 軸に関する回転は任意となる。プログラム実装において、このような場合には、回転はないものとして処理することができる。

上記のような二次元的な問題に還元できることは、あらかじめ情報として与えられている場合を除き、リンク・行列の数値的外観を判別するだけではわからず、回転とスケールの分解を行った結果初めて、例えば、３のスケールの主軸の内ある主軸に関するスケールが 1.3、残りの２軸に関するスケールが等しく 0.9 であることにより判別される。

二次元の問題に還元できる場合、二つの回転行列は、スケールの主軸の回転だけを一意的に決定し、それ以外は任意であるように選択できる（自由度１）。スケールの主軸の一つだけを二つの向きの間で移動させる無数の回転の内、回転角度を最小とする回転が存在し、その軸は、回転前の主軸と回転後の主軸の外積としてプログラムにおいて一つに決めることができる（乱数を用いて決めるよりは良い方法である）。この回転は、主軸の回転が、単位球面上の大円上の回転であるような、回転角が最小となる回転経路に対応している。

スケールが３軸全てに関して等方的である場合（一様な体膨張・収縮の場合）には、回転は全く任意となる。このような場合は、プログラム上は回転なし（単位行列）として処理すればよい。

第二の回転行列は、リンク・行列の部分として取り出した回転・スケールを表す行列に、既に求めた第一の回転行列の逆行列即ち転置行列と、スケールを表す対角行列の逆行列即ち対角成分を逆数とした行列を前から掛けることにより一意的に求められる。

上記の処理は、本処理系においては、ライブラリ関数の内、４－２(6)保存データの利活用系のためのライブラリ関数に含まれており、リンク・行列から回転成分を四元数として取得する Lr 関数と，スケール成分を取得する Ls 関数として実装されている。

この計算処理の主目的は、シーングラフとして記述された集落等におけるモデルの階層

間の位置関係を記述しているリンク・行列を、任意形式のファイルへの出力などに際して、より単純な要素に分解することにある。

しかし前述のように、同次行列で表現したスケールの変更のない純粋な回転を多数回繰り返した後に、計算誤差により発生し、行列の中に混入しているスケール要素（単位行列からの偏差）を評価するためにも使用することができる。

逆に、本処理系の中でも、無数の回転を繰り返す VC-3M の姿勢情報とキャリブレーション値の管理のためには、回転に伴い、本来存在しないスケール要素が誤差として発生することを未然に防ぐために、姿勢とその補正情報を管理するために同次行列ではなく、（４）で解説する四元数を用いている。

③ プログラムの実装とテスト

計算原理については、コンソール・アプリ実行形式 tensor.exe を生成するビルド tensor.sln を用いて、プログラムの検証のために、以下のようなテストを行った。

・三次元の任意の行列 M を乱数から生成する
・これを、PDR に分解する
・PDR の積を計算したものと、M との格差を自乗平均値として評価する
・格差が大きい場合にログを出力し、デバッグを行う

これらの処理は、内部で記録管理しているリンク・行列を対象として、保存出力段階における分解出力、回転のみを表現する場合の誤差の評価、および単純な回転に還元できる場合の省略処理に使用する。

付録 CD-ROM に関する解説
tensor ディレクトリの配下には、VS2008 開発環境におけるビルド(tensor.sln)と、2005/2005.sln という VS2005 開発環境におけるビルドがある。
VS2008 のビルドにおける実行形式は、tensor/Debug/tensor.exe
VS2005 のビルドにおける実行形式は、tensor/2005/Debug/2005.exe
三次元版のソースコードは、tensor/tensor/tensor.c で、二次元版の試作は、tensor140709.c にアーカイブされている。
二次元の解法（三角関数の微分）については、main2d 関数（当初は main 関数）にアーカイブされている。
pmat3 関数以降が、Cardano による三次方程式の解法を用いた三次元の処理である。

これらの成果を用いて、ライブラリ関数 Lr(), Ls()を実装した。あるリンクが選択されている状態で、Ls()はスケールを抽出し、Lr()は回転要素を抽出して四元数形式で返す。Lr の引数として PRE が指定された場合には、スケールの前の回転を、POST が指定された場合には、スケールの後の回転を返し、引数がない場合にはそれらの合成を返す。

④ ライブラリ関数によるアクセス

解析系のライブラリ関数として、いずれも四元数型の Lt, Lr, Ls 関数を用意した。

Lt()は、同次行列の並進部分(平行移動)を返す。

Ls()は、スケール要素を返す。

Lr()は、回転要素を返す。引数として、PRE または POST が指定された場合には、同次行列を回転・スケール・回転に分解した上で、PRE ならば第一の回転を、POST ならば第二の回転を返す。

これらのライブラリ関数の最も簡単な使用方法として、次のようにプログラムする。

リスト 4-4-4 ライブラリ関数によるリンク行列へのアクセス

```
int main(){
  int i;
  quat q;
  i=0;
  while(i= L() ){ //全てのリンクに順次アクセスする
    logf("LINK[%d]¥n",i);
    q = Lt(); //平行移動
    logf("Lt()=%q¥n",q);
    q = Lr();//単純回転
    logf("Lr()=%q¥n",q);
    q = Lr(PRE);//スケールと二つの回転に分解した後の第一の回転
    logf("Lr(PRE)=%q¥n",q);
    q = Lr(POST);//スケールと二つの回転に分解した後の第二の回転
    logf("Lr(POST)=%q¥n",q);
    q = Ls();//スケールと二つの回転に分解した後のスケール
    logf("Ls()=%q¥n",q);
    for(j=0;j<16;j++){
      logf("m[%d]=",j);
      logf("%f¥n",Lm());//同次行列の 16 の配列要素を順次取得する
    }
  }
}
```

リスト 4-4-5 リンク行列の出力例

```
LINK[2]
Lr()=_Q(0.650542,0.192147,0.153718,-0.793963)
Lr(PRE)=_Q(0.683013,-0.183013,-0.183013,0.683013)
Lr(POST)=_Q(0.707107,0.000000,-0.000000,0.000000)
Lt()=_Q(0.000000,0.000000,0.000000,0.000000)
Ls()=_Q(0.000000,1.200000,1.000000,0.800000)
m[0]=0.000000
m[1]=-0.866025
m[2]=-0.400000
m[3]=0.000000
m[4]=1.200000
m[5]=0.000000
m[6]=0.000000
m[7]=0.000000
m[8]=0.000000
m[9]=-0.500000
m[10]=0.692820
m[11]=0.000000
m[12]=0.000000
m[13]=0.000000
m[14]=0.000000
m[15]=1.000000
```

（３）行列の自由度の幾何学的な意味

ここで、回転とスケールを表す行列の幾何学的な意味と、その冗長性について整理する。

1）任意の行列（２×２、３×３、４×４）は、PDR の形に分解でき、

回転→スケール→回転を合成したものである。P と R は回転を表す直交行列であり、D はスケールを表す対角行列である。この時、P,D,R の各自由度は、

1　　2　　1　（二次元の場合）　合計 1+2+1＝4=2^2

3　　3　　3　（三次元の場合）　合計 3+3+3=9=3^2

6　　4　　6　（四次元の場合）　合計 6+4+6=16=4^2

二次元の回転は、角度だけで記述できる一つの自由度しかもたない。

三次元の回転は、３の自由度をもつ。

四次元の回転は、６の自由度をもつ。空間的な回転を表す３の次元と、空間３次元が第４次元と成す傾斜角（仮に第４の次元を時間とすると並進速度）を表す３の次元から成る。

このように、リンク・行列を回転→スケール→回転の３操作を合成したものと見るならば、自由度が行列の要素数と過不足なく一致していることがわかる。

2) 行列式の値は、膨張率に等しい

3) 正規直交行列の場合、回転行列であり、転置行列が逆行列と一致し、固有値は１である

このとき固有ベクトルは回転の軸となる。

4) 対称行列の場合、固有値は各軸の伸縮率に等しく、固有ベクトルは伸縮軸である

5) 対角行列の場合、対角成分は、各軸の伸縮率である

④⑤において等しい固有値λ_1, λ_2(重根)に対応する固有ベクトルは、面内の任意の直交軸である。また３重根の場合には等方的伸縮を表し、固有ベクトルは任意の直交３軸である。

$M-\lambda I$が同一平面上にあることから、この各行をベクトルと見て、それらの外積から固有ベクトルを求めることができる。二つの固有値が等しい場合には、ユニークな固有値に属するベクトルをまず求め、これに直交する任意の二つの単位ベクトルを求める。

３の固有値が等しい場合には、等方的な伸縮を表す。この場合、任意の直交する単位ベクトルをもって固有ベクトルとすることができる。

３の固有ベクトルを行として並べることにより、回転行列を求めることができる。

３の固有ベクトルは、回転後の座標系の座標軸に対応する基底ベクトルである。あるベクトルにこの行列を掛けると、回転後の座標系の各軸との内積を成分とするベクトルが得られる。これは、座標変換を意味する。

ある固有値がゼロの場合、形状の次元が下がる。例えば、一つの固有値がゼロである場合、言い換えるとスケールを表す対角行列の一つの対角成分がゼロである場合、立体が潰れて面になる。また、二つの固有値がゼロである場合、線となり、三つともゼロである場合、あらゆる座標点が１点に変換される。

（４）四元数を用いた移動と回転の表現

一つのスカラー値と、三次元ベクトルに対応する三つの座標値を組み合わせた四元数は、ベクトル演算と空間的な回転を、余分な冗長性なく表現することができる。このため、同次行列のように計算によるスケール要素が誤差として混入し蓄積するおそれがなく、誤差は軸の向きと回転角に関するもののみである。また、データに冗長性がないことから、計算回数を節約することができる。このため、一つのセッションの間に非常に多くの回転に関する座標計算を行うＶＣ－３Ｍにおける視点情報の計算に使用した。

また、三次元ベクトルに関する処理を、一つの数値（シンボル）を用いて簡潔に表現できることから、メタファイルの記述に使用できるように、変数、配列、関数の数値型の一つとして標準で組み込んでおり、これを引数や戻り値として使用できるライブラリ関数を

用意した。

① 表現方法

1) 実数虚数表現

また、虚数単位 **i**， **j**， **k** を用いて、

$Qt + Qx\ \mathbf{i} + Qy\ \mathbf{j} + Qz\ \mathbf{k}$

数式表現できる。ハミルトンにより四元数が発見された頃の表現方法であった。

$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1,\ \mathbf{i}\ \mathbf{j} = \mathbf{k},\ \mathbf{j}\ \mathbf{i} = \mathbf{i},\ \mathbf{k}\ \mathbf{i} = \mathbf{j},\ \mathbf{j}\ \mathbf{i} = -\mathbf{k},\ \mathbf{k}\ \mathbf{j} = -\mathbf{i},\ \mathbf{i}\ \mathbf{k} = -\mathbf{j}$

2) 配列表現

四つの浮動小数による配列として

```
float f[4] = {Qt,Qx,Qy,Qz};
```

と表現できる。

3) スカラーベクトル表現

また、四元数 **Q** をスカラー部分Ｑｓとベクトル部分 **Ｑｖ** に分けて表現することもできる。

$$\mathbf{Qv} = \begin{bmatrix} Qx \\ Qy \\ Qz \end{bmatrix} \qquad \mathbf{Q} = \begin{bmatrix} Qs \\ \mathbf{Qv} \end{bmatrix}$$

ベクトルの内積・外積等に慣れているプログラマーにはわかりやすい表現である。

4) 行列表現

他に、群論の研究等において４×４の実数行列や、虚数を含む２×２の行列により種類の基底を表現する表記法が行われるが割愛する。

② 演算

1) 加減算

加減算は、４の要素のそれぞれに関して独立して加減を行う。よって、演算は可換である。

$\mathbf{Q_1} + \mathbf{Q_2} = \mathbf{Q_2} + \mathbf{Q_1}$

$\mathbf{Q_1} - \mathbf{Q_2} = -(\mathbf{Q_2} - \mathbf{Q_1})$

2) 乗算

配列表現で、$\mathbf{Q_1}$ を{W1,X1,Y1,Z1}, $\mathbf{Q_2}$ を{W2,X2,Y2,Z2}とすると、

四元数積 $\mathbf{Q_1Q_2}$ は、

$\{W_1W_2 - X_1X_2 - Y_1Y_2 - Z_1Z_2,$

$W_1X_2 + X_1W_2 + Y_1Z_2 - Z_1Y_2,$

$W_1Y_2 - X_1Z_2 + Y_1W_2 + Z_1X_2,$

$W_1Z_2 + X_1Y_2 - Y_1X_1 + Z_1Y_2\}$と表せる。また、スカラーとベクトルで表現すると、

$$\mathbf{Q_1Q_2} = \begin{bmatrix} W1 \\ \end{bmatrix} \begin{bmatrix} W2 \\ \end{bmatrix} = \begin{bmatrix} W1\,W2 - \mathbf{V1 \cdot V2} \\ \end{bmatrix}$$

$$\mathbf{V1} \quad \mathbf{V2} \quad = \quad W1\,\mathbf{V2} + W2\,\mathbf{V1} + \mathbf{V1} \times \mathbf{V2}$$

特に、スカラー成分 W_1 と W_2 がゼロの場合には、

{- X_1Y_1 - Y_1Y_2 - Z_1Z_2,

Y_1Z_2 - Y_2Z_1, -X_1Z_2+Z_1X_2, X_1Y_2-Y_2X_1}

ベクトル部のみからなる $\mathbf{Q_1}$ と $\mathbf{Q_2}$ をベクトル V_1 と V_2 として見た時、この積のスカラー部は-V_1・V_2（内積）であり、ベクトル部は、$V_1 \times V_2$（外積）となっている。

$$\mathbf{Q_1Q_2} = \begin{bmatrix} 0 \\ \mathbf{V1} \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{V2} \end{bmatrix} = \begin{bmatrix} -\mathbf{V1} \cdot \mathbf{V2} \\ \mathbf{V1} \times \mathbf{V2} \end{bmatrix}$$

このベクトルの外積が可換ではないことに対応して、四元数は乗算に関して可換ではない。つまり $\mathbf{Q_1Q_2}=\mathbf{Q_1Q_2}$ は成立しない。

3) ゼロ元

任意の $\mathbf{Q_1}$ に対して、$\mathbf{Q_1Q_2}=\mathbf{Q_1}$ となるような $\mathbf{Q_2}$ は、全ての要素がゼロである。

4) 単位元

任意の $\mathbf{Q_1}$ に対して、$\mathbf{Q_1Q_2}==\mathbf{Q_1}$ となる $\mathbf{Q_2}$ は、{1,0,0,0}である。

この時、$\mathbf{Q_2Q_1}==\mathbf{Q_1}$ もまた成立する。

5) 絶対値

$\mathbf{Q}$={W,X,Y,Z}の絶対値を、$\sqrt{}(W^2+X^2+T^2+Z^2)$として定義する。

6) 逆元

任意の $\mathbf{Q_1}$={W_1,X_1,Y_1,Z_1}に対して $\mathbf{Q_1Q_2}$={1,0,0,0}となるような $\mathbf{Q_2}$={W_2,X_2,Y_2,Z_2}は、$\mathbf{Q_1}$ の絶対値を R とした時、$W_2=W_1/R^2$、$X_2=-X_1/R^2$、$Y_2=Y_1/R^2$、$Z_2=Z_1/R^2$ として求められる。

7)超越関数

複素数と同様に、指数関数、対数関数を定義することができる。スカラー部は指数関数的な倍率として作用し、ベクトル部はその方向を維持したままでスカラー部とベクトル部の間で、長さ２πを周期として振動的に作用する。

③ 回転の表現

三次元空間における回転に関する座標変換は、絶対値が１に等しい一つの四元数 $\mathbf{Q_r}$ によって表現することができ、任意のベクトル $\mathbf{Q}$={0,V}の座標変換は、$\mathbf{Q_r\,Q\,/Q_r}$ によって表現できる。

この時、$\mathbf{Qr}$ のベクトル部 Vr は、回転軸と一致する。また、Qr を回転角を θ とした場合、$\mathbf{Qr}$ のスカラー部は、$\cos(\theta/2)$であり、ベクトル部は長さ１のベクトルに $\sin(\theta/2)$を乗じたものである。

$\mathbf{Qr}$={$\cos(\theta/2)$, Xr $\sin(\theta/2)$, Yr $\sin(\theta/2)$, Zr $\sin(\theta/2)$}

特別な場合として、回転角がゼロである場合、$\mathbf{Qr}$ は単位元{1,0,0,0}に等しい。

この表現方法は、本処理系における LINK_XFORM の、ROTATE_A の表現方法と同じものである。

二つの四元数の乗算 $\mathbf{QrQ/Qr}$ の過程は、幾何的には以下のようになっている。

1) 第一の乗算 **QrQ** によって、**Q** の成分の内、Qv と直交するベクトル成分は、Vr を軸として θ /2 の角度だけ回転する。この時、**Q** の内 Qv と平行の成分と、Q のスカラー成分との間でも θ /2 だけ回転が生じる。

2) 次にこの結果を **Qr** で除す（$\mathbf{Qr}^{-1}$ を後ろから乗じる）ことにより、**Q** の内、Qv と直交するベクトル成分は、Vr を軸として更に θ /2 の角度だけ回転し、合計して最初の位置から θ の角度だけ回転する。この時、**Q** の内 Qv と平行の成分と、**Q** のスカラー成分との間では逆に－ θ /2 だけ回転する。結果的に、スカラー成分と軸成分は、元の **Q** と同じ値に戻る。

従って、この計算は、第四の次元と可視的な三次元空間との間の相互関係をキャンセルし、空間的な回転だけを結果的に作用している。

3) **QrQQr** の乗算を行うと、②とは異なり、**Qr** を軸とした **Q** のベクトル成分の回転は元に戻り、最初と同じ向きとなる。

このとき、**Qr** の軸方向の **Q** の成分と **Qr** のスカラー成分との間の回転は更に進み θ に達する。Q のスカラー成分がゼロであった場合、Qr の軸方向に縮むような三次元的変形が生じる。このとき、対応する部分がスカラー成分に移転しており、$\sqrt{(Qw^2+Qx^2+Qy^2+Qz^2)}$という計量による四次元的な長さは保たれ、四次元空間における回転となっている。

Qr のスカラー成分 Qw と θ が共に純虚数である場合、Q の長さは$\sqrt{(-Qw^2+Qx^2+Qy^2+Qz^2)}$の形の計量において不変となり、ローレンツ変換による（三次元的）変形が生じる。

この時、cos(i θ)=cosh(θ)、sin(i θ)=**i** sinh(θ)である。

θ が純虚数であった場合、これを i θ（ θ は実数）と表すと、cos(i θ)=cosh(θ)、sin(i θ)=**i** sinh(θ)である。これに伴い **Qr** のベクトル成分 Qx、Qy、Qz が共に純虚数である場合、Q の大きさは$\sqrt{(Qw^2-Qx^2-Qy^2-Qz^2)}$の形の計量において不変となり、ローレンツ変換による（三次元的）変形が生じる。現代物理学によれば、現実に我々が生活しているのはこのような空間である。この場合、長さゼロの領域は大きさの無い点ではなく、光速で拡大する超球の面上の点（自由度３）となり、その外側の（超光速でないとアクセスできない）領域は距離が負値となる。これは θ が実数の四次元空間の計量における、「点よりも小さな領域」に対応する。

超球上の点までの距離は、どのような座標変換（四次元空間における回転）を行っても長さゼロで一定であることは光速が一定であることに対応する。これは三次元空間において点の大きさが座標変換を行ってもやはりゼロであることと直感的には類似している。

回転を表す四元数を **Qs**、並進を表す四元数を **Qv** と表し、これをある任意の四元数 **P**（空間座標と、第四の次元におけるある非ゼロの値を有する）に適用すると、P は、

QvQsP/QsQv に変換・写像される。この時、**Qs** と **Qv** のベクトル部が互いに独立であれば、それぞれは３の自由度を有し、この変換の合成は合計６の自由度を有する。このことは、２(5)で４×４の行列の回転に関する自由度が６であることと整合している。

形式的表現に用いられる sin(**i** θ)または **i** sinh(θ)として形式的に現れる純虚数は最終的に解消するため、変換後の四元数のスカラー部またはベクトル部に、実数と虚数が混在す

ることはない。

四元数の演算における積 **AB** をスカラー部とベクトル部での表現において、

$$\begin{matrix} a & b \\ A & B \end{matrix} = \begin{matrix} ab + A \cdot B \\ aB + bA + A \times B \end{matrix}$$

と定義したときの双曲四元数がこれに相当する。

この時、**A** の長さは、a^2-A^2 となる。回転を表現する **Qs** するベクトル部は三次元空間のスカラー部は１以上の値となり、ベクトル部は三次元空間全体に広がる。原点は無回転に対応し、その場合のスカラー部は１となる。この空間においては、四元数の全ての成分が実数であっても長さが負となる領域が存在し、結合則が成立しない。

④ 回転行列からの回転軸と回転角の計算

剛体の姿勢を回転行列で表現する場合、回転前の剛体に固定された座標系の各座標軸上の長さ１の点{1,0,0},{0,1,0},{0,0,1}のそれぞれの、回転後の座標値を回転前の座標軸を用いて求め、この３のベクトルを行として並べて行列を作成した回転行列が用いられる。

回転行列から、回転を表現する四元数 **Qr** を求めるためには、剛体上の回転に伴い移動する２点に関して移動をベクトルとして求め、これらの外積を計算したものを正規化すれば、軸の向きを単位ベクトル Vr として求められ。次に、この軸の回りの回転角を求め、cos からスカラー成分、sin からベクトル成分をそれぞれ算出する。

剛体に固定した座標系で表現できる場合、回転行列を Mr、軸を Vr としたとき、MrVr≡Vr であるから、(Mr-I) Vr≡０である。行列 Mr-I は、回転軸方向の高さを全て相殺し、全空間の座標点を回転軸と垂直で原点を通る平面上の点に平行移動する写像を表す。このことから、対角成分から１を引いた Mr-I の各行を３のベクトル（上記平面内にある）と見立て、任意の二つを選んで外積を求めれば、３のベクトルと直交する Vr と平行である。

数値計算上は、３組の外積の和、または長さが最も大きい外積を使用する。

回転角 θ は、上記の外積から acos 関数で計算することができる。スカラー部を $\cos(\theta)$ とし、単位ベクトルとして表現した回転軸に $\sin(\theta)$を乗じたものをベクトル部とすることにより、回転を表現する四元数が得られる。

特殊な場合として、回転が不動（回転角ゼロ）の場合、Mr は単位行列であり、**Qr** は、スカラー部が１でベクトル部が全てゼロとなる。

もう一つの特殊な場合として、180 度の回転の場合 $\cos(\theta/2)$がゼロとなり、符号が逆の長さが１のベクトル部を持ち、スカラー部がゼロの **Qr** が同じ回転を定めるため、どちらか片方を選択しなければならない。実は、$\theta + n \times 180$ 度の回転は、θ と同じ座標変換をもたらすことになるため、実用的には、$-90 < \theta <= 90$ または $0 <= \theta < 180$ に切上切捨処理を行った上でデータを管理する。

⑤ 四元数と回転行列の相互変換

三次元の回転行列

$$M=\begin{bmatrix} M11 & M12 & M13 \\ M21 & M22 & M23 \\ M31 & M32 & M33 \end{bmatrix}$$

は、ベクトル V=(Vx, Vy, Vz)を次のように回転する。

MV=(M11Vx+M12Vy+M13Vz,　M21Vx+M22Vy+M23Vz,　M31Vx+M32Vy+M33Vz,)

従って、

X 軸上の原点から 1 の点（1,0,0）は、(M11,M21,M31)に、

Y 軸上の原点から 1 の点（0,1,0）は、(M12,M22,M32)に、

Z 軸上の原点から 1 の点（0,0,1）は、(M13,M23,M33)にそれぞれ移動する。

言い換えると、行列Mは、X軸、Y軸、Z軸の回転移動結果を列として横に並べたものである。

長さ１６の浮動小数の一次元配列として同次行列を表現したリンク・マトリクスにおいては、Mの各成分は、次の配列要素に対応している。

$$M=\begin{bmatrix} M[0] & M[4] & M[8] \\ M[1] & M[5] & M[9] \\ M[2] & M[6] & M[10] \end{bmatrix}$$

長さ 1 の四元数**Q**（Qt,Qx,Qy,Qz）による座標点**V**(0,Vx,Vy,Vz)の回転は、

QV／**Q**として表現される。

回転軸をベクトルで A=(Ax,Ay,Az)、長さ＝１、回転角を θ とすると、

Qt =cos(θ /2), Qx= Ax sin(θ/2),　Qy=Ay sin(θ /2),　Qz=Az sin(θ /2),)である。

X 軸上の座標値 1 の点（1,0,0）を、四元数 **X**=(0,1,0,0)として表すと、

QX = (-Qx, Qt, Qz,-Qy)

QX/Q = **QX**×(Qt,-Qx,-Qy,-Qz) = (-Qx, Qt, Qz,-Qy) ×(Qt,-Qx,-Qy,-Qz)　=

(　-QxQt+QtQx+QzQy-QyQz,

　　QxQx+QtQt-QzQz-QyQy,

　　QxQy+QtQz+QzQt+QyQx,

　　QxQz-QtQy+QzQx-QyQt　) =

　　(　0,　$Qt^2+Qx^2-Qy^2-Qz^2$,　2QtQz+2QxQy,　2QxQz-2QtQy　)

よって、回転行列Mの第 1 列は、四元数の成分を用いて、

M11 = M[0] = $Qt^2+Qx^2-Qy^2-Qz^2$

M21 = M[1] = 2QtQz+2QxQy

M31 = M[2] = 2QxQz-2QtQy

と表せる。同様に、

$$M=\begin{bmatrix} Qt^2+Qx^2-Qy^2-Qz^2 & 2QxQy-2QtQz & 2QxQz+2QtQy \\ 2QxQy+2QtQz & Qt^2-Qx^2+Qy^2-Qz^2 & 2QyQz-2QtQx \\ 2QxQz-2QtQy & 2QyQz+2QtQx & Qt^2-Qx^2-Qy^2+Qz^2 \end{bmatrix}$$

Y軸上の座標値１の点（0,1,0）を、四元数 **Y**=(0,0,1,0)として表すと、

QY = (-Qy, -Qz, Qt, Qx)

QY/Q = QY×(Qt,-Qx,-Qy,-Qz) = (-Qy, -Qz, Qt,Qx) ×(Qt,-Qx,-Qy,-Qz)=

(-QyQt-QxQz+QtQy+QxQz,

QxQy-QtQz-QtQz+QxQy,

QyQy+QtQt-QzQz-QxQx,

QxQz-QtQy+QzQx-QyQt) =

(0, 2QxQy-2QtQz, $Qt^2-Qx^2+Qy^2-Qz^2$, 2QtQx+2QyQz)

よって、回転行列Mの第２列は、四元数の成分を用いて、

M12 = M[4] = 2QxQy-2QtQz

M22 = M[5] = $Qt^2-Qx^2+Qy^2-Qz^2$

M32 = M[6] = 2QtQx+2QyQz

と表せる。同様に、

Z軸上の座標値１の点（0,0,1）を、四元数 **Z**=(0,0,0,1)として表すと、

QZ = (-Qz, Qy, -Qx, Qt)

QZ/Q = **QZ**×(Qz,-Qx,-Qy,-Qz) = (-Qz, Qy, -Qx,Qt) ×(Qt,-Qx,-Qy,-Qz)=

(-QzQt+QyQx-QxQy+QtQz,

QzQx+QyQt+QxQz+QtQy,

QzQy+QyQz-QxQt-QtQx,

QzQz-QyQy-QxQx+QtQt=

(0, 2QxQz+2QtQy, -2QtQx+2QyQz, $Qt^2-Qx^2-Qy^2+Qz^2$)

よって、回転行列Mの第３列は、四元数の成分を用いて、

M13 = M[8] = 2QxQz+2QtQy

M23 = M[9] = -2QtQx+2QyQz

M33 = M[10]= $Qt^2-Qx^2-Qy^2+Qz^2$

と表せる。

M全体で整理すると、

と表せる。

各列をベクトルとして見た時の長さは、四元数のそれぞれの長さの積であり、全て１で

あるから、１である（証明略）。

また、異なる列の内積は、四元数の第一成分として計算され、

（**QX**／**Q**）（**QY**／**Q**）＝**QXY**／**Q**＝**QZ**／**Q**であり、この第一成分（スカラー部）はゼロである。

次に、回転行列Mから四元数Qを求める。

まず、対角成分の和（トレース）は、

$Tr = M11 + M22 + M33 = (Qt^2+Qx^2-Qy^2-Qz^2)+(Qt^2-Qx^2+Qy^2-Qz^2)+(Qt^2-Qx^2-Qy^2+Qz^2) =$

$3\,Qt^2-Qx^2-Qy^2-Qz^2 = 3\,Qt^2-(1-Qt^2) = 4\,Qt^2-1$ により、

$Qt = sqrt(Tr + 1)/2$

が求められる。対角行列の場合（無回転）、Tr は３で、Qt=1.0 となる。180 度回転させるような場合には、Tr は－１で、Qt=0.0 となる。

Qt が十分大きい場合、

$M21-M12=(2QtQz+2QxQy)-(2QxQy-2QtQz)=4QtQz$

$M13-M31=(2QtQy+2QxQz)-(2QxQz-2QtQy)=4QtQy$

$M32-M23=(2QtQx+2QyQz)-(2QyQz-2QtQx)=4QtQx$

よって、Qx、Qy、Qz は、

$Qx = (M32-M23)/4Qt$

$Qy = (M13-M31)/4Qt$

$Qz = (M21-M12)/4Qt$

として算出することができる。

回転角が 180 度付近では、Qt が小さな値となり、ゼロ除算エラーが生じる。その場合、Qt=0 として、M から Qt を削除すると、

となる。なお、Qt=0 のとき、Qx2+Qy2+Qz2=1.0 となる関係を用いた。

対角成分から Qx,Qy,Qz を直接求めることができ、

```
Qt = 0.0;
Qx = sqrt(M11*0.5 + 0.5);
Qy = sqrt(M22 *0.5+ 0.5);
Qz = sqrt(M33 *0.5+ 0.5);
```

として四元数 **Q** を求めることができる。

$$M = \begin{bmatrix} Qx^2-Qy^2-Qz^2 & 2QxQy & 2QxQz \\ 2QxQy & -Qx^2+Qy^2-Qz^2 & 2QyQz \\ 2QxQz & 2QyQz & -Qx^2-Qy^2+Qz^2 \end{bmatrix}$$

$$= \begin{bmatrix} 2Qx^2-1.0 & 2QxQy & 2QxQz \\ 2QxQy & 2Qy^2-1.0 & 2QyQz \\ 2QxQz & 2QyQz & 2Qz^2-1.0 \end{bmatrix}$$

回転するベクトルの差分ベクトルの外積として回転軸を求めることもできる。X 軸上の座標点を示す **X**(0,1,0,0)と回転後の **QX/Q** の差分を成分で表示すると、

QX/Q - X =(0, $Qt^2+Qx^2-Qy^2-Qz^2-1$, 2QxQy+2QtQz, 2QxQz-2QtQy)

同様に **Y**(0,0,1,0)と回転後の **QY/Q** の差分は、

QY/Q - Y =(0, 2QxQy - 2QtQz, $Qt^2-Qx^2+Qy^2-Qz^2-1$, 2QyQz+2QtQx)

この積は、途中を省略すると(-4QxQy, 4QxQz, 4QyQz, $4\ Qz^2$)＝ 4 Qz(-QxQy/Qz, Qx,Qy,Qz)となり、ベクトル部は回転軸と平行である。

⑥ 四元数Qrによる携帯端末の姿勢の記述

地表に固定された、東をＸ軸、北をＹ軸、上方をＺ軸とするローカルな座標系を考える。

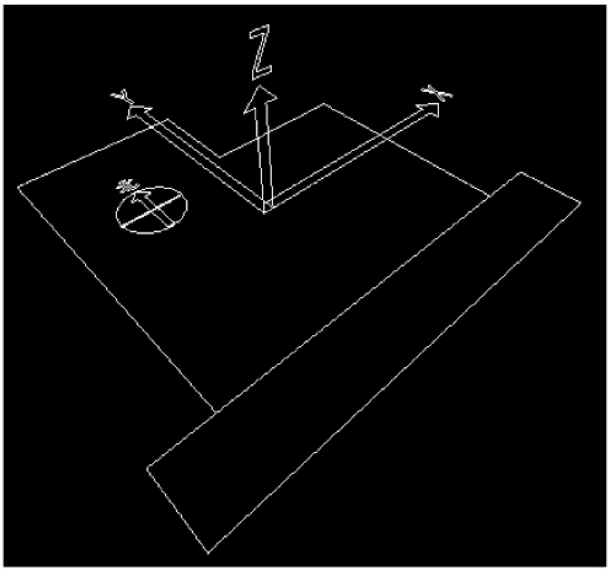


図 4-4-16 敷地の座標系

携帯端末を一つの剛体と見なすと、この物体に固定された物体座標軸を定義することができる。画面および背面カメラの軸線をＶ軸、画面上方を U 軸、画面右手をW軸とする。加速度センサ、および磁気センサは、この座標系において常に同じ向きを計測している。

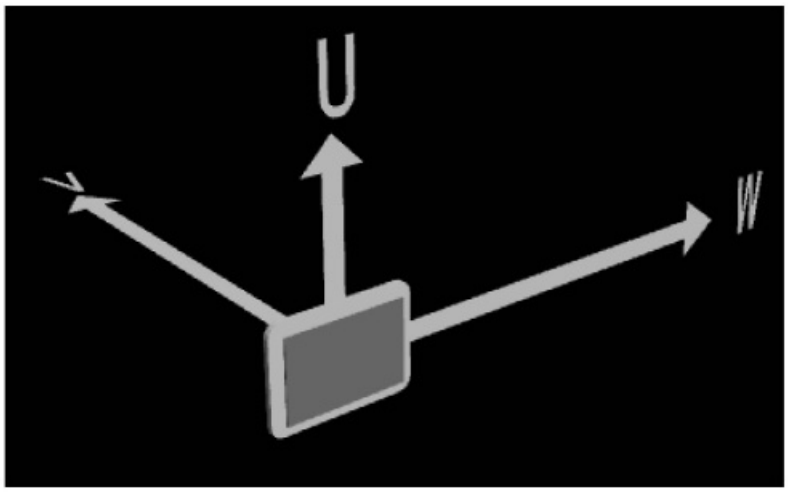


図 4-4-17 携帯端末の座標軸

端末を、地面と水平にディスプレイが上面、背面カメラが下面となるように置き、画面の縦が南北で上が北となるように置いた時の姿勢を基本姿勢とすると、この時U、V、Wの各軸をローカルな座標系で表現すると、{0,1,0,},{0,0,-1},{1,0,0}である。

この状態を基本姿勢とすると、任意の向きに三次元回転させた携帯端末の姿勢は、回転行列により表現され、よってこの回転行列に対応する四元数 Q_r によって表現できる。

加速度センサはU、V、Wの３軸に対応した計測値を出力するため、これに基づいて、携帯端末の座標系による下の向きを把握することができる。同様に、磁気センサは、このU、V、Wの３軸に対応した計測値を出力する。一般に日本付近では磁北の向きは東に偏り、かつ水平よりも少し下を向いている。そこで、本処理系においては、まず計測された加速度のベクトルと磁気のベクトルの内積を計算して東の向きとし、これと加速度のベクトルの内積を再計算して北のベクトルとしている。このようにして、センサ計測値から求めた、携帯端末の姿勢を算出し、基本姿勢において計測された姿勢からの回転として表現する四元数 Q_m として、時々刻々変化する携帯端末の現在の姿勢を表現することができる。

更に、携帯端末が置かれた場所での加速度の向きの真下とのずれ、磁北の真北とのずれを補正値とした時、この補正値もまた座標変換を表す四元数 Q_c として記述することができるため、この値を計測された Q_m に乗じることにより正しい姿勢 Q_r を、基本姿勢からの回転として取得することができる。この補正値は、一度取得しておけば、その近傍において短時間の間有効である。実際には直流で送電されている電気鉄道の線路付近では電車の通過に伴い磁場の変化が生じる。また、帯磁した鉄製工作物の付近でも影響を受ける。

携帯端末自体の帯磁は、磁気センサの計測値に定数項としてバイアスを与える。このバイアスは、携帯端末を 360 度回転させた時に３計測値が描く円軌道（３軸各センサの感度の違いや直角からのずれがある場合、楕円軌道）の中心の原点からの偏差として把握でき、姿勢の算出の前に、センサ計測値に対する固定的な定数項として補正することができる。

加速度センサの計測は、例えば手で支持していることによる伝わる手の震えの振動によ

り擾乱を受ける。このノイズ成分は、計測値の秒単位での時間平均を計算する方法により、姿勢計算前に除去している。

回転を表す四元数 **Qr** の全成分にあるスカラー値 k を掛けても、姿勢 P に対する回転は、

(k **Qr**)P/(k **Qr**)

は k が非ゼロであれば同じであるから、記録のための **Qr** は、長さ 1 に正規化されたものであって構わない。更に計測値された姿勢に対する補正後の姿勢 **Qr** の成分の内、スカラー成分 t は、ベクトル成分から$\sqrt{1-x^2-y^2-z^2}$として容易に計算できることから、データとしてはベクトル成分だけを保存しておけば、以後の処理に必要となる携帯端末の姿勢を記述する情報としては十分である。

このベクトル成分は、半径 1 の球内の任意の点に対応する。まったく移動がない基本姿勢に対応する **Qr**＝(1,0,0,0)は、ベクトル(0,0,0)として保存され、これは球の中心に対応する。

例えば、端末を表示画面が天を向き、上が北の基本姿勢に置いて、180 度まで反時計回りに水平回転すると、この回転 **Qr** のベクトル成分は、回転軸である Z 軸に重なる軌道上を球の中心から北極まで移動する。また、時計回りに 180 度まで水平回転すると、南極まで移動する。北極と南極は同じ回転操作を意味し、更に回転させると北極に戻って南下移動し、一周した時点で原点に回帰する。任意の回転軸周りの回転は、中心を通る軸線上の移動であり、例えば X 軸周りの回転は X 軸上の点に対応する。

端末を裏返して、表示画面が地を向き、上が北となる位置に対応する球内の点は、赤道上の経度がゼロの点に対応する。画面が地を向いた水平姿勢のままで端末を回転させると、姿勢に対応した **Qr** のベクトル成分、言い換えると端末を 180 度回転させる回転軸の向きは、単位球の赤道上を一周する（図 4-4-16）。

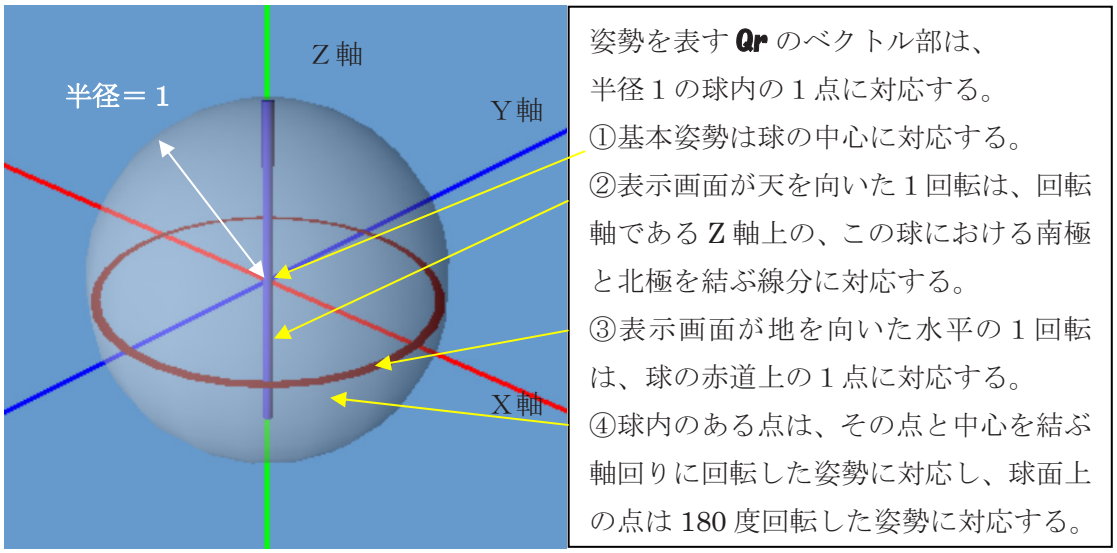


図 4-4-18 携帯端末の姿勢を表現する **Qr** のベクトル 3 成分の変域

VC-3M においては、磁気センサと加速度センサから携帯端末の姿勢を表す四元数を quat クラスで算出し、視点移動に応じて表示処理を行う MMove 関数(mobile.c)に渡している。

＊なお、回転を表す四元数のベクトル成分からの残るスカラー成分を計算することができるため、記録保存や関数への引数としては３成分だけで十分であるが、キャリブレーション等の内部の計算処理において生成した四元数からベクトル成分を抽出する際には、スカラー成分の符号を検査し、負値である場合には、利用するベクトル成分の符号を反転させる必要がある点には注意する必要がある。

⑦ センサ計測値からローカル座標系への変換

携帯端末のセンサにより取得される位置の計測値は、ＧＰＳセンサが取得する緯度、経度、標高である。表示すべきモデル（三次元データをメタファイルに従って解読した結果）は、東をＸ軸、北をＹ軸、上方をＺ軸とする座標系により記述されている。この座標の原点位置が、緯度、経度、標高として表現されている。その記述方法に関しては、本文の２－３～２－５で解説した。

携帯端末がモデルの表示を行うための視点座標を得るためには、緯度、経度、標高の計測値を、モデルを記述しているローカル座標に変換する必要がある。このためには、モデル原点の経度と計測された経度の差から端末位置のＸ座標を、モデル原点の緯度と計測された緯度との差から端末位置のＹ座標を、モデル原点の標高と計測された標高の差から端末位置のＺ座標を計算すれば良い。

Ｘ座標に関しては、ＶＣ－３Ｍにおいては、地表面の湾曲（大円である経線の曲がり）が問題となるほどの範囲を視点移動することはないため、簡便に、経度の差を、その緯度における地軸への垂線の長さに円周率を乗じたものに掛けることにより求めている。Ｙ座標に関しては、緯度の差を、地球の半径に円周率を乗じたものに掛けることにより求めている。Ｚ座標に関しては、この緯度経度の地点におけるジオイド面からの高さを用いている。

歪んだ地球の重力場がもたらす、凹凸のある「等ポテンシャル面」（海抜ゼロの面）であるジオイドの形状に関しては、全球に関する粗いデータと、国土地理院により計測され公開されている日本付近の詳細なデータが利用可能である。これは地球楕円体からの高さの偏差として記述されているため、当該緯度経度の地点におけるジオイドをメッシュデータから補完計算で求め、GPS センサにより計測された高さのジオイド面からの相対的な高さを視点の高さとして使用する。ローカルな座標系の原点の標高は、通常は基準点測量により求められた標高に基づいているため、幾何学的な地球楕円体からの高さではなく、「水準器」により計測された水準面（つまり等重力面）からの高さであり、幾何学的にはジオイド面からの高さに対応するものである。

ジオイドによる補正を行っても、標高に関する計測誤差が相当程度残ることから、本処理系においては、特に敷地周辺の標高基準面を地盤面のデータとしてデータに添付し、緯度経度に相当する地点の視点高さを、地盤面の高さから取得し、これに標準的な視点の高さ（1.5m 程度）を加えた高さを視点の高さとして表示する方法を用意した。この方法は、地盤嵩上げ前の過去の地盤面での表示等の実用的な目的のためにも応用することができる。

このために、VC-3M の処理系においては、表示に使用する視点の高さの決定に際して、計測されキャリブレーションにより補正された水平位置を含む面を、「地面」の属性（& GROUND）が設定されているオブジェクトを構成する面から検索し、この地点の高さをこ

の面の頂点のＺ値から補間計算し、このような面が複数存在する場合にはＺ値の内で最も大きい値をもって、視点の高さを求めるための地面の高さとしている。地面の属性が設定されている面がその地点に存在しなければ、センサとジオイドから計算された標高を表示のために使用する。

このほか、VC-3M においては、２－１に解説したように、現場における表示状態に基づいて、直ちに水平位置や高さの計測誤差を補正する「キャリブレーション」の手段を提供している。これらのキャリブレーションや誤差補正に使用するデータの記憶および計算には、四元数を用いている。

団地等の図面や CAD データから敷地や建物のデータを作成する場合、平面直角座標系で記述されたデータを使用することが多い。この座標系は単に原点と回転角が異なるデカルト座標系ではなく、原点位置で地球楕円体に接した水平面への射影として水平位置が求められたものに、ジオイド面からの高さが垂直位置として記述された、非ユークリッド幾何となっているが、日本国内の場合には、全域が１９のエリアに区分され測量業務等には支障の無い制度が提供されている。

本処理系においては、世界測地系における緯度経度標高から、１９系の内の一つの平面直角座標系を指定して、XYZ 値を算出する処理を cci_quat.c の中に用意し、メタファイルからライブラリ関数 quat ike2wld(int, quat)により利用できるようにした。この変換処理は､国土地理院のホームページから公開されていた BASIC プログラムを元に C 言語による計算に書き換え、５ｍメッシュ数値地図標高のメタデータとして添付されていた四隅の座標値と緯度経度の値を用いて検算した。

注意すべき点は、西暦 2000 年を境に、以前の国家座標系（日本測地系）から世界測地系に更新されている点、および東日本大震災により、東日本で数ｍに及ぶ大きな誤差が発生している点である。日本周辺のように地殻変動が活発な地域においては、土地に固定されたオブジェクトは土地と共に変形していくため、例えば位置を確認するための杭（三角点）や建物も移動していく。これに対応するために、三角点の位置が計測し直される。このとき計測された変動を全てのオブジェクトの座標値に反映させて修正を掛けることは煩雑であるため、平面直角座標系から緯度経度標高に変換した計算結果（緯度経度標高）に対して、メッシュ毎の補正値を加える方法で修正を行っている。

奥尻島の場合について見ると、青苗に一等三角点が１カ所存在する。この三角点に関する測量結果等の履歴は、国土地理院において閲覧することができる。

明治３３(1900)年７月にこの三角点は設置された。

明治３８(1905)年７月に測量された結果は、

Ｎ４２° 3’１６．３４２”　Ｅ１３９° ２７’０９．８８８”

X = -215,774.72　Y = -65.982.28　H = 15.88

でこの値は、平成５年までそのまま使われた。

平成５(1993)年７月の北海道南西沖地震の後、１１月４日に改測された結果は、

Ｎ４２° 3’１６．３３９”　Ｅ１３９° ２７’０９．７７５”

X = -215,774.790 Y = -65,982.889 H = 15.205

となっており、南に 7cm, 西に 60.9cm 移動し、高さは 67.5cm 下がった。

（復興計画のために作成された基本図（1:2500）や、災害公営住宅の図面は、この座標系に基づいて作成されている。）

平成１２(2000)年には、世界測地系への換算が行われ、この一等三角点は、

Ｎ４２° ０３’ ２５．３２７８” Ｅ１３９° ２６’ ５７．２６３９”

X = -215,518.495 Y = -66,277.898 H = 15.21m

に改められた。

この例からわかるように、測地系の切り替えが行われた西暦 2000 を挟んだ二つの時期の平面直角座標系の数値を比較するためには、

新しい座標値→世界測地系における緯度経度への変換→変動パラメータによる補正（緯度経度）→日本測地系における緯度経度への変換→旧国家座標系という手順で変換を行う必要がある。

東日本大震災の区域に関しては大きな変動が生じ、国土地理院によるパラメータが公開されたが、その後の余効変動によるゆっくりとした土地の移動は 2016 年時点でも続いている。従って、過去に存在したオブジェクト（それを記述するローカル座標の原点）の記録作成時点における位置が緯度経度標高の値として保存されている場合、それを未来のある時点における現状（背景）の中に表示するためには、２地点間における土地の移動を反映させた方が実用的と考えられ、その変換処理は未来の利活用時点における処理系と土地の移動の記述方法反映方法に委ねられる。

リスト 4-4-6 二次元の行列を二つの回転と一つのスケールに分解する処理（三角関数）

```
/*****************************************
 *  2d.c (151023, by H.Kobayashi)        *
 *  二次元行列をPDRに分解する            *
 *  角度をパラメータとした微分による     *
 *****************************************/

#include "stdafx.h"

FILE *rp,*wp;//バイナリファイルから乱数を読み込む
char *fname="debug/2005.exe";

void error(char *mes){
          printf("#error[%s]¥n");
}

void printmat(float m[][2]){
          int i,j;
          for(i=0;i<2;i++){
                    for(j=0;j<2;j++){
                              printf("%32f ",m[i][j]);
                    }
                    printf("¥n");
          }
}

void pmat(char *kata, float m[][2]){
          int i,j;
          printf("mat[%s]:¥n",kata);
          for(i=0;i<2;i++){
```

```
                        for(j=0;j<2;j++){
                                printf("%16f ",m[i][j]);
                        }
                        printf("¥n");
                }
}

void initmat4(float m[][2]){
                m[0][0] = 10.1f;
                m[1][1] = 0.9f;
                m[1][0] = 0.2f;
                m[0][1] = 0.1f;
}

void initmat(float m[][2]){
                int i,j;
                for(i=0;i<2;i++){
                        for(j=0;j<2;j++){
                                m[i][j] = (float) j-i+1;
                        }
                }
}

void initmat3(float m[][2]){//右度伸長
                m[0][0] = m[1][1] = 1.0f;
//              m[0][1] = m[1][0] = 0.6f;//成功
//              m[0][1] = m[1][0] = 0.9f;//成功(45度の細長に)
//              m[0][1] = m[1][0] = 0.99f;//成功(45度の針状に)
//              m[0][1] = m[1][0] = 2.0f;//失敗(残差６)→Detで解決
                m[0][1] = m[1][0] = 1.0f;//失敗(エラー)右度の線上に潰れる場合{1,1;1,1}→D2=0で解決
}

void initmat2(float m[][2]){//横滑り
                m[0][0] = m[1][1] = 1.0f;
                m[0][1] = -2.0f; //{1,2;0,1}
                m[1][0] = 0.0f; //22.5度
//              m[0][1] = 0.0f; //{1,0;2,1}
//              m[1][0] = -2.0f; //22.5度
//              m[1][0] = -10.0f;//5.65度,残差.000003
//              m[0][1] = 1.2f;//-29.5度
//              m[0][1] = 1.0f;//-31.7度
//              m[0][1] = -1.0f;//31.7度
//              m[0][1] = 0.f;
                m[1][0] = 0.1f;
}

void initmat1(float m[][2]){//単純伸縮無回転
                m[0][0] = 3.0f;
                m[1][1] = 1.0f;
                m[0][1] = m[1][0] = 0.0f;
}

float cosd(float deg){
                return (float)cos(deg * (float)atan(1.0) / 45.0);
}

float sind(float deg){
                return (float)sin(deg * (float)atan(1.0) / 45.0);
}

void initmat0(float m[][2]){//単純回転無伸縮
                float S,C;
                C = (float) cosd(30.0);
```

```
          S = sind(30.0);
          m[0][0] = C;
          m[1][1] = C;
          m[0][1] = -S;
          m[1][0] = S;
}

void randmat3(float m[][2]){ //大きな数値
          m[0][0] =  2470.68f;
          m[0][1] = -8550.02f;
          m[1][0] = 12751.81f;
          m[1][1] = 17503.44f;
}

void randmat2(float mat[][2]){
          mat[0][0] = 0.343234f;
          mat[0][1] = 0.787890f;
          mat[1][0] = 0.637897f;
          mat[1][1] = 0.936891f;
}

void randmat1(float mat[][2]){
          mat[0][0] = 0.343234f;
          mat[0][1] = 0.787890f;
          mat[1][0] = 0.637897f;
          mat[1][1] = 0.936891f;
}

void randmat0(float mat[][2]){
          mat[0][0] = 1.0f;
          mat[0][1] = 0.01f;
          mat[1][0] = 0.0f;
          mat[1][1] = 1.0f;
}

void samplemat1(float mat[][2]){
          mat[0][0] = 1.0f;
          mat[0][1] = 1.0f;
          mat[1][0] = 0.0f;
          mat[1][1] = 1.0f;
}

void samplemat2(float mat[][2]){
          mat[0][0] = 1.0f;
          mat[0][1] = 2.0f;
          mat[1][0] = 0.0f;
          mat[1][1] = 1.0f;
}

#define _CRT_SECURE_NO_WARNINGS

float frand(){
          int v;
//        if(!rp) rp = fopen(fname,"rb");
          if(!rp) fopen_s(&rp, fname,"rb");
          if(!rp){
                    printf("fopen(%s) error at frand¥n",fname);
                    exit(140704);//stdlib.h
          }
          fread(&v,sizeof(int),1,rp);
          return ((float)(v))*0.001f;
}
```

```
void randmat(float mat[][2]){
          mat[0][0] = frand();
          mat[0][1] = frand();
          mat[1][0] = frand();
          mat[1][1] = frand();
}

float rad2deg(float r){
          return r * 45.0f / (float) atan(1.0);
}

float deg2rad(float r){
          return r * (float) atan(1.0)/ 45.0f;
}

void calcd(float m[][2]){
          float a,b,c,d;
          float d1,d2,base,route,det;
          float abcd2, ac2,bd2,abcd;
//        float v,vx,vy;
          a = m[0][0];
          b = m[0][1];
          c = m[1][0];
          d = m[1][1];
          det = a*d - b*c;
          ac2 = a*a + c*c;
          bd2 = b*b + d*d;
          abcd = a*b + c*d;
          abcd2 = ac2 + bd2;
          base = 0.5f * abcd2;
    route = 0.5f * (float)sqrt(abcd2 * abcd2 - 4.0f * (ac2 * bd2 - abcd*abcd));
          d1 = (float)sqrt(base + route);
          if(det < 0.f) d2 = -(float)sqrt(base - route);
          else d2 = (float)sqrt(base - route);
          printf("D1=%f, D2=%f¥n",d1,d2);
}

void inv(float m1[][2], float m2[][2]){
          float det;
          det = m2[0][0]*m2[1][1] - m2[0][1]*m2[1][0];
          m1[0][0] = m2[1][1] / det;
          m1[1][1] = m2[0][0] / det;
          m1[0][1] = - m2[0][1] / det;
          m1[1][0] = - m2[1][0] / det;
}

void mult(float m1[][2], float m2[][2], float m3[][2]){
          int i,j;
          float w[2][2];
          for(i=0;i<2;i++){
                    for(j=0;j<2;j++){
                              w[i][j] = m2[i][0]*m3[0][j] + m2[i][1]*m3[1][j];
                    }
          }
          for(i=0;i<2;i++){
                    for(j=0;j<2;j++){
                              m1[i][j] = w[i][j];
                    }
          }
}

void ten(float tm[][2], float m[][2]){//転置
          tm[0][0] = m[0][0];
```

```
		tm[0][1] = m[1][0];
		tm[1][0] = m[0][1];
		tm[1][1] = m[1][1];
}

void getdiv2(float m[][2], float q1[][2], float d[][2], float q2[][2]){
		//定石手法
		float s[2][2],tm[2][2];
		ten(tm,m);
		mult(s,tm,m); //正値対称
}

int checkeps(float v){
		if(EPS<v) return 0;
		if(v<-EPS) return 0;
		return 1;
}

int isnol(float m[][2]){
		if(!checkeps(m[0][0])) return 0;
		if(!checkeps(m[0][1])) return 0;
		if(!checkeps(m[1][0])) return 0;
		if(!checkeps(m[1][1])) return 0;
		return 1;
}

void ident(float m[][2]){
		m[0][0]=m[1][1]=1.0f;
		m[0][1]=m[1][0]=0.0f;
}

void nol(float m[][2]){
		m[0][0]=m[1][1]=m[0][1]=m[1][0]=0.0f;
}

void getdiv(float m[][2], float q1[][2], float d[][2], float q2[][2]){
/*M=Q2*D*Q1の形に分解する*/
		float K,C2,C,S,/*T,*/Dx1,Dy1,D1,Dx2,Dy2,D2,Det;
		float Ax,Ay; //長軸の傾き
		float T,deg;
		//K:写像後の円の長軸を与えるパラメータ
		calcd(m);

		if(isnol(m)){//ゼロ行列なら
				ident(q1);//無回転
				ident(q2);
				nol(d);//伸縮はゼロへ
				return;
		}
		Det = m[0][0]*m[1][1] - m[0][1]*m[1][0]; //体積倍率に相当。負値あり
		printf("体積倍率:%f¥n",Det);
		Ay = 2.0f*(m[0][0]*m[0][1] + m[1][0]*m[1][1]); //ab + cd
		Ax = (m[0][0]*m[0][0] - m[0][1]*m[0][1] + m[1][0]*m[1][0] - m[1][1]*m[1][1]); //aa -bb + cc - dd
		if(checkeps(Ax)){
//			T = deg2rad(45.0);
				printf("２θが鉛直¥n");
				C2 = 0.0f;
				K = 0.0f; //とりあえず
		}else{
				K = Ay / Ax;
				T = (float)atan(K);
				printf("２θの向き：%f度¥n",rad2deg((float)atan(K)));
				if(Ax < 0.f)
```

```
                        C2 = -1.0f / (float)sqrt( 1.0 + K*K );
                else
                        C2 = 1.0f / (float)sqrt( 1.0 + K*K );
                T = (float)acos(C2);//debug
        }
        C = (float)sqrt(0.5 + 0.5*C2);//写像前の円周上の点
        if(Ay < 0.f) S = (float)-sqrt(0.5 - 0.5*C2);//sin符号は同じ
        else S = (float)sqrt(0.5 - 0.5*C2);
        //元の単位球第一軸
        Dx1 = m[0][0] * C + m[0][1] * S;//写像後の楕円の軸
        Dy1 = m[1][0] * C + m[1][1] * S;
        D1 = (float)sqrt(Dx1*Dx1 + Dy1*Dy1);//軸の長さ（の半分）
        printf("D1の向き：%f度(D1=%f)¥n",rad2deg((float)atan(S/C)),D1);
        //元の単位球第二軸
        Dx2 = m[0][0] * (-S) + m[0][1] * C;//写像後の楕円の軸(C,S)->(-S,C)
        Dy2 = m[1][0] * (-S) + m[1][1] * C;
        D2 = (float)sqrt(Dx2*Dx2 + Dy2*Dy2);//軸の長さ（の半分）
        printf("D2の向き：%f度(D2=%f)¥n",rad2deg((float)atan(C/(-S))),D2);
        if(!checkeps(D1*D2 - Det)){
                printf("体積誤差¥n");
        }
        if(D2 < D1){ //D1が長軸
                if(!checkeps(D1)){ //長軸に長さあり：最も一般の場合１
                        if(Det < 0.f) D2 = -D2;
                        q1[0][0] = q1[1][1] = C;
                        q1[0][1] = S;
                        q1[1][0] = -S;
                        q2[0][0]=q2[1][1]=Dx1/D1; //cos
                        q2[0][1] = -Dy1/D1;
                        q2[1][0] = Dy1/D1;
                        d[0][0] = D1;
                        d[1][1] = D2;
                        d[0][1] = d[1][0] = 0.0;
                        deg = rad2deg((float)atan(Dy1/Dx1));
                        printf("q2: %lf¥n", deg);
                }else{ //２軸とも長さゼロ:点に縮退
                        ident(q1);
                        ident(q2);
                        nol(d);
                }
                if(Det<0.f){//第二軸の長さがだけゼロ
                }else{
                }
        }else{ //D2が長軸
                if(!checkeps(D2)){
                        if(Det < 0.f) D1 = -D1;
                        q1[0][0] = q1[1][1] = -S;
                        q1[0][1] = C;
                        q1[1][0] = -C;
                        q2[0][0]=q2[1][1]=Dx2/D2; //cos
                        q2[0][1] = -Dy2/D2;
                        q2[1][0] = Dy2/D2;
                        d[0][0] = D2;
                        d[1][1] = D1;
                        d[0][1] = d[1][0] = 0.0;
                        deg = rad2deg((float)atan(Dy2/Dx2));
                        printf("q2: %lf¥n", deg);
                }else{
                        ident(q1);
                        ident(q2);
                        nol(d);
                }
        }
```

```
}

float diff(float m1[][2], float m2[][2]){
          int i,j;
          float d;

          d = 0.0;
          for(i=0;i<2;i++){
                    for(j=0;j<2;j++){
                              d += (m1[i][j]-m2[i][j])*(m1[i][j]-m2[i][j]);
                    }
          }
          return (float)sqrt(d);
}

void checkdiv(float m[][2], float q1[][2], float d[][2], float q2[][2]){
//mが、q1*d*q2に等しいことを検証する
          float w[2][2];
          mult(w,q2,d);//Q1*D
          mult(w,w,q1);//Q1*D*Q2
          printf("----------checkdiv start---------¥n");
          pmat("q1",q1);
          pmat("d",d);
          pmat("q2",q2);
          pmat("m:before",m);
          pmat("m:after",w);
          printf("残差：%16f¥n",diff(m,w));
          printf("----------checkdiv end---------¥n");
}

showsifatm(float m[][2]){
          int i;
          float d,s,c,Dx,Dy,D;
          for(i=0;i<180;i++){
                    d = (float)(i)*3.14f/180.0f;
                    c = (float)cos(d);
                    s = (float)sin(d);
                    Dx = m[0][0] * c + m[0][1] * s;
                    Dy = m[1][0] * c + m[1][1] * s;
                    D = (float)sqrt(Dx*Dx + Dy*Dy);
                    printf("dot[%d] = (%lf,%lf) %lf¥n",i,Dx,Dy,D);
          }
}

outlssg(float m[][2],char*COLOR){
          float div,x,y,Px,Py,rate;
          float xmax,ymax;
          float thmin,thmax,rmin,rmax,r;
          int i;
          //axis
          thmax = rmax= xmax = ymax = 0.f;
          thmin = rmin = 1e10;

          fprintf(wp," GRAPH=GROUP();¥n");
          rate = (float)atan(1.0)/4.50f;
          for(i=0;i<=36;i++){//円周
                    div = (float)i * rate;
                    x = (float)cos(div);
                    y = (float)sin(div);
                    Px = m[0][0]*x + m[0][1]*y;
                    Py = m[1][0]*x + m[1][1]*y;
                    fprintf(wp,"P%d = COORD(%f,%f,0);¥n",i,Px,Py);
                    fprintf(wp,"V%d = VERTEX(P%d);¥n",i,i);
```

```
                        if(xmax < Px) xmax = Px;
                        if(ymax < Py) ymax = Py;
                        r = (float)sqrt(Px*Px + Py*Py);
                        if(r < rmin){
                                thmin = div;
                                rmin = r;
                        }
                        if(rmax < r){
                                thmax = div;
                                rmax = r;
                        }
                }
                printf("Rmin(%f度) = %f, Rmax(%f度)= %f¥n",rad2deg(thmin),rmin,rad2deg(thmax),rmax);
                fprintf(wp,"LG=LINE(V0");
                for(i=1;i<=36;i++){
                        fprintf(wp,",V%d",i);
                }
                fprintf(wp,");¥n");
                fprintf(wp,"GROUP_LINE(GRAPH,LG);¥n");
                //axis
                fprintf(wp," AXIS=GROUP();¥n");
                fprintf(wp," P0=COORD(0,0,0);   V0=VERTEX(P0);¥n");
                fprintf(wp," PX=COORD(%f,0,0);VX=VERTEX(PX);¥n",xmax);
                fprintf(wp," PY=COORD(0,%f,0);VY=VERTEX(PY);¥n",ymax);
                fprintf(wp," LA=LINE(VY,V0,VX);¥n");
                fprintf(wp," MAXIS=MATERIAL(YELLOW);¥n");
                fprintf(wp," LINE_MATERIAL(LA,MAXIS);¥n");
                fprintf(wp," GROUP_LINE(AXIS,LA);¥n");
                //sqare
                fprintf(wp," SQUARE=GROUP();¥n");
                fprintf(wp," P10=COORD(%f,%f,0);V10=VERTEX(P10);¥n",m[0][0],m[1][0]);//(1,0)
                fprintf(wp," P01=COORD(%f,%f,0);V01=VERTEX(P01);¥n",m[0][1],m[1][1]);//(2,1);
                fprintf(wp," P11=COORD(%f,%f,0);V11=VERTEX(P11);¥n",m[0][0]+m[0][1],m[1][0]+m[1][1]);//(1,3)
                fprintf(wp, "LS=LINE(V0,V10,V11,V01,V0);¥n");
                fprintf(wp, "LINE_COLOR(LS,%s);¥n",COLOR);
                fprintf(wp, "GROUP_LINE(SQUARE,LS);¥n");
}

int main2d(int argc, char* argv[])
//int main(int argc, char* argv[])
{
                float M[2][2],Q1[2][2],Q2[2][2],D[2][2];
#if 1 //単発,グラフ作成
                float DQ1[2][2];
//              initmat0(M);//単純回転
//              initmat1(M);//単純伸縮
//              initmat2(M);//単純剪断
//              initmat3(M);//体積が負に
//              initmat4(M);//無変化＋微小
//              randmat1(M);//小さなランダム：失敗
//              samplemat1(M);//報告[例１]
                samplemat2(M);//報告[例２]
                getdiv(M,Q1,D,Q2);
                checkdiv(M,Q1,D,Q2);
//              wp = fopen("graph.geo","wt");
                fopen_s(&wp,"graph.geo","wt");
                fprintf(wp," RED=COLOR(1,0,0);¥n");
                fprintf(wp," BLUE=COLOR(0,0,1);¥n");
                fprintf(wp," GREEN=COLOR(0,1,1);¥n");
                outlssg(Q1,"BLUE");
                mult(DQ1,D,Q1);
                outlssg(DQ1,"GREEN");
                outlssg(M,"RED");
```

```
		fclose(wp);
#else //乱数連発
		int i;
		for(i=0;i<10;i++){
			if(i==4){ //特定のエラーの追跡
				printf("[B]");
			}
			printf("TRIAL[%d]¥n",i);
			randmat(M);
			getdiv(M,Q1,D,Q2);
			checkdiv(M,Q1,D,Q2);
		}
#endif
		if(rp) fclose(rp);
//		getchar();
//		printf("3.14e-20f=%ef¥n",3.14e-20f);
		return 0;
}

/*******************************************
 * 開発記録                                *
 * cardano 141208 として作成したものから抜粋*
 * keikan/next/tenrsor                      *
 *******************************************/
```

リスト 4-4-7 三次元の行列を二つの回転と一つのスケールに分解する処理（三次方程式）

```
/*************************************
 *  cardano.c (160211)               *
 *  三次元行列をPDRに分解する        *
 *  固有値問題を三次方程式として解く  *
 *************************************/
/*
  main関数から、test5を起動する。
  test5では、解析する行列をM[3][3]に格納した上で、mat2qagを呼び出す。
    mat2qag(M,Q1,A,Q2);
  最初の回転がQ1,スケールがA、第二の回転がQ2として返される。
    qag2mat(Q2,A,Q1)を呼び出して、元の行列を検算する。
  printmat3が、行列をコンソール出力する。

*/


#include "stdafx.h"

double det(double m[3][3]){//m[3][3]の行列式を返す
		return m[0][0]*m[1][1]*m[2][2] + m[0][1]*m[1][2]*m[2][0] + m[0][2]*m[1][0]*m[2][1]
		     - m[0][0]*m[1][2]*m[2][1] - m[0][2]*m[1][1]*m[2][0] - m[0][1]*m[1][0]*m[2][2];
}

void MT(double t[3][3], double m[3][3]){//T = M tM
		int i,j;
		for(i=0;i<3;i++){
			for(j=0;j<3;j++){
				t[i][j] = m[i][0]*m[j][0] + m[i][1]*m[j][1] + m[i][2]*m[j][2];
			}
		}
}

void TM(double t[3][3], double m[3][3]){//T = tM M
		int i,j;
		for(i=0;i<3;i++){
```

```
                    for(j=0;j<3;j++){
                              t[i][j] = m[0][i]*m[0][j] + m[1][i]*m[1][j] + m[2][i]*m[2][j];
                    }
          }
}

void cp(double u[3], double v[3]){//u[3] = v[3]
          int i;
          for(i=0;i<3;i++){
                    u[i] = v[i];
          }
}

void ad(double u[3], double v[3]){//u[3] += v[3]
          int i;
          for(i=0;i<3;i++){
                    u[i] += v[i];
          }
}

void sb(double u[3], double v[3]){//u[3] -= v[3]
          int i;
          for(i=0;i<3;i++){
                    u[i] -= v[i];
          }
}

double l2(double v[3]){ //  | v |
          return v[0]*v[0]+v[1]*v[1]+v[2]*v[2];
}

double in(double u[3], double v[3]){// u・v
          return u[0]*v[0]+u[1]*v[1]+u[2]*v[2];
}

void ex(double w[3], double u[2], double v[2]){//w = u×v
          w[0] = u[1]*v[2] - u[2]*v[1];
          w[1] = u[2]*v[0] - u[0]*v[2];
          w[2] = u[0]*v[1] - u[1]*v[0];
}

void nm(double v[3]){//ベクトルを正規化する
          int i;
          double r;

          r = sqrt(l2(v));
          for(i=0;i<3;i++) v[i]/=r;
}

void Mmul(double C[3][3], double A[3][3], double B[3][3]){//C = AB
          int i,j;
          for(i=0;i<3;i++){
                    for(j=0;j<3;j++){
                              C[i][j] = A[i][0]*B[0][j] + A[i][1]*B[1][j] + A[i][2]*B[2][j];
                    }
          }
}

void Mt(double M[3][3]){//転置
          int i,j;
          double v;
          for(i=0;i<3;i++){
                    for(j=0;j<i;j++){
```

```
                                v = M[i][j];
                                M[i][j] = M[j][i];
                                M[j][i] = v;
                        }
                }
}

void rod(double w[3], double u[3], double v[3]){//wに同軸上のベクトルを同じ向きに加算する
                double vec[3];//,r;
                vec[0] = u[1]*v[2] - u[2]*v[1];
                vec[1] = u[2]*v[0] - u[0]*v[2];
                vec[2] = u[0]*v[1] - u[1]*v[0];
                if( l2(w) == 0.0 ) cp(w,vec);
#if 1
                else ad(w,vec);
#else //最初の値が非常に小さい負の値なら、全体が負の値となってしまう。
                else if(0.0<=(r=in(w,vec)))
                        ad(w,vec);
                else
                        sb(w,vec);
#endif
}

void printmat3(double m[3][3]);

void eigenvector(double m[3][3], double eigenvalue, double vec[3]){
//mの固有値eigenvalueに属する固有ベクトルを計算しvecに格納
                int i;
                double u[3],v[3],w[3];

                for(i=0;i<3;i++){
                        u[i] = m[0][i];
                        v[i] = m[1][i];
                        w[i] = m[2][i];
                        vec[i] = 0.0;
                }
                u[0] -= eigenvalue;
                v[1] -= eigenvalue;
                w[2] -= eigenvalue;
                rod(vec,u,v);
                rod(vec,v,w);
                rod(vec,w,u);
                nm(vec);
                //printf("eigenvecor(%lf) returns:[%lf, %lf, %lf]¥n",eigenvalue,vec[0],vec[1],vec[2]);
}

int dif(double x, double y){//差がe-4より大なら１
                double r;
                r = fabs(x-y);
                if(1e-4 < r) return 1;
                return 0;
}

void getortho(double prim[3], double second[3], double third[3]){
                //第一のベクトルprimから、これと直交する二つを求める
                int i,im;
                double r;
                r = fabs(prim[0]);
                im = 0;
                for(i=1;i<3;i++){
                        if(fabs(r < prim[i])){//向きが最も似ている座標軸を探す
                                im=i;
                                r = fabs(prim[i]);
```

```
                }
        }
        switch(im){
                case 0://x軸=>第二の軸をYZ平面上に
                        second[0] = 0.0;
                        second[1] = prim[2];
                        second[2] = -prim[1];
                        if(!dif(0.0, l2(second))){//第一の軸がピタリX軸に一致するなら
                                second[1] = 1.0;
                                second[2] = 0.0;
                        }else{
                                nm(second);//規格化
                        }
                        ex(third, prim, second);
                        break;
                case 1:
                        second[1] = 0.0;
                        second[2] = prim[0];
                        second[0] = -prim[2];
                        if(!dif(0.0, l2(second))){//第一の軸がピタリY軸に一致するなら
                                second[2] = 1.0;
                                second[0] = 0.0;
                        }else{
                                nm(second);//規格化
                        }
                        ex(third, prim, second);
                        break;
                case 2://z軸=>第二の軸をXY平面上に
                        second[2] = 0.0;
                        second[0] = prim[1];
                        second[1] = -prim[0];
                        if(!dif(0.0, l2(second))){//第一の軸がピタリZ軸に一致するなら
                                second[0] = 1.0;
                                second[1] = 0.0;
                        }else{
                                nm(second);//規格化
                        }
                        ex(third, prim, second);
                        break;
        }
}

void galois(double vecs[3][3]){
//３固有値から求めた回転行列には、２４の回転形と、その鏡像がありうる
//無回転（単位行列）に近い形を最適解として選択する
        int i,j;
        for(i=0;i<3;i++){
                if(vecs[i][i] < 0.0){
                        for(j=0;j<3;j++){
                                vecs[i][j] = -vecs[i][j];
                        }
                }
        }
}

int eigenmat(double m[3][3], double eigenvalue[3], double vecs[3][3]){
//mの３固有値から、３固有ベクトルvecsを求める。重根の場合には適当に決める。●完成度が低い
        int rank,i,uniq,j;

        rank = 0,uniq = 0;
        if( dif(eigenvalue[0],eigenvalue[1])) rank++;
        else uniq = 2;
        if(dif(eigenvalue[1],eigenvalue[2])) rank++;
```

```
		else uniq = 0;
		if(dif(eigenvalue[2],eigenvalue[0])) rank++;
		else uniq = 1;
//この時、rankは０か２か３
// 0 （０組の固有値が異なる）→単純膨張なら、単位行列で可
// 2 （二組の固有値が異なる）→回転体
// 3 （全ての固有値が異なる）→一般の場合
//●固有値がゼロの場合（縮退）の場合分けをごっちゃにしている
		if(rank == 3){//全ての固有値が異なる：通常の歪
			for(i=0;i<3;i++){
				eigenvector(m,eigenvalue[i], vecs[i]);
			}
			galois(vecs);
		}else if(rank == 2){//球→回転体の変形（uniqは唯一の独自固有値）
			printf("回転体変形¥n");//uniqを使うのは、rankが２の場合だけ
			eigenvector(m,eigenvalue[uniq],vecs[uniq]);//他と異なる固有値に帰属する固有ベクトル
			getortho(vecs[uniq],vecs[(uniq+1)%3],vecs[(uniq+2)%3]);
		}else{//等方変形
			printf("等方的膨張収縮変形¥n");
			for(i=0;i<3;i++){
				for(j=0;j<3;j++){//vecsは、単位行列に設定（何でも良い）
					if(i==j) vecs[i][j]=1.0;
					else vecs[i][j] = 0.0;
				}
			}
		}
		return rank;
}

double realpart3(double x, double y){
		//x + iy の複素数三乗根の実部を返す
		double r,t;
		t = atan(y/x);
//		printf("atan(%lf/%lf)=%lf¥n",y,x,t);
		r = sqrt(x*x + y*y);
		if(x < 0.0) return -pow(r, 1.0/3.0) * cos(t/3.0);
		return pow(r, 1.0/3.0) * cos(t/3.0);
}

double u(double p, double q){
		double u3,d;
		d = q*q + p*p*p;
		if(d < 0.0)//実数の根が３存在する場合
			return realpart3(-q, sqrt( -d));
		u3 = -q + sqrt( d );//これで良いはず
		if(0.0 <= u3) return pow(u3,1.0/3.0);
		else return -pow(-u3, 1.0/3.0);
}

double v(double p, double q){
		double v3,d;
		d = q*q + p*p*p;
		if(d < 0.0)//実数の根が３存在する場合
			return realpart3( -q, sqrt(-d));
		v3 = -q - sqrt( d );
		if(0.0 <= v3) return pow(v3, 1.0/3.0);
		else return -pow(-v3, 1.0/3.0);
}


//x3 + 3*p*x + 2*q = 0 の形の方程式
int judge(double p, double q){//151023 停留点間区間がX軸と交差するかで方程式の判定
//戻り値1:1根3:3根0:三重2:二重
```

```
	double x1,x2; //微分係数ゼロの点
	double floor, ceiling;
	if(!dif(p, 0.0)){
		if(!dif(q, 0.0))
			return 0;//三重
		return 1; //解は一つ、残りは虚数
	}
	if(0.0 < p) //単調増加
		return 1; //解は一つ、残りは虚数、
//微分は、xx + 3*p = 0 よって、xx = +-sqrt(-p)
	x2 = sqrt(-p);
	x1 = -x2;
//停留点における式の値
    ceiling = x1*x1*x1 + 3.0*p*x1 + 2.0*q;
	floor   = x2*x2*x2 + 3.0*p*x2 + 2.0*q;
	if(!dif(0.0, ceiling))
		return 2; //二重根
	if(!dif(0.0, floor))
		return 2; //二重根
	if(ceiling < 0.0)
		return 1; //解は一つ、残りは虚数解・異常
	if(0.0 < floor)
		return 1; //解は一つ、残りは虚数解・異常
	return 3; //解は三つ、固有値問題の一般通常の場合
}

double x1(double p, double q){//一つの解を見つける
	double x,xnext,d,dnext,slop;
	int count, jj;

	jj = judge(p, q);//解の数（３が通常。１なら不適切問題）

	count = 0;
	x = u(p,q) + v(p,q);
	//ここで解析的計算は終わるが、以下の検算で残差を検出し微修正する。
again:
	dnext = x*x*x + 3.0*p*x + 2.0*q; //dは残差
	if(dnext == 0.0){
		return x;//残差が無ければ終了
	}
	if(count == 0) d = dnext;//初回
	else if(fabs(d) < fabs(dnext)){//解から遠ざかる
		return x;
	}
//この方程式の微分
	slop = 3.0*(x*x + p);
	xnext = x - d / slop;
	count ++;
	if(xnext == x){//解の近傍で振動する場合にはこの条件は成立しない
		return x;
	}
	x = xnext;
	goto again;
}

#if 0
void check(double a, double b, double c, double x){//デバッグ用
	double d;
	d = x*x*x + a*x*x + b*x + c;
}
#endif

int cardano(double a, double b, double c, double XS[3]){
```

```
	//x3 + ax2 + bx + c = 0  の形の方程式を解く
	double p,q,X,D;
	//printf("cardano(%lf,%lf,%lf);¥n",a,b,c);
	//printf(" i.e.  x3 %+lf x2 %+lf x %+lf = 0¥n",a,b,c);
   // 変数変換を行い、x3 + 3*p*x + 2*q = 0 の形に方程式を整理する
	p = (3.0*b - a*a) / 9.0;
	q = (27.0*c + 2.0*a*a*a - 9.0*a*b)/54.0;
	// 得られた解x1からa/3 を引いたものが元の変数xで表現した解である
	X = x1(p,q);

	XS[0] = X - a/3.0;//一つの基本解を求める
	D = -3.0 * (X*X + 4.0*p);//残り二つの解に関する判別式
	if(!dif(D,0.0)){//重根ある場合
		XS[1] = XS[2] = -0.5*X - a/3.0;
		if(!dif(XS[0],XS[1]))
			return 0; //三重根
		return 2;//二重根が等しい
	}else if(D < 0.0){//残りの二解は虚数成分を含む
		printf(" //cardano解は一つ¥n");//本処理系の場合は、災害
		XS[1] = 0.0;
		XS[2] = 0.0;
		return 1;
	}
//以下、一般の場合（解は３個）
	XS[1] = -0.5*(X - sqrt(D)) - a/3.0;
	XS[2] = -0.5*(X + sqrt(D)) - a/3.0;
	return 3;
}

//行列mの３固有値を計算し、eigen[3]に格納する
void matcardano(double m[3][3], double eigen[3]){
	double a,b,c;//固有方程式の係数
//２乗の係数
	a = -m[0][0] - m[1][1] - m[2][2];
//１乗の係数
	b = m[1][1]*m[2][2] + m[2][2]*m[0][0] + m[0][0]*m[1][1]
	  - m[1][2]*m[2][1] - m[0][2]*m[2][0] - m[0][1]*m[1][0];
//０乗の係数
	c = -det(m);
	cardano(a,b,c,eigen);
}

void printmat3(double m[3][3]){//コンソールにm[3][3]を出力（デバッグ用）
	int i,j;
	for(i=0;i<3;i++){
		for(j=0;j<3;j++){
			printf("%10.2lf",m[i][j]);
		}
		printf("¥n");
	}
}

void setmat(double M[3][3]){//テスト用既知の行列Q,A,GからM=QAGを作成する
	double work[3][3];
	double A[3][3]={3,0,0, 0,2,0, 0,0,1}; //解析結果は、これと一致しなければならない
	double Q[3][3]={0,1,0, -1,0,0, 0,0,1}; // M = QAG
	double G[3][3]={1,0,0, 0,0,1, 0,-1,0};
	Mmul(work,Q,A);
	Mmul(M,work,G);
	printf("[SETMAT]¥n");
	printf("---------Q----------¥n");
	printmat3(Q);
	printf("---------A----------¥n");
```

```
		printmat3(A);
		printf("---------G----------¥n");
		printmat3(G);
}

double anasmat(double M[3][3]){//行列式を、外積・内積の計算で求める。detと同じ事
		double u[3],det;
    // V = M[0]×M[1]、det = (M[0]×M[1])・M[2]
		ex(u,M[0],M[1]);
		det = in(u,M[2]);
		printf("「DET=%lf」¥n",det);
		return det;
}

void printeigen(double e[3]){//３固有値を表示する
		int i;
		printf("EIGEN START¥n");
		for(i=0;i<3;i++){
			printf("%10.2lf¥n",e[i]);
		}
		printf("EIGEN END¥n");
}

/*任意の行列Mを、二つの回転Q1,Q2と拡大Aに分解し、Q1＊A*Q2の形に表現する*/
void mat2qag(double M[3][3], double Q[3][3], double A[3][3], double G[3][3]){
		double mt[3][3];// M * tM
		double eigen[3];//固有値３個
		double AQ[3][3];
		int i,j;

		MT(mt,M);//mt = M * tM
		matcardano(mt,eigen);//行列[3][3]から固有値を求める
		eigenmat(mt,eigen,Q);//３固有値eigenに対応する固有ベクトルを並べた回転行列Qを求める
		for(i=0;i<3;i++){//３固有値の平方根を対角成分とするスケール行列Aを求める（正対角行列）
			for(j=0;j<3;j++){
				if(i!=j) A[i][j] = 0.0;
				else A[i][j] = sqrt(eigen[i]);
			}
		}
		for(i=0;i<3;i++){//AQ=^A^Q （^Aは対角逆数、^Qは転置）
			for(j=0;j<3;j++){
				AQ[i][j] = Q[j][i] / A[j][j];
			}
		}
		Mmul(G,AQ,M);//Gを求める。これで、M=QAGが完成。
}

void qag2mat(double Q[3][3], double A[3][3], double G[3][3]){
		double W[3][3],M[3][3];
		Mmul(W,Q,A);//W=Q*A
		Mmul(M,W,G);//M=W*G=Q*A*G
		printmat3(M);
}

void test5(){ //テスト関数：適当な行列Mを作成し、解析させる
		//ある行列を、QAG分解する
		//(1)まず、必ず分解できる、QAGについて調べる
//		double M[3][3] ={1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,1.9};//適当な場合(det = 0)
//		double M[3][3] ={1.1,1.2,1.3,1.4,1.5,1.6,1.7,1.8,2.0};//適当な場合(det = 0)
//		double M[3][3] ={1.0,0.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0};//例０（無回転無変形）
//		double M[3][3] ={1.5,0.0,0.0, 0.0,1.0,0.0, 0.0,0.0,2.0};//例０（背が伸びる）
//		double M[3][3] ={1.0,2.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0};//例１（二次元）
//		double M[3][3] ={0.0,-0.87,-0.4, 1.2,0.0,0.0, 0.0,-0.5,0.69};//例４（三次元）
```

```
        double M[3][3] ={0.0,1.2,0.0, -0.87,0.0,-0.5, -0.4,0.0,0.69};//例４（三次元）
        double Q1[3][3], A[3][3], Q2[3][3], INV[3][3];
        double det;

//      setmat(M);//既知の答から問題を作成する関数
        printf("[CHALLENGE]¥n");
        printf("--------M--");
        det = anasmat(M);
        printmat3(M);
        mat2qag(M,Q1,A,Q2);
        printf("[RESULT]¥n");
        printf("--------Q--");
        det = anasmat(Q1);
        printmat3(Q1);//これはQAGの内G
        printf("--------A--");
        det = anasmat(A);
        printmat3(A);
        printf("--------G--");
        det = anasmat(Q2);
        printmat3(Q2);//これはQAGの内Q
        printf("-------QAG---------¥n");
        qag2mat(Q1,A,Q2);
        printf("--------G*tG-------¥n");
        MT(INV,Q2);
        printmat3(INV);
//      printf("-------Q2AQ1=QAG----------¥n");
//      qag2mat(Q2,A,Q1);//Mと同じものが再現される筈
}

int main(int argc, char* argv[]){
//      main2d();二次元の回転を三角関数で解く、d.cの方法
        test5();
}

/*****************************************
 * 開発記録                              *
 * cardano 141208 として作成したものから抜粋*
 * keikan/next/tenrsor                    *
 *****************************************/
```

４－５．ビルドと開発環境

（１）はじめに：CPU と OS の略史

本研究において例示的に試作した４種類の利活用処理系は、現時点で広く使用されているデスクトップ型ないしノート型のコンピュータに Windows を OS として搭載しているマシン、およびタブレット型の携帯端末に Android を OS として搭載しているマシンの上で稼働する。三次元グラフィックスを処理することができるマシンと OS は、過去２０年の間に大きく変遷し、アプリケーションを開発するためのコンパイラやリンカを含む「開発環境」も変化してきた。現在普及し広く使われている様態が最終的な姿として将来長く留まるとは考えられない。そこでまず初めに、近過去におけるこれらの変遷を概観しておく。なお、下記の OS や開発環境は旧建設省建築研究所、国総研において景観シミュレーションシステム等の開発に使用されたものであるが、発表年代等については、Wikipedia 等に基づいている。各種 CPU や OS の変遷の全体像を通史的に辿った歴史書は未発見である。

Windows は、インテル社製の 86 系の CPU の上の OS（オペレーションシステム）として主に 1995～2015 年の間広く使用されてきた。それまでの MS-DOS の GUI（グラフィカルユーザーインターフェース）として開発された系列(3.1,95,98,ME)と、カーネルから再構築された WindowsNT の系列(3.5、4.0、2000)がしばらく併存したが統合され、XP(2001)、Vista(2006), 7(2009), 8(2012)を経て 10(2015)に至る。また統合後もサーバ用 OS として、2000、2003、2008、2013 が供給された。Window NT3.5~4.0 は DEC の Alpha(1992)、PowerPC(1991)等の異なる CPU もサポートした。

インテル製 CPU を用いた IBC-PC は初期の MS-DOS(1982-2002)を OS としてコマンドラインによりシングルタスクでプログラムを動作させていた。CPU は 8086(1978)から 80286(1982)までは 16 ビットで動作し Windows3.1(1993)を GUI としていたが、80386(1985)から 32 ビットとなり、486(1989), Pentium(1993)に至り、OS も Windows95、NT から 32 ビットとなった。更に Core 2 から 64 ビットになり、スタートアップ時に 32 ビットか 64 ビットの動作モードを選択できるようになった。これに対応して、WindowsXP および 2003Server 以降、OS も導入時に 32 ビットモードの x86 か、64 ビットモードの x64 かを選択できるようになった。64 ビットの OS を導入した場合であっても、32 ビット CPU 用に開発されたアプリケーションを実行(エミュレート)することはできるが、64 ビット専用の機械語を用いた実行形式は、32 ビットの OS 上では現在動かすことができない。

AMD 社(1969-)は当初セカンドソースとして x86, x64 のアーキテクチャを用いてインテル社製品の互換 CPU を製造していたが、Alpha 用基板に装着可能なバスを有する高性能互換 CPU として Athron(1999)を開発し、以後主に処理速度の面で競争が展開された。大電力を消費するマザーボード（基盤）には大きな冷却機能が装備されるようになった。

オープンソースで開発された OS である LINUX(1994～)は、ワークステーションの OS である UNIX(1969～)をベースに Windows 用として当時普及していた x86 マシンを主なターゲットとして開発された。サーバの OS として広く用いられている。後述の Android(2007

～)も、主たる CPU は異なるが、LINUX をカーネル部分に使用している。

この他 80 年代には、インテル系と拮抗する形でモトローラの 6800(８ビット)、68000（16 ビット）が存在し、Windows と競合する形で Macintosh 等に使用されていたが、機械語を単純化しコンパイラで最適化する RISC 系 CPU の PowerPC がその後普及し、68 系の CPU は専ら工業用制御機器等に使用されるようになった。寿命が長い機器のメンテのために日本のメーカーで CPU の製造は続けられている。

6800 を改良した MOS6502（８ビット）もコマンドライン時代の Apple コンピュータや家庭用ゲーム機に使用され、これを発展させた ARM アーキテクチャの CPU が、携帯電話で普及しスマートフォンや携帯用 PC にも採用された。これに対応して Microsoft により開発された省電力小型機種向けの WindowsRT が開発され、続く CE(1996～)は様々な CPU に対応して.NET Framework の中間言語で走行した。これをベースとした Windows Mobile(2003~12)は、広く普及した ARM 系 CPU を用いた携帯端末の上でのみ走行した。

携帯端末ではキーボードやマウスを使用せずに画面（表示の上のタッチパネル）上のタップ操作で指示や入力を行う。デスクトップ PC においても Windows VISTA 以降は、画面（表示の上のタッチパネル）上のタップ操作を受け付けるインターフェースを提供するようになった。ARM チップを搭載した携帯端末では iPhone の iOS と、（３）で解説する Android が OS として広く使われている。

WindowsXP 以降、インターネットに常時接続する PC が普及するに従い、これをサーバへの攻撃の踏み台などに悪用するマルウェア（ウィルスソフト）が蔓延し、ウィルス対策ソフト（WEB 閲覧やメール添付を通じて進入するファイルの検査や処理）の導入と、OS のセキュリティ・パッチやアップデートによる脆弱性の解消が行われるようになった。

Windows10 は、インテル系 CPU のための最後のバージョンとされ、以後無償でアップデートが行われるとされている。

一つの Windows バージョンの上で問題なく動作するアプリケーションを開発し完成した状態が実現すれば、以後はメンテナンスの段階に入る。メンテナンスのためにはソースコードを修正し、コンパイルしなければならない。このとき、開発環境が更新されているならば、実行形式において生じている問題点を解決することと別に、新たな開発環境に適応するためのソースコードの修正が必要となる場合がある。

（２）開発用言語

①アセンブリ言語

各マシン固有の機械語を、ニモニックで表現したソースコードを編集し、機械語を生成（アセンブル）する。変数アドレス、ジャンプ先やサブルーチンに名前を付けることができる。各 CPU の命令コードセットに固有の開発環境である。実行する CPU とは異なる CPU の上でもアセンブラを実行することができる（クロス・アセンブラ）。小規模な組込用システムの開発を、作業能率のよい高度な OS 上の上で進めることができる。

②FORTRAN

構造解析や住宅統計調査など、数値解析をプログラムする言語として大型計算機の上で使用できた。PC の上でも利用可能である。言語仕様が枯れているため、定番のアルゴリズムを記述するためにはプログラムの寿命が長い。変数＝数式の形で数値代入を表現し、for でループを表現し、if で条件分岐を表現するスタイルは C 言語、BASIC 言語、JAVA 言語等の原型となった。コンパイラ言語として使用されている。

③C 言語

機械語に近い原始的な言語。現在でもシンプルな組込用マイコンのソフト開発や、高度なマシンの OS の深い階層の開発に用いられている。セミコロン「;」を文の区切りとする書法は、様々な言語の原型となった。コンパイラ言語として提供された。

ポインタを用いて、マシン上のメモリや入出力ポートのアドレスに直接アクセスできることから、組込み系などのハードウェアに結びついたシステム開発に用いられている。

一方で、関数型の処理系であることから、ライブラリ関数として特殊な処理を行う関数をパッケージ化することによる拡張が可能である。基本的なコンソール入出力を行う関数も、stdio(standard in/out)ライブラリとして外部化されている。三次元画像表示を行うライブラリである OpenGL もそのような一つである。OpenGL2.0 においては、CPU とは並行して画像処理を専門に行う CPU（画像処理を行うことから特に GPU と呼ばれる）に対するプログラムをシェーダ記述言語（C 言語に類似）で記述して、OpenGL の初期化命令関数の引数として GPU のための C 言語ソースコード文字列を画像処理系に渡し、並行処理の内容をプログラムすることもできる。

動的なメモリ管理を、malloc 関数と free 関数で行う。デバッグにおいて、malloc 関数で取得されたメモリブロックが不要となってから free 関数で 1 度だけ解放され、以後はアクセスされないように丁寧に確認する必要がある。メモリブロックが残されたままポインタが書き換えられ変数が消滅するとメモリリークとなる。ポインタが別の変数にコピーされ、free された後のメモリブロックが参照されると、動作は保証されない。

free 関数で解放された後の断片化された記憶領域の整理、いわゆるガーベージコレクション（GC）は OS に委ねられ、プログラムから要求する関数は用意されていない。メモリの限界に近い大規模な三次元データを扱う場合でも、少なくとも後述の初期の BASIC のように長時間プログラムが停止するような状況は生じていない。

アプリケーションの側から、処理に必要のないタイミングで GC を行うためには、専用の割り当て－解放関数をプログラムするか、Boehm GC のような、フリーで提供されているライブラリを使用する。

本処理系の場合には、利活用実装例のデバッグにおいて、malloc と free をリストでトレースしながら管理し、メモリーリークゼロとしている。また、メタファイルの文法においては、長さが既知の特定のデータファイルを処理できれば十分であることから、静的なメモリ管理のみを用いている。

④BASIC 言語

初期のマイクロコンピュータに、ROM に搭載されプログラミングのために提供され、ラインエディタで入力されたプログラムを直接解釈し実行するインタープリタとして動作した。文は改行を単位とし、行頭には行番号が昇順で付され、GOTO 文でジャンプ、GOSUB でサブルーチン呼び出しを記述した。1980 年代に普及した IBM-PC シリーズには MS-BASIC が、また国産 NEC 社製 PC-9801 シリーズでは、N88BASIC が OS に同梱されていた。後に MS-DOS が標準 OS となってからは、コンパイラ機能も追加され速度が向上した。Windows アプリを開発するために、画面表示とコールバックをプログラムできる VisualBasic が開発された。仕様が大幅に変更され、従来のプログラムはレガシー化した。

マイクロソフト系の事務処理ソフト用のマクロ記述用として、あるいはサーバの機能を記述するスクリプト言語 VBScript として現在でも使用されている。オブジェクト指向の影響を受け、BASIC においても各種の組込みオブジェクトを部品としてプログラミングに利用できる。

変数を型宣言することなく直接代入できる点や、メモリ管理が OS により自動的に行われる点が C 言語とは異なっている。変数の現在の型を知るためには、オブジェクト.GetType()で型オブジェクトを取り出し、GetTypeCode(型オブジェクト)で型のコードを取り出す。

この型は、VC++では、VARIANT 構造体として定義されており、型を示す vt メンバと、様々なデータ型を示すユニオンから成っている。文字列は BSTR 型で、メモリブロックを取得してポインタを変数に持つ。VC-4D において、SQL データベース上のデータの入出力に際して、_variant_t 型の変数を使用している(sqllib.cpp)。

1980 年代の BASIC では、メモリブロックの取得に際してメモリが不足すると集中的なガーベージコレクション（散在する空き領域の整理）が開始され、長時間ユーザの操作に応答しない状態になったが、処理アルゴリズムは改善されている。使用後、オブジェクトを格納した変数=nothing と宣言することにより、ガーベージコレクションの対象であることが明示される。

⑤C++言語

オブジェクトが記述できるように拡張された C 言語で、処理系別の各コンパイラによりマシン語を生成する。独自に定義した数値型に対して、＋や＊等の演算を定義することができる。OS の機能がクラスとして提供され、各種設定やコールバックがメンバ変数、メンバ関数として利用できるため、これを部品として組み合わせて高度な処理を記述するために便利である。C 言語の構造体定義を拡張したようなクラス定義、関数冒頭でない任意の場所での変数宣言、演算子のユーザ定義などが可能である。しかし OS に依存する既存のライブラリクラスやテンプレートを使用するとプラットフォーム（動作環境）が限定され、動く最小限のコードを書けるまでに多くのオーバーヘッドを抱えることになる。

メモリの動的管理には、new　と delete を実行する。malloc – free と混用してもビルドは通るが、交錯しないような慎重なデバッグが求められる。

System::GC により、OS のガーベージコレクションを強制的に起動することができる。

⑥java 言語

中間言語による実行形式を生成するコンパイラである。中間言語を実行するインタープリタ（JRE：Java Runtime Environment 等）がセットアップされた環境で様々な CPU のマシン上で実行することができる。VC-3M の場合には、Android を OS とする arm アーキテクチャの携帯端末の上で動作する。VC-4D の場合には、Windows サーバの上で動作する。

WEB サーバとして apache を使用する場合には、tomcat をセットアップして、web コンテンツの中に埋め込まれたプログラムをサーバ側で実行する。IIS には、java で記述されたスクリプトを実行する機能はない。

組込みライブラリクラスに依存するために、異なる OS への可搬性は低い。

変数は型宣言して用いる。動的メモリ管理は、ガーベージコレクタが自動的に実行する。

⑦javascript

html 文書の中に、<script language="javascript"> プログラム </script>の形で埋め込み、アニメーションを含む様々な表現を可能とした。やがてサーバ側のスクリプトにも利用できるようになった。多く WEB ブラウザ上で動作するため、クライアントのファイルシステムに攻撃的なアクセスできないよう制約が課されている。

⑧C#言語

マイクロソフトにより.net framework のためのプログラム言語として開発された。様々なマシンの上で動作する仮想マシンである共通言語基盤(CLI ： Common Language Infrastructure)のための共通中間言語(CIL)による共通実行形式(CLR ： Common Language Runtime)として生成する。C++言語や BASIC 言語とモジュールを共有できる。

⑨コーディングレス言語

1980 年代から、画面に表示されたモジュール間を GUI 上で結線することによりシミュレーションから 3D 表示までを行う実行形式を生成するような、プログラムレス言語は存在していた。Macintosh 上の Think C コンパイラ(1986-)や、Windows のための Visual Studio においても、操作画面の設計（テキストボックスや、ボタンなどのコントロールの配置）を、マウスによる画面操作で行うためのリソースエディタが用意され、結果はリソースファイル（テキストファイル）に保存され、ビルドの中に含めることができた。近年では、RAD(rapid application development)という概念で括られている。IDE（エディタ、コンパイラ、リンカ、デバッガ等を一体化した統合開発環境）の多くは RAD である。

処理アルゴリズムを表現したフローチャートを UML 形式でテキストファイルに保存し、それに基づいて実行形式を自動生成するような仕組みも研究されている。

⑩バッチコマンド

1970 年代、UNIX 上で Bourne shell や C shell をインターフェースとしてファイル操作、プログラム実行などを指示するコマンドを入力し、解釈実行する実行形式が利用できた。一連のコマンドを束ねたスクリプトを作成し、それを実行することにより、定型の処理を

簡便に起動することができた。

PC 上では初期においては BASIC 言語のコマンドを用いてディスクの初期化等を行っていたが、80 年代前半の MS-DOS 以降は、UNIX のシェルスクリプトのようなコマンドを解釈実行する command.com が導入された。WindowsNT では、cmd.exe がこの役割を担った。2006 年からは、PowerShell が利用可能となり、Get-Process(現在実行中のプロセスをリスト表示する)のような機能が加わった。2015 年現在では、cmd.exe と powershell.exe のどちらも利用可能であるが、command.com は windows xp 以降のマシンには搭載されていない。

Android 上でコマンドラインを入力するシェルは提供されていないが、terminal emulator というアプリケーションが提供されている。

⑪SQL 言語

データベースに関する処理を系統化した言語として広く使用されている。データベースの下に複数のテーブルを作成し管理する。構築、挿入、書き換え、削除等の処理を記述できる他、一群の処理をストアードプロシージャとしてライブラリ化、パッケージ化することができる。バックアップやアクセス権管理などの機能が高度に発達している。一方で、基本的な文法において、文字列の処理が素朴であり、データ文字列の中にプログラムとして解釈可能なコードを埋め込む攻撃が行われる。

⑫HTML 言語、XML 言語、PostScript、Tex 言語等

人間が読むための様式付きの文書を、機械にも解釈できる文字列として表現する言語である。当初学術文献記録用に開発された HTML 形式は、その後 WEB コンテンツ配信用に広く普及した。レーザープリンタの制御用に発達した PostScript 言語が、現在では PDF 文書として記録保存用にも用いられている。

XML は、テキスト文書の構造をタグで表示することにより、いわばメタファイルを文書自体の中に埋め込んだようなデータを作成可能とした。更に、文書全体を解釈し読み込むライブラリを使用することにより、解読処理を容易にプログラムすることができる。

現在では多くの三次元データが、XML 形式で記述され、独自の拡張子を付して提供されている。但し、一部データが欠損したようなファイルや、メモリに搭載しきれない巨大なデータを処理する場合には工夫が必要である。

⑬その他

本研究に直接関係しないため割愛したが、下記のような諸言語には注意を払う必要がある。古くから現在まで人工知能の研究に用いられている LISP は、普及した AutoCAD のユーザによる機能拡張のためにも使用されている。スタック演算処理系である Forth は、機械語レベルのデバイスドライバから、高度な処理までをシームレスに組み上げる体系である。COBOL 言語は事務処理系として現在でもロングテールで使用されている。

システム設定のための小道具的な言語も含めると、数万以上のプログラム言語が歴史上存在したと推定される。

（３）利活用処理系試作における開発環境

本書に収録した４の実装形態の開発にあたり、共通する基幹部分（コンパイラ等）のソースコードを枯れた C 言語により作成し、異なる OS や開発環境の上で実行形式を作成して可搬性を検証した。一方、利活用処理系における画面表示やセンサ値の取得やデータベース入出力等の実装には、各プラットフォーム上で現在広く使われている java、MFC 等を使用した。

最近の開発環境と言語は、素早くアプリケーションを作成することを目標に置いており、バグの無いプログラムをじっくりと作成することはほとんど眼中にない。多くのマシンがインターネットに接続され、悪意のある攻撃のリスクに晒される中、OS や開発環境を含む各種アプリケーションにおいても頻繁なアップデートが求められている。このため、OS や開発環境は短期的に変化しつつあり、データの長期保存と利活用を目的とする本研究にとって、現時点における開発環境の詳細を解説することは本質的な意味がないように思われる。当面、多くの解説資料が出版物として、あるいは WEB 上のコンテンツとして入手可能である。しかしながら、十数年の後には、これらは既に過去のものとなっており、保存データを利活用するための処理系を開発しようとする時点においては、例えば 1960 年代の真空管テレビの詳細な解説資料古書にも似たようなものとなっている可能性がある。

このような西暦 2000 年前後に日進月歩の状態を体験した開発環境や OS について、利用者の側の記録を残しておくことは、データの長期保存のための技術の必要性の一端を逆にアンチテーゼとして説明することにもなると考える。

本研究においては、プログラミングにはネットワークに接続しないスタンドアロンのマシンを使用し、このマシンを用いたメール交信や WEB ブラウズは行っていない。このため、OS や開発環境のアップデートも行っていない。静かで安定した環境下での開発を行った。ネットワークへのアクセスは、開発中のソースコードや研究資料を持たない、通信専用のマシン（旧式のハードに最新の OS を搭載）を傍らに置いて、開発環境とは独立させた。開発環境にもバグは存在する。しかし、そのことを理解した上で、着手から概成までを一定の環境で進めることは、問題を単純化する。概成した後に初めて、完成に向けて可搬性を検証するために他の開発環境への移植を試みた。

本書で解説した４の実装例に関しては、以下の開発環境を使用した。

①VC-1C(Windows 上のコンソールアプリ)

Microsoft 社製 Visual Studio2005(C 言語)

②VC-2V

Microsoft 社製 VisualStudio2005(C/C++言語)

③VC-3M

Android NDK(C コンパイラ、Windows 上で動作するクロスコンパイラ)

Android SDK(Java 言語コンパイラ、Windows 上で動作するクロスコンパイラ)

ECLIPSE(エディタなどを含む統合開発環境)

携帯端末エミュレータ（Windows 上で携帯端末の画面を模擬するテスト環境）

④VC-4D

Microsoft 社製 VisualStudio2005（C 言語）

Java（WindowsServer2003 上のサーバースクリプト）

（４）Windows と VisualStudio 開発環境

① 様々な開発ツール

Windows のためのアプリケーション開発環境は BASIC インタープリタのためのラインエディタ、手書きでリソースファイルを作成した段階から、エディタ、コンパイラ、リンカ、デバッガを一つのアプリケーション上で総合的に起動する環境に進んだ。Turbo-C、Borland-C、Visual C++等が C 言語のソースコードの開発に広く使用され、マイクロソフト社製の Visual Studio は、C、C++、BASIC、JAVA、html 文書、SQL、WEB アプリケーションなどの複数言語をサポートした。

日本語などのアジア言語への対応。

様々なバージョンの OS への対応。

ソースコードの管理。

インストーラの作成。

バージョン管理

② 統合開発環境

開発環境の更新は、主に新たに導入された OS の機能を活用するために行われてきた。この時期、キーボードとマウス、というしばらく定番となっていた入力装置に加えて、タッチパネルが普及し、ノート型 PC においても利用可能となった。画面に触れるタップ操作に対応してアプリケーションが動作するためには、これに対応したライブラリを用いてビルドを行う必要がある。

本研究において、Windows 系のアプリケーションの開発には、C++コンパイラなどを含む Microsoft 社製の Visual Studio 2005 を使用した。2015 年現在では、VisualStudio2015 がリリースされている。あえて 2005 年にリリースされたバージョンの開発環境を用いた理由は、それ以前のコンパイラに比して完成度が高く、本研究において以後のバージョンで提供されている新しい機能を必要としていないこと、および Windows98 から 8 に至る比較的幅広い OS の上で実行可能である点にある。具体的には、Windows98/Me/NT4 で動作する Win32 プログラムを作成できる最後のバージョンであり、かつ 64 ビットの命令も生成することのできる最初のバージョンである。コンパイラのバージョンは 8.0 で、対応する _MSC_VER コードは 1400 である。

開発環境としてユーザ（プログラマ）が操作する各種画面を提供するプログラムは、devenv.exe という名称の統合開発環境(IDE: Integrated Development Environment)である。これには、ソースコードをテキストファイルとして編集するためのエディタや、ユー

ザが操作に使用する画面をデザインするためのリソースエディタの機能が統合され、一つの画面でコードの編集からデバッグまでを実行することができる。

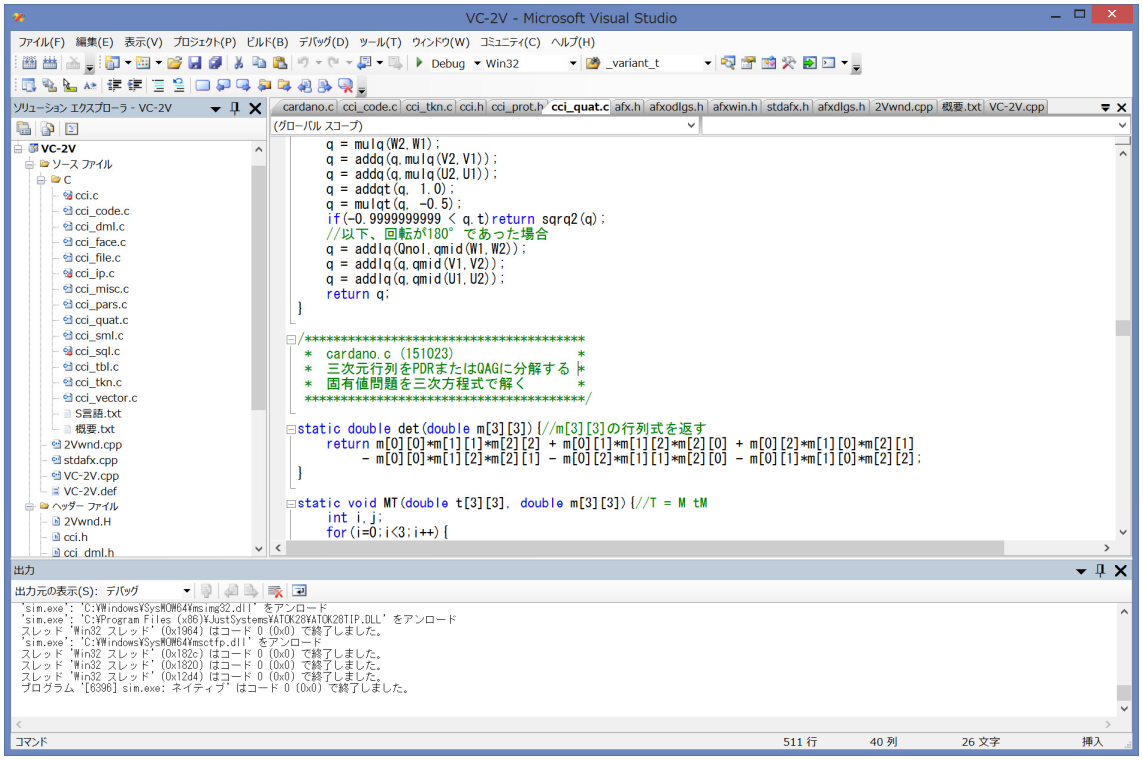


図 4-5-1 VS2005 操作画面

開発全体（ソリューション）を記録するマスターファイルとして、「.sln」という拡張子のテキストファイルが作成され、この中に複数のプロジェクトが登録されている。一つの実行形式(.exe または.dll)を生成する個々のプロジェクトについては、例えば C/C＋＋言語による開発プロジェクトを示す、「.vcproj」という拡張子の XML 形式のファイルで定義されている。

これらのファイルには、冒頭に開発環境のバージョンが冒頭に記載されているため、VS2008 以降の新しい環境に移行すると自動的に書き換えられる。古い開発環境にビルド一式を戻すためには、このバージョン番号を手書きで戻さなければならない。

プロジェクト毎の定義ファイル(vcproj)には、プロジェクトを構成する各種ファイル（ソース、ヘッダー、リソース等）が、相対アドレスで記述されており、、ID 番号が付されており（例えば「ProjectGUID="{DDD6B078-A15E-4377-BC21-ACD6C62A933D}"」）、プロジェクト一式は、各種ソースコードを含むサブディレクトリ全体を同一マシン上の他のディレクトリまたは、同じ開発環境がセットアップされた他のマシン上の異なるディレクトリパスにコピーしてもビルドを行うことができ、この ID 番号は変化しない。コンパイル、リンクの様々な付帯条件も記述されており、その内容は編集画面においてプロジェクトのプロパティを設定した内容と一致している。

③コンソールアプリ

利活用処理系の例として作成した VC-1C、およびサーバ側の処理の一部を担う VC-4D は、操作画面を持たないコンソールアプリとして開発した。コンソールアプリの場合には、シンプルな処理でありながら、ファイルアクセス、数値演算を行うことから、C 言語に付帯して提供されるライブラリ関数を使用する。ライブラリ関数は、LIBC.LIB（スタティックリンクの場合）等のライブラリに含まれ、関数形が”stdio.h”などのヘッダーファイルとして提供されている。

リスト 4-5-1 VC-1C のメイクファイル(.vcproj)

```
<?xml version="1.0" encoding="SHIFT_JIS"?>
<VisualStudioProject Keyword="Win32Proj" RootNamespace="cci2005"
ProjectGUID="{F4591204-300A-444A-AC20-DB2DF4CBA3A8}" Name="cci2005" Version="8.00" ProjectType="Visual C++">
  <Platforms>
    <Platform Name="Win32"/>
  </Platforms>
  <Configurations>
    <Configuration Name="Debug|Win32" CharacterSet="0" ConfigurationType="1"
     IntermediateDirectory="$(ConfigurationName)"
     OutputDirectory="$(SolutionDir)$(ConfigurationName)">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool"/>
      <Tool Name="VCCLCompilerTool" DebugInformationFormat="3" Detect64BitPortabilityProblems="true"
     WarningLevel="3" BrowseInformation="1" UsePrecompiledHeader="0" RuntimeLibrary="3"
     BasicRuntimeChecks="3" MinimalRebuild="true"
     PreprocessorDefinitions="WIN32;_DEBUG;_CONSOLE;_CRT_SECURE_NO_DEPRECATE"
     AdditionalIncludeDirectories=""C:¥@keikan¥NEXT¥SIM209¥SRCNT¥LIBRARY¥IMPORT=""""
     Optimization="0"/>
      <Tool Name="VCManagedResourceCompilerTool"/>
      <Tool Name="VCResourceCompilerTool"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="1" GenerateDebugInformation="true"
     LinkIncremental="2" OutputFile="c:¥@keikan¥ksim¥bin¥VC-1C.exe"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>
    <Configuration Name="Release|Win32" CharacterSet="1" ConfigurationType="1"
     IntermediateDirectory="$(ConfigurationName)"
     OutputDirectory="$(SolutionDir)$(ConfigurationName)" WholeProgramOptimization="1">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool"/>
      <Tool Name="VCCLCompilerTool" DebugInformationFormat="3" Detect64BitPortabilityProblems="true"
     WarningLevel="3" UsePrecompiledHeader="0" RuntimeLibrary="2" BasicRuntimeChecks="3"
     PreprocessorDefinitions="WIN32;NDEBUG;_CONSOLE;_AFXDLL"
     AdditionalIncludeDirectories=""C:¥@keikan¥NEXT¥SIM209¥SRCNT¥LIBRARY¥IMPORT=""""
     Optimization="0"/>
      <Tool Name="VCManagedResourceCompilerTool"/>
```

```
      <Tool Name="VCResourceCompilerTool"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="1" GenerateDebugInformation="true"
      LinkIncremental="1" OutputFile="c:¥@keikan¥ksim¥bin¥VC-1C.exe" EnableCOMDATFolding="2"
     OptimizeReferences="2"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool" EmbedManifest="false"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>
  </Configurations>
  <Files>
    <Filter Name="ソースファイル" UniqueIdentifier="{4FC737F1-C7A5-4376-A066-2A32D752A2FF}"
     Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx">
      <File RelativePath=".¥cci.c"> </File>
      <File RelativePath=".¥cci_code.c"> </File>
      <File RelativePath=".¥cci_dummy.c"> </File>
      <File RelativePath=".¥cci_face.c"> </File>
      <File RelativePath=".¥cci_file.c"> </File>
      <File RelativePath=".¥cci_ip.c"> </File>
      <File RelativePath=".¥cci_misc.c"> </File>
      <File RelativePath=".¥cci_pars.c"> </File>
      <File RelativePath=".¥cci_quat.c"> </File>
      <File RelativePath=".¥cci_tbl.c"> </File>
      <File RelativePath=".¥cci_tkn.c"> </File>
      <File RelativePath=".¥cci_vector.c"> </File>
      <File RelativePath=".¥ccimake.bat"> </File>
    </Filter>
    <Filter Name="ヘッダーファイル" UniqueIdentifier="{93995380-89BD-4b04-88EB-625FBE52EBFB}"
     Filter="h;hpp;hxx;hm;inl;inc;xsd">
      <File RelativePath=".¥cci.h"> </File>
      <File RelativePath=".¥cci_kanji.h"> </File>
      <File RelativePath=".¥cci_prot.h"> </File>
    </Filter>
  <Globals> </Globals>
</VisualStudioProj
```

リスト 4-5-2 VC-4D のメイクファイル

```
<?xml version="1.0" encoding="shift_jis"?>-<VisualStudioProject Keyword="MFCProj" RootNamespace="VC4D"
ProjectGUID="{42EF7CD0<1158<4513<A93C<608B6563DF01}" Name="VC-4D" Version="8.00" ProjectType="Visual C++">
  <Platforms>
    <Platform Name="Win32"/>
  </Platforms>
  <ToolFiles> </ToolFiles>
  <Configurations>
    <Configuration Name="Debug|Win32" CharacterSet="2" UseOfMFC="2" ConfigurationType="1"
      IntermediateDirectory="$(ConfigurationName)"
      OutputDirectory="$(SolutionDir)$(ConfigurationName)">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool" ValidateParameters="false" MkTypLibCompatible="false"
     PreprocessorDefinitions="_DEBUG"/>
      <Tool Name="VCCLCompilerTool" PreprocessorDefinitions="WIN32;_WINDOWS;_DEBUG"
     DebugInformationFormat="4" Detect64BitPortabilityProblems="true" WarningLevel="3"
     UsePrecompiledHeader="0" RuntimeLibrary="3" BasicRuntimeChecks="3" MinimalRebuild="true"
```

```
      AdditionalIncludeDirectories="..¥..¥..¥library¥import;..¥..¥..¥..¥include;..¥..¥flow¥C;.¥"
      Optimization="0"/>
      <Tool Name="VCManagedResourceCompilerTool"/>
      <Tool Name="VCResourceCompilerTool" PreprocessorDefinitions="_DEBUG"
      AdditionalIncludeDirectories="$(IntDir)" Culture="1041"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="2" GenerateDebugInformation="true"
      LinkIncremental="2" AdditionalDependencies="odbc32.lib odbccp32.lib"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>

    <Configuration Name="Release|Win32" CharacterSet="2" UseOfMFC="1" ConfigurationType="1"
      IntermediateDirectory="$(ConfigurationName)"
      OutputDirectory="$(SolutionDir)$(ConfigurationName)" WholeProgramOptimization="1">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool" ValidateParameters="false" MkTypLibCompatible="false"
      PreprocessorDefinitions="NDEBUG"/>
      <Tool Name="VCCLCompilerTool" PreprocessorDefinitions="WIN32;_WINDOWS;NDEBUG"
      DebugInformationFormat="3" Detect64BitPortabilityProblems="true" WarningLevel="3"
      UsePrecompiledHeader="0" RuntimeLibrary="0" MinimalRebuild="false"
      AdditionalIncludeDirectories="..¥..¥..¥library¥import;..¥..¥..¥..¥include;..¥..¥flow¥C;.¥"/>
      <Tool Name="VCManagedResourceCompilerTool"/>
      <Tool Name="VCResourceCompilerTool" PreprocessorDefinitions="NDEBUG"
      AdditionalIncludeDirectories="$(IntDir)" Culture="1041"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="2" GenerateDebugInformation="true"
      LinkIncremental="1" AdditionalDependencies="nafxcw.lib libcmt.lib" EnableCOMDATFolding="2"
      OptimizeReferences="2" IgnoreDefaultLibraryNames="nafxcw.lib;libcmt.lib"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>
  </Configurations>
  <References> </References>
  <Files>
    <Filter Name="ソースファイル" UniqueIdentifier="{4FC737F1<C7A5<4376<A066<2A32D752A2FF}"
      Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx">
      <File RelativePath="..¥..¥Flow¥C¥cci_vector.c"> </File>
      <File RelativePath=".¥samples.c"> </File>
      <File RelativePath=".¥SqlConnector.cpp"> </File>
      <File RelativePath=".¥sqllib.cpp"> </File>
      <File RelativePath=".¥stdafx.cpp">
        <FileConfiguration Name="Debug|Win32">
          <Tool Name="VCCLCompilerTool" UsePrecompiledHeader="1"/>
        </FileConfiguration>
        <FileConfiguration Name="Release|Win32">
          <Tool Name="VCCLCompilerTool" UsePrecompiledHeader="1"/>
        </FileConfiguration>
```

```
      </File>
      <File RelativePath=".¥VC-4D.cpp"> </File>
      <File RelativePath=".¥VC-4DDlg.cpp"> </File>
      <Filter Name="C">
        <File RelativePath="..¥..¥Flow¥C¥cci_code.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_face.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_file.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_misc.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_pars.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_quat.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_sql.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_tbl.c"> </File>
        <File RelativePath="..¥..¥Flow¥C¥cci_tkn.c"> </File>
      </Filter>
    </Filter>

    <Filter Name="ヘッダーファイル" UniqueIdentifier="{93995380<89BD<4b04<88EB<625FBE52EBFB}"
     Filter="h;hpp;hxx;hm;inl;inc;xsd">
      <File RelativePath="..¥..¥Flow¥C¥cci.h"> </File>
      <File RelativePath="..¥..¥Flow¥C¥cci_dml.h"> </File>
      <File RelativePath="..¥..¥Flow¥C¥cci_sql.h"> </File>
      <File RelativePath=".¥Resource.h"> </File>
      <File RelativePath=".¥SqlConnector.h"> </File>
      <File RelativePath=".¥sqllib.h"> </File>
      <File RelativePath=".¥stdafx.h"> </File>
      <File RelativePath=".¥VC-4D.h"> </File>
      <File RelativePath=".¥VC-4DDlg.h"> </File>
    </Filter>

    <Filter Name="リソースファイル" UniqueIdentifier="{67DA6AB6<F800<4c08<8B7A<83BB121AAD01}"
     Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx;tiff;tif;png;wav">
      <File RelativePath=".¥res¥VC-4D.ico"> </File>
      <File RelativePath=".¥VC-4D.rc"> </File>
      <File RelativePath=".¥res¥VC4D.rc2"> </File>
    </Filter>
    <File RelativePath=".¥ReadMe.txt"> </File>
  </Files>

  -<Globals>
    <Global Name="RESOURCE_FILE" Value="VC-4D.rc"/>
  </Globals>
</VisualStudioProject>
```

④Windows アプリケーション

MFC 以前の Windows SDK を用いた開発においては、WinMain 関数の中で、リソース ID とコールバック関数を引数として DialogBox 関数を呼び出して操作画面を作成し、そこに対するユーザの操作は、コールバック関数の呼び出しの中で、メッセージの種類により switch 文で処理を振り分けるスタイルが採られていた。これらの処理は C 言語でも記述可能である。

C++言語では、ダイアログボックスは、一つの Windows オブジェクトのクラスとして整理され、構築、各種のサービス提供から破却まで、メンバ関数によりプログラムできるようになった。Windows の体系を構成する様々のオブジェクトをクラスに整理したものが、MFC(Microsoft Foundation Class)。例えば、ファイルを選択するポップアップ画面を開く CFileDialog というクラスは、afxdlgs.h というヘッダーで定義され、プログラム中で使用

することができ、リンカによりこの処理を行う MFC ライブラリがプログラムに追加される。

リスト 4-5-3 VC-2V のメイクファイル

```
<?xml version="1.0" encoding="shift_jis"?>
<VisualStudioProject Keyword="MFCDLLProj" RootNamespace="VC-2V"
ProjectGUID="{DDD6B078-A15E-4377-BC21-ACD6C62A933D}" Name="VC-2V" Version="8.00" ProjectType="Visual C++">
  <Platforms>
    <Platform Name="Win32"/>
  </Platforms>
  <ToolFiles> </ToolFiles>
  <Configurations>
    <Configuration Name="Debug|Win32" CharacterSet="2" UseOfMFC="2" ConfigurationType="2"
      IntermediateDirectory="$(ConfigurationName)"
     OutputDirectory="$(SolutionDir)$(ConfigurationName)">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool" MkTyp
     LibCompatible="false" PreprocessorDefinitions="_DEBUG"/>
      <Tool Name="VCCLCompilerTool"  PreprocessorDefinitions=
     "WIN32;_WINDOWS;_DEBUG;_USRDLL;RDH_NONE;MULTI_LANG;PLUGIN;_CRT_SECURE_NO_DEPRECATE"
     DebugInformationFormat="4" Detect64BitPortabilityProblems="true" WarningLevel="3"
     UsePrecompiledHeader="0" RuntimeLibrary="3" BasicRuntimeChecks="3" MinimalRebuild="true"
     AdditionalIncludeDirectories=
     "..¥..¥library¥import;..¥..¥SIM;"..¥..¥SIM¥@merge="""";..¥..¥..¥include;..¥flow¥C"
     Optimization="0"/>
      <Tool Name="VCManagedResourceCompilerTool"/>
      <Tool Name="VCResourceCompilerTool" PreprocessorDefinitions="_DEBUG"
     AdditionalIncludeDirectories="$(IntDir)" Culture="1041"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="2" GenerateDebugInformation="true"
     ModuleDefinitionFile=".¥VC-2V.def"   AdditionalLibraryDirectories=
      "..¥..¥library¥dll¥debug;c:¥@keikan¥ksim¥bin;..¥..¥VS2005Project;..¥..¥library¥LocalLang¥debug"
     LinkIncremental="2" OutputFile="C:¥@keikan¥ksim¥bin¥VC-2V.dll"
     AdditionalDependencies=
     "LocalLang.lib Sim.lib dll.lib odbc32.lib odbccp32.lib opengl32.lib glu32.lib glaux.lib"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>

    <Configuration Name="Release|Win32" CharacterSet="2" UseOfMFC="2" ConfigurationType="2"
     IntermediateDirectory="$(ConfigurationName)"
     OutputDirectory="$(SolutionDir)$(ConfigurationName)" WholeProgramOptimization="1">
      <Tool Name="VCPreBuildEventTool"/>
      <Tool Name="VCCustomBuildTool"/>
      <Tool Name="VCXMLDataGeneratorTool"/>
      <Tool Name="VCWebServiceProxyGeneratorTool"/>
      <Tool Name="VCMIDLTool" MkTypLibCompatible="false" PreprocessorDefinitions="NDEBUG"/>
      <Tool Name="VCCLCompilerTool"
     PreprocessorDefinitions="WIN32;_WINDOWS;NDEBUG;_USRDLL;RDH_NONE;MULTI_LANG;PLUGIN"
     DebugInformationFormat="3" Detect64BitPortabilityProblems="true" WarningLevel="3"
     UsePrecompiledHeader="0" RuntimeLibrary="2"
     AdditionalIncludeDirectories="..¥..¥library¥import;..¥..¥SIM;
     ..¥..¥SIM¥@merge;..¥..¥..¥include;..¥flow¥C" Optimization="0"/>
```

```
      <Tool Name="VCManagedResourceCompilerTool"/>
      <Tool Name="VCResourceCompilerTool" PreprocessorDefinitions="NDEBUG"
     AdditionalIncludeDirectories="$(IntDir)" Culture="1041"/>
      <Tool Name="VCPreLinkEventTool"/>
      <Tool Name="VCLinkerTool" TargetMachine="1" SubSystem="2" GenerateDebugInformation="true"
     ModuleDefinitionFile=".¥VC-2V.def"   AdditionalLibraryDirectories=
     "..¥..¥library¥dll¥release;c:¥@keikan¥ksim¥bin=;
     ..¥..¥VS2005Project:..¥..¥library¥LocalLang¥release"
     LinkIncremental="1" OutputFile="C:¥@keikan¥ksim¥bin¥VC-2V.dll"
     ddditionalDependencies=
     LocalLang.lib Sim.lib dll.lib odbc32.lib odbccp32.lib opengl32.lib glu32.lib glaux.lib"
     nableCOMDATFolding="2" OptimizeReferences="2"/>
      <Tool Name="VCALinkTool"/>
      <Tool Name="VCManifestTool" EmbedManifest="false"/>
      <Tool Name="VCXDCMakeTool"/>
      <Tool Name="VCBscMakeTool"/>
      <Tool Name="VCFxCopTool"/>
      <Tool Name="VCAppVerifierTool"/>
      <Tool Name="VCWebDeploymentTool"/>
      <Tool Name="VCPostBuildEventTool"/>
    </Configuration>
  </Configurations>
  <References> </References>

  <Files>
    <Filter Name="ソースファイル" UniqueIdentifier="{4FC737F1-C7A5-4376-A066-2A32D752A2FF}"
     Filter="cpp;c;cc;cxx;def;odl;idl;hpj;bat;asm;asmx">
      <File RelativePath=".¥2Vwnd.cpp"> </File>
      <File RelativePath=".¥stdafx.cpp">
        <FileConfiguration Name="Debug|Win32">
          <Tool Name="VCCLCompilerTool" UsePrecompiledHeader="1"/>
        </FileConfiguration>
        <FileConfiguration Name="Release|Win32">
          <Tool Name="VCCLCompilerTool" UsePrecompiledHeader="1"/>
        </FileConfiguration>
      </File>
      <File RelativePath=".¥VC-2V.cpp"> </File>
      <File RelativePath=".¥VC-2V.def"> </File>
      <Filter Name="C">
        <File RelativePath="..¥Flow¥C¥cci_code.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_dml.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_face.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_file.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_misc.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_pars.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_quat.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_sml.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_tbl.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_tkn.c"> </File>
        <File RelativePath="..¥Flow¥C¥cci_vector.c"> </File>
      </Filter>
    </Filter>

    <Filter Name="ヘッダーファイル" UniqueIdentifier="{93995380-89BD-4b04-88EB-625FBE52EBFB}"
     Filter="h;hpp;hxx;hm;inl;inc;xsd">
      <File RelativePath=".¥2Vwnd.H"> </File>
      <File RelativePath="..¥Flow¥C¥cci.h"> </File>
      <File RelativePath="..¥Flow¥C¥cci_dml.h"> </File>
      <File RelativePath="..¥Flow¥C¥cci_file.h"> </File>
      <File RelativePath="..¥Flow¥C¥cci_prot.h"> </File>
      <File RelativePath="..¥Flow¥C¥cci_quat.h"> </File>
      <File RelativePath=".¥stdafx.h"> </File>
      <File RelativePath=".¥VC-2V.h"> </File>
```

```
      <File RelativePath=".¥VC-2V.rc.h"> </File>
    </Filter>

    <Filter Name="リソースファイル" UniqueIdentifier="{67DA6AB6-F800-4c08-8B7A-83BB121AAD01}"
     Filter="rc;ico;cur;bmp;dlg;rc2;rct;bin;rgs;gif;jpg;jpeg;jpe;resx;tiff;tif;png;wav">
      <File RelativePath=".¥res¥ALOWDD.BMP"> </File>
      <File RelativePath=".¥res¥ALOWDU.BMP"> </File>
      <File RelativePath=".¥res¥ALOWUD.BMP"> </File>
      <File RelativePath=".¥res¥ALOWUU.BMP"> </File>
      <File RelativePath=".¥res¥ICON3.ICO"> </File>
      <File RelativePath=".¥res¥SIM.ICO"> </File>
      <File RelativePath=".¥res¥TOOLBAR.BMP"> </File>
      <File RelativePath=".¥VC-2V.rc"> </File>
      <File RelativePath=".¥res¥VC-2V.rc2"> </File>
    </Filter>
    <File RelativePath=".¥ReadMe.txt"> </File>
    <File RelativePath="..¥..¥..¥..¥..¥ksim¥Language¥id¥VC-2V.dll.id.xml"> </File>
    <File RelativePath="..¥..¥..¥..¥..¥ksim¥Language¥ja¥VC-2V.dll.ja.xml"> </File>
  </Files>
  <Globals>
    <Global Name="RESOURCE_FILE" Value="VC-2V.rc"/>
  </Globals>
</VisualStudioProject>
```

⑤リンクするライブラリの選択

C 言語のランタイムライブラリ、および MFC ライブラリのリンクに際して、スタティック・リンクとするか、ダイナミック・リンクとするかが選択でき、前者の場合にはアプリケーションの実行形式の中にこのクラスに相当するプログラムは含められ、後者の場合にはプログラム(exe)がロードされた時点で dll ファイルが別途ダイナミックにロードされる。

MFC を提供する DLL のバージョンは、使用する開発環境に対応している。より古いバージョンの MFC を使用する場合には、プロジェクトのプロパティ設定において明示的に指定する必要がある。VS2005 の場合には、mfc80.dll 等となっており、VS2008 では mfc90.dll となる。

ダイナミック・リンクの場合に、リンクする dll ファイルを、Windows システムに伴ってセットアップされている dll ファイルとするか、あるいは実行形式と同じディレクトリにセットアップされた dll ファイルとするかが選択できる。前者の場合、セキュリティの高い設定がなされたシステムにおいては、Windows システムにセットアップされた dll ファイルを走行させることが許可されない。後者の場合には、テストとデバッグを行った際に使用した dll を指定することができる。

更に、後者の場合で、使用する dll ファイルをセットアップに含め、アプリケーション実行形式と同じディレクトリに置く場合(SxS: Side by Side)、ダイナミック・リンクしている dll を特定するために、マニフェストを添付する。これは実行時に参照する dll を記述する xml 形式のテキストファイルであり、「実行形式名.exe.manifest」という名称のファイル(拡張子「.manifest」)として実行形式に添付される。また設定によりこの内容を実行形式に埋め込むこともできる（マニフェスト・ツールの入力と出力の設定)。

使用する DLL ファイルは、<dependency>タグの中に記述されている。DLL を使用しな

いスタティック・リンクのコンソールアプリの場合には、このタグは存在しない。

実行形式が起動する時に、同じディレクトリにこのタグで指定した DLL ファイルが存在する場合には、DLL として同時にロードされ、諸関数が呼び出される。

リスト 4-5-4 マニフェストファイル

①コンソールアプリ vc-1c.exe のマニフェスト vc-1c.exe.manifest

```
<?xml version=" 1.0"  encoding=" UTF-8"  standalone=" yes" ?>
<assembly xmlns=" urn:schemas-microsoft-com:asm.v1"  manifestVersion=" 1.0" >
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type=" win32"  name=" Microsoft.VC80.CRT"  version=" 8.0.50608.0"
processorArchitecture=" x86"  publicKeyToken=" 1fc8b3b9a1e18e3b" ></assemblyIdentity>
    </dependentAssembly>
  </dependency>
</assembly>
```

②Windows アプリ vc-1c_D.exe のマニフェスト vc-1c_D.exe.manifest

```
<?xml version=" 1.0"  encoding=" UTF-8"  standalone=" yes" ?>
<assembly xmlns=" urn:schemas-microsoft-com:asm.v1"  manifestVersion=" 1.0" >
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type=" win32"  name=" Microsoft.VC80.CRT"  version=" 8.0.50608.0"
processorArchitecture=" x86"  publicKeyToken=" 1fc8b3b9a1e18e3b" ></assemblyIdentity>
    </dependentAssembly>
  </dependency>
  <dependency>
    <dependentAssembly>
      <assemblyIdentity type=" win32"  name=" Microsoft.VC80.MFC"  version=" 8.0.50608.0"
processorArchitecture=" x86"  publicKeyToken=" 1fc8b3b9a1e18e3b" ></assemblyIdentity>
    </dependentAssembly>
  </dependency>
</assembly>
```

③上記①②が参照する CRT(C ランタイム)のマニフェスト Microsoft.VC80.CRT.manifest

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright © 1981-2001 Microsoft Corporation -->
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <noInheritable/>
    <assemblyIdentity
        type="win32"
        name="Microsoft.VC80.CRT"
        version="8.0.50608.0"
        processorArchitecture="x86"
        publicKeyToken="1fc8b3b9a1e18e3b"
    />
    <file name="msvcr80.dll"/>
    <file name="msvcp80.dll"/>
    <file name="msvcm80.dll"/>
</assembly>
```

④上記②のみが参照する MFC のマニフェスト Microsoft.VC80.MFC.manifest

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Copyright © 1981-2001 Microsoft Corporation -->
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <noInheritable/>
    <assemblyIdentity
        type="win32"
        name="Microsoft.VC80.MFC"
        version="8.0.50608.0"
        processorArchitecture="x86"
        publicKeyToken="1fc8b3b9a1e18e3b"
```

```
    />
    <file name="mfc80.dll"/>
    <file name="mfc80u.dll"/>
    <file name="mfcm80.dll"/>
    <file name="mfcm80u.dll"/>
</assembly>
```

⑥マルチスレッド

スタティック・リンクを用いる場合リンクするライブラリを、シングルタスク(LIBC)か、マルチタスク(LIBCMT)か選択することができる。ダイナミック・リンクを用いる場合には、マルチタスクのみとなる。

スレッドとは、概ね関数呼び出しにおけるリターン・スタックを介した呼び出しの連鎖である。どこかのレベルで入力待ちなどの時間を要する処理が発生した場合には、そのスレッドは終了しない。時間がかかる処理を別スレッドとして分離することにより、その他の処理を進めることができる。

利活用処理系の例の一つである VC-2V において、大きなデータを処理の処理状況をプログレス・インジケータとして正しく表示するために、メタファイルのコンパイルとデータファイルの処理を、独立したスレッドを起動して実行している。

まず、独立したスレッドとする MyThreadFunc を定義し、その中でメタファイルのコンパイルとデータの読み込みを行う。

リスト 4-5-5 スレッド関数の定義(2VWnd.cpp)

```
DWORD WINAPI MyThreadFunc(LPVOID lpParam){ ・・・　（この中でコンパイルと読み込み処理を行う）
```

次に、スレッドを開始する関数を用意する。

リスト 4-5-6 スレッド関数の起動関数(2VWnd.cpp)

```
void CreateAndStartThread(){
     g_iCount = 0;
     g_bRunning = true;
     hThread = CreateThread(NULL,0,MyThreadFunc,NULL,0,&ThreadId);
}
```

メタファイルとデータファイルの指定や処理開始を指示するダイアログにおいて、開始ボタンが押された場合のコールバックにおいて、諸条件が整っていることを確認の後、最後にこのスレッド開始関数を呼び出す(リス 4-5-7)。これにより、処理はコールバック関数から抜けて、新たなメッセージを受け付ける状態になる。一方、起動されたスレッドにより読み込み処理が行われる。、スレッド側からはコンパイルが終わりデータ読み込みに入ってから、このハンドルを用いて進捗状況を表示している。

リスト 4-5-7 スレッド関数の起動関数の呼び出し(2VWnd.cpp)

```
void C2vWnd::OnBnClickedOk(){
     ・・・・（諸条件の確認等）・・・・
     hwnd = m_hWnd;//マルチスレッド用(hwndは大域変数)
     CreateAndStartThread();
}
```

別スレッドとして開始された読み込み処理の中で、プログレス・インジケータ表示のためのデータ（進捗率）が計算され、上記の操作画面に対してメッセージを送付する。する

と、操作画面はこのメッセージを受けて、読み込み処理継続中も進捗率の表示を行う。なおこの例では、開始ボタンによりデータ読み込み（ダウンロード）のスレッドを起動する際に､大域変数 hwnd にこのウィンドウのハンドルを渡して進捗率の表示先を伝えている。データ読み込みを、開始ボタンと同一のスレッドの中で実行すると、全ての読み込みが終わるまで開始ボタンのコールバックが終了しないため、進捗状況を反映した画面更新要求はメッセージポンプの中に保留されてしまい、ユーザに届かない。

更に、最近のマシン・OS のように CPU コアを複数有する場合には、マルチスレッドとすることにより、並進処理を行うことができるが、上記に例示した処理系ではそのような処理は行っていない。

⑦メモリリークの検査と除去の方法

VisualStudio においては、プロセスが終了した時点で、未解放のメモリブロックのリストを出力するデバッグ機能が利用できる。しかし、これらのメモリブロックの生成場所が特定できなければ、プログラムの修正の能率は上がらない。

景観シミュレータのための外部関数として作成した LandXML.exe のビルドとデバッグの際に導入した、メモリリーク検出機構を、cci_misc.c に応用した。

デバッグモードにおいて、メモリブロックリストを作成し、Free で削除する。このリストには、メモリブロックのアドレスと削除フラグが記録されている。

終了時に、削除されていない残余のメモリブロックの最初のもののアドレス（出現順位）をデバッグ用のファイルに出力する。

次回処理対象とするファイル等を同じ条件として起動した時に、ファイルからロードした記録上の残余の先頭のアドレスに対応する ALLOC 命令が実行された時点でブレークを掛ける。この命令によるメモリブロック取得命令が、メモリリークの原因個所として特定される。

cci_face で、d3Malloc 等を使用しているが、仮想コンバータのビルドと動作環境が利活用形態によってはダイナミック・リンクにより sim.exe と連動するとは限らないので、ここでのメモリブロックリストは、d3dml ライブラリのものではなく、cci_misc の中に独立して作成するものを使用した。

（５）Android 用アプリケーションの実行環境

①CPU

Android は、2015 年現在では携帯端末（スマートフォン、タブレット）の OS として広く用いられており、ARM アーキテクチャの CPU が用いられている。携帯端末の各種 CPU の機械語の上で DALVIC 仮想マシンのエミュレータが実行されており、この仮想マシンの機械語として、Android　OS やアプリケーションソフトが実行されている。

②OS と記憶構成

Android OS のカーネル部分は、LINUX である。しかし、様々なアプリケーションがプレ

インストールされ直ちに使用可能な状態で市販された携帯端末の液晶表示画面に現れる表示は、背景の前に各種アプリケーションのアイコン（縮小画像）が表示されている。

ハードディスク装置の代わりに、SD カードが装着されており、各種アプリケーションやデータは不揮発性のメモリに格納されている。SD カードのパスは、/mnt/sdcard 等となっている。

記憶装置に格納されているディレクトリやファイルを一覧するための、「ファイルマネージャ」のようなアプリケーションや、コマンドラインでファイル一覧を表示するようなシェルは、標準ではセットアップされておらず、入手してセットアップする必要がある。

携帯端末に、マイクロ SD カードを装着するスロットが存在する場合には、そこに SD カードを装着することにより、ファイルマネージャでファイルを一覧し、コピーすることができる。更に、セットアップ用ファイル（拡張子「apk」をタップして起動することができる。この方法により、開発したアプリケーションやデータをセットアップすることができる。

USB インターフェース経由で携帯端末を PC に接続することにより、PC の側から見てあたかも外付けの外部記憶装置のように端末の内部のファイルを一覧することができる。この接続方法により、セットアップに必要なファイルを携帯端末の中の記憶装置にコピーすることができる。

③各種センサ

携帯端末には、背面カメラ、GPS センサ、加速度センサ、磁気センサが備えられており、API 関数を通じてその値を取得することができる。また、計測値が変化した場合にのみ、コールバックが呼び出されるように初期化することもできる。

④OpenGL ES

二次元、三次元の描画処理を行うライブラリであり、PC 上で利用可能な OpenGL（Windows の場合には、実体は OpenGL32.dll 等）を簡略化した「組込用」のバージョンである（Embedded Edition）。四以上の頂点を持つ多角形（ポリゴン）を直接描画することができないため、バッファリングを行っている。頂点の法線ベクトル、頂点カラー、テクスチャ座標についても、これに対応して３頂点を単位とするバッファ（配列）に格納して描画コマンドを発行する。

⑤アプリの状態遷移

ホーム画面からアプリのアイコンをタップすることで VC-3M アプリが起動し、メイン画面が表示される。この過程で、onCreate(), onStart(), onResume()の３のメソッドが呼び出される。メイン画面上のボタンのタップにより例えばリアルタイム表示が選択されると、モデル選択画面のアクティビティが起動するため、メイン画面は裏に隠れる。この時、onPause()メソッドが呼び出される。メイン画面で「終了」ボタンがタップされると、onPause(), onStop()、onDestroy()を経てアプリは終了する。

リアルタイム表示が終了すると、メイン画面は再び表示され、onResume()メソッドが呼

び出される。

メイン画面が開いた状態で、ホームボタンがタップされホーム画面に移った場合も onPause(), onStop() が呼び出される。

これらの関数は、プログラムの中から呼び出されるのではなく、ユーザによる操作やイベントの発生（例えば電話がかかってくる）に伴い呼び出される。

実機の上では、他のアプリやプロセスが稼働しており、画面表示の上で相互に影響しあう。アプリがホーム画面や他のアプリの裏面に隠れた状態が長く続き、メモリなどの資源が不足すると、OS によりプロセスが終了させられる。別のアプリである「設定」アプリを用いて、実行中の VC-3M を強制終了した場合にも VC-3M のプロセスは終了し、全てのアクティビティは破棄される。従って、起動過程よりも終了過程の方が多様で複雑である。メモリや OpenGL やログファイルのハンドルなどのリソースの解放を適切にプログラムしなければアプリケーションは完成しない。アプリケーションの側では一貫性のある対処を行う必要がある。各画面に対応した特殊な処理が必要な場合には、Activity クラスから各関数をオーバーライドした関数を作成し、その中で処理を記述する。特別な処理が不要で、特に関数を記述しなければ、Activity クラスの中の各処理が実行される。

例えば、VC-3M のリアルタイム表示は、OpenGL による描画や各種センサ値を用いたこの処理系で最も複雑な RealtimeActivity クラスにおいて処理されている。onCreate でコンパイラによるメタファイルを解釈・実行してデータファイルから表示に使用するモデルをロードし、onDestroy でモデルをアンロードしている。onResume において、各種センサの起動と OpenGL 画面の初期化を行い、onPause においてそれらの終了処理を行っている。onStart、onStop、onRestart の状態遷移に関しては特別な処理は行っていない。
onStart の後は、直ちに onResume に進み、前面にアプリが表示される。onResume に進まずに onStop となる場合はない。

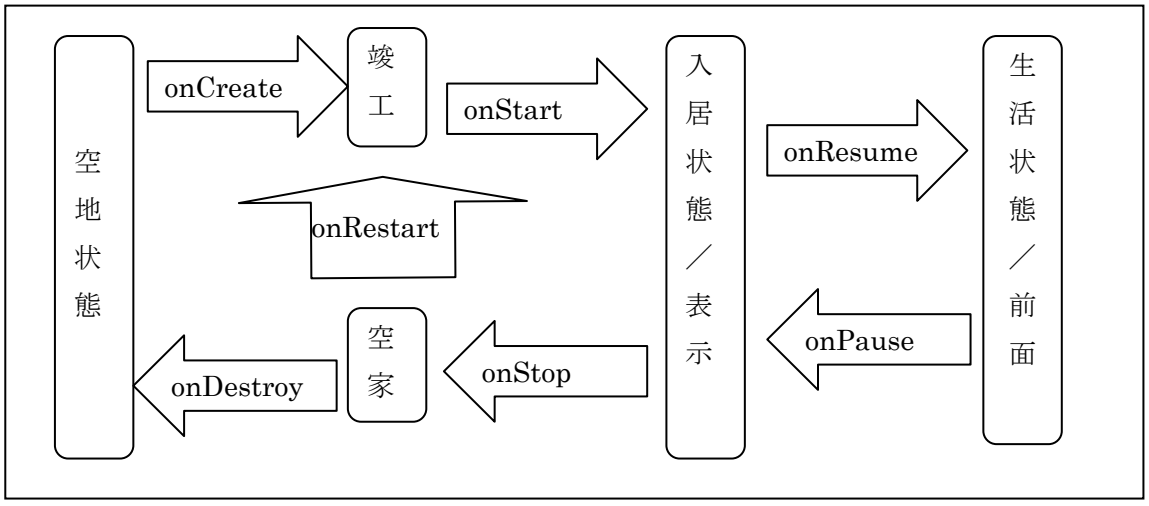


図 4-5-2 アプリの状態遷移図（各関数の役割を示すため、状態を住宅に喩えて表現した）

（６）Android 用アプリケーションのクロス開発環境

この開発環境全体は、Windows2003Server（デスクトップ型)、Windows Vista/x86（ノート型）及び Windows 8.1/x64（ノート型）の上で動作させ、問題なく利用でき、また異なるマシン上での成果を移植して引き続き作業できることを確認した。その際に用いた方法と、いくつかのわかりにくい障害について記録しておく。

① 開発に用いるホストマシン

プログラムの開発を、最終的に利用するコンピュータ（ターゲットマシン）の上で、あるいは同じ CPU や OS のマシンの上で開発することは一般的であり、開発環境はシンプルなものでよい。一方、携帯端末、モバイル機器、あるいは自動車、家電製品、エレベータ等に組み込まれて動作するプログラムの開発を行う場合には、開発に使用するマシンと同様のテスト環境を、開発に使用するマシンの上で模擬的に実現し、実機でのテストの前までのプログラミング作業を安全快適な執務環境の中で実施することが広く行われ、クロス開発環境と呼ばれている。本研究においては、VC-3M がこれに該当する。

Android を OS とする携帯端末自体を開発（コンパイル・ビルド）に用いることは現段階ではまだ一般的ではなく、PC の OS（Windows 等）の上で動作するエディタ、コンパイラ、リンカなどから構成されるクロス開発環境を用いて、異なる OS で走行しているターゲット・マシンのためのアプリケーションを開発することがまだ一般的である。本研究においては、Windows8 を OS とする、ノート型の PC の上に、クロス開発環境を構築してコーディング、テスト、デバッグを行った。これにより、野外の現場でのテスト結果を踏まえ、宿などの安全な場所でプログラム修正などを行うこともできた。

②SDK(System Development Kit)

Java 言語で記述されたソースコードをコンパイルし、仮想マシンのための機械語（中間コード）を生成するコンパイラが SDK である。Windows 上で動作するクロスコンパイラとして無償提供されているものを、ダウンロードして使用した。SDK だけを用いて、Android アプリケーションを作成することができる。実際には Java 言語のソースコードのコンパイルは、Eclipse 統合開発環境をセットアップし、SDK のコンパイラ等を起動する作業環境が広く用いられている。統合開発環境に付属したエディタで修正したソースコードを保存する操作により、自動的にコンパイラ、リンカが実行され、実行形式を生成する。

Windows マシンにおいては、Program Files/Java 配下にセットアップされている。

③Eclipse 統合開発環境

Java ソースコードの編集とコンパイルを実行する Windows 上の作業環境である。WEB サイトから無償でダウンロードし利用することができる。基本的な特徴として、Windows のレジストリを使用せずに、xml 形式の設定ファイルに各種設定を保存している。Android 開発のためのプラグイン(ADT: Android Development Tools)が、セットアップされている。

統合開発環境の実行形式は、セットアップ先のディレクトリ配下の eclipsec.exe から起動する。最初に cmd.exe が起動し、exlipec.exe1 が実行されると、ワークスペースの選択

画面が開く。

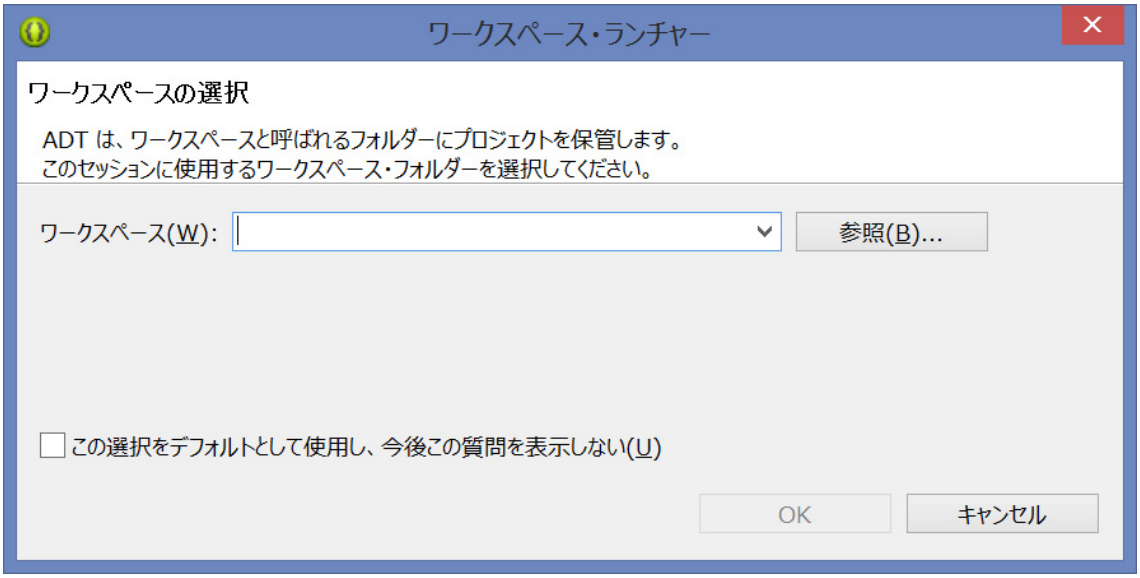


図 4-5-3 ワークスペース選択画面

ここで、現在開発中の各種ファイルを格納しているディレクトリをワークスペースとして指定すると、統合開発環境のメイン画面が開く。

プルダウンメニューの[ウィンドウ]の下に、[パースペクティブ]の項目があり、作業目的により表示する画面の構成を選択することができる。プログラム修正においては、「階層ビュー」を選択する。他に、[C/C++]、[Java]、[DDMS]、[デバッグ]、[リソース]などのビューが選択できる。

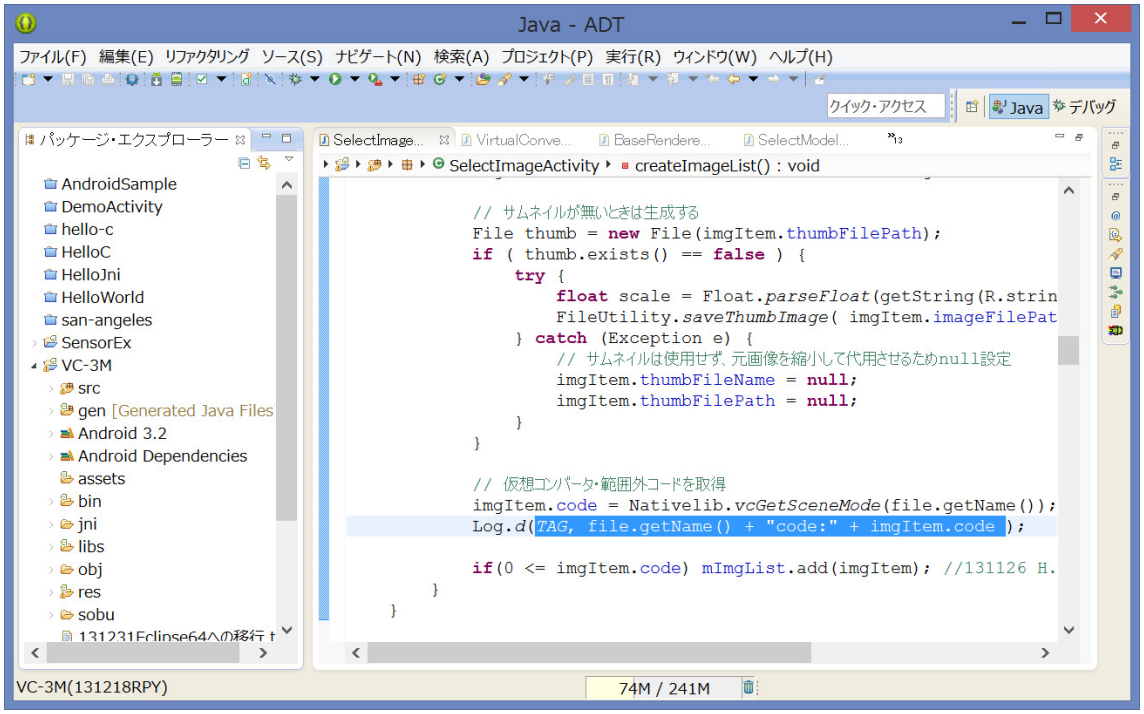


図 4-5-4 Eclipse の統合開発環境の操作画面（階層ビュー）

エディタを用いて JAVA ソースコードを編集し、SDK を用いてコンパイル、リンクすることができる。

更に生成した実行形式のセットアップ（拡張子「.apk」）を、Android 端末のエミュレータ AVD に直ちにロードし、実行テストすることができる。

④NDK(Native Development Kit)

C 言語で記述されたソースコードをコンパイルし、ARM 系 CPU のための機械語(ネイティブコード)を生成するコンパイラが NDK である。ARM 系以外の CPU を用いた Android マシン上では動作しない。

VC-3M の場合には、高速処理が必要となる描画処理の部分を、景観シミュレーションのために C 言語で開発されたライブラリ関数 g3drl.c のソースコードを活用して、NDK を用いて、DLL（ダイナミック・リンク・ライブラリ、Android の場合、拡張子は「.so」）を作成している。アプリケーションにおいては、Java で記述されたフレームからこの DLL をロードし、ユーザの操作や各種センサの計測値も Java 言語のプログラムで取得してこの DLL に渡し、画面表示などの高速処理を実行している。

NDK は WEB からダウンロードしてセットアップすることができる。Windows においては、NDK のセットアップ先の直下にある ndk-build.cmd というバッチファイルを実行することにより、/prebuilt/windows/bin/make.exe に引数として build-local.mk というメイクファイルが渡され、コンパイル、リンクが実行される。

なお、拡張子 cmd は WindowsNT 系のバッチコマンドの拡張子で、MS-DOS 系のバッチコマンドの拡張子 bat とほぼ同様の意味であり、bat の拡張子のバッチコマンドは Windows2000、XP 以降 NT 系に統一された Windows のバージョンのマシンでも動作する。MS-DOS においては、新たな周辺機器やアプリケーションの増設等に際してシステムの初期化段階で実行される autoexec.bat を編集することがユーザにより行われていたが、Windows 以降はユーザの目に触れる機会が無くなった。バッチファイルは、コマンドラインによる様々な設定や実行形式の起動を指示するインタープリタのためのスクリプト言語によるプログラムであり、変数の入力と参照(%変数名%)、整数型の数式演算、条件分岐(if,else, goto :ラベル名)、サブルーチン呼び出し(call)、並行処理する別プロセスの起動(start)などを行うことができ、現在でもシステム管理のために広く用いられている。各コマンドの解説は、コンソール画面(cmd.exe)で、

```
コマンド /?
```

とタイプすることにより参照することができる。

VC-3M の場合には、java ソースコードを含めたビルド一式を格納したディレクトリ VC-3M の配下の jni の配下に C 言語で書かれた NDK のソースコードを配置すると共に Android.mk というメイクファイルにソースコードを列挙し、コマンドラインにてここに移動して、

```
ndk-build
```

とタイプすることにより、パスが通っている NDK ディレクトリの上記の ndk-build.cmd が

実行され、現在のディレクトリにある定義ファイル Android.mk の指示に従ってビルドが実行され、VC-3M/libs/armeabi の下にアプリケーションがロードすることができる ARM プロセッサのネイティブコードによるダイナミック・リンク・ライブラリである、libVirtualConverter.so が作成される。

ndk-build.cmd の最後の行で、メイクファイル（拡張子.mk）を解釈し実行する make.exe が呼び出され、引数として build-local.mk というメイクファイルが実行されている。その中では、コンパイル、リンクのための処理プログラム群（ツール・チェイン）を定義した個別のメイクファイル（init.mk、add-application.mk、setup-imports.mk、setup-app.mk および build-all.mk）が呼び出され、本ビルド固有の Android.mk の中で指定されたソースコードのコンパイル、リンクが実行される。本処理系の場合には、Android.mk のみを編集し、その他のメイクファイル等はセットアップ状態のまま使用してビルドを行った。

リスト 4-5-8 ndk-build.cmd の内容

```
@echo off  //コマンドをコンソールに出力しない
rem This is a Windows cmd.exe script used to invoke the NDK-specific GNU Make executable  //コメント
set NDK_ROOT=%~dp0 //このバッチのディレクトリを設定（オプション：「“」を除く、ドライブ文字だけ、パスだけ）
set NDK_MAKE=%NDK_ROOT%/prebuilt/windows/bin/make.exe  //NDK_ROOT配下の実行形式を設定
%NDK_ROOT%¥prebuilt¥windows¥bin¥make.exe -f %NDK_ROOT%build/core/build-local.mk %* || exit /b %ERRORLEVEL%
//make.exe に build-local.mk を実行させる。 %*で、コマンドラインの引数も make.exe に受け渡している。
//エラー情報をリターンしているので、このバッチファイルを下請実行させる上位のバッチファイルを組むことも可能である。
```

注）//以下は解説のために付した注記である

Android.mk は、改行で区切られたシンプルなテキストファイルであり、VS2005 における vcproj ファイルに似た役割を担っている。この中の LOCAL_SRC_FILES 変数にコンパイルすべきソースコード群が「¥」区切りでリストされている。

リスト 4-5-9 仮想コンバータのための Android.mk の内容

```
# Copyright (C) 2009 The Android Open Source Project
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_C_INCLUDES := $(LOCAL_PATH)/mobile

LOCAL_MODULE    := VirtualConverter
LOCAL_SRC_FILES := ¥
 mobile/cci.c¥
 mobile/cci_code.c¥
 mobile/cci_dml.c¥
```

```
 mobile/cci_file.c¥
 mobile/cci_pars.c¥
 mobile/cci_misc.c¥
 mobile/cci_quat.c¥
 mobile/cci_sml.c¥
 mobile/cci_tbl.c¥
 mobile/cci_tkn.c¥
 mobile/cci_face.c¥
 mobile/D3dml.c¥
 mobile/D3malloc.c¥
 mobile/D3mat.c¥
 mobile/D3pick.c¥
 mobile/Dbms.c¥
 mobile/S3sml.c¥
 mobile/S3set.c¥
 mobile/S3get.c¥
 mobile/S3delete.c¥
 mobile/dummy.c¥
 mobile/G3drl.c¥
 mobile/G3load.c¥
 mobile/G3font.c¥
 mobile/G3ground.c¥
 mobile/StereoSight.c¥
 mobile/ScnFile.c¥
 mobile/mobile.c¥
 VirtualConverter.c¥
 mobile/image.c¥
 IMGSGI/Open.c¥
 IMGSGI/Close.c¥
 IMGSGI/FILBUF.c¥
 IMGSGI/FLSBUF.c¥
 IMGSGI/NAME.c¥
 IMGSGI/PIX.c¥
 IMGSGI/RDWR.c¥
 IMGSGI/Rle.c¥
 IMGSGI/Row.c¥

LOCAL_MODULE_FILENAME := libVirtualConverter

LOCAL_LDLIBS := -lGLESv1_CM -ldl -llog

include $(BUILD_SHARED_LIBRARY)
```

ソースコードリストの内､cci から始まるファイルは､仮想コンバータ基幹部分を構成する。d3dml.c から g3ground.c までは、景観シミュレーションシステムのライブラリ関数のソースコードを再利用した、メモリ上の三次元データ構造体の管理や表示処理を行うプログラムである。Scnfile.c と mobile.c と VirtualConverter.c は、Java 言語で書かれた本処理系(VC-3M)の全体フレームから NDK により開発した処理呼び出すためのインターフェースである。IMGSGI 配下は、SGI 形式の画像ファイルをテクスチャとして読み込み、表示に使用するためのプログラムである。LOCAL_MODULE_FILENAME は、作成するダイナミック・リンク・ライブラリの名称（拡張子.so）である。LOCAL_LDLIBS は、OpenGL ES1.0 を使用することを宣言している。

なお、ndk-build には、スイッチを付けることができ、概略以下のようになっている。全体は、「-?」のスイッチでコンソールに表示される。

リスト 4-5-10 ndk-build.cmd コマンドに付加する主なスイッチ

```
-? –h –help ヘルプ（スイッチの一覧）
-d 豊富なデバッグ情報
-f メイクファイルの指定
-i エラーの無視
-k ビルドできない対象があっても継続
-n ビルドを実行せず、コマンドのみ出力
-s サイレント
-v バージョン表示
-w 未定義変数の警告
```

エラー無くコンパイル・ビルドが完了すると、LOCAL_MODULE_FILENAME で指定された libVirtualConverter.so（拡張子 so はライブラリを示す）が生成する。次のステップで、Java 言語による実行形式全体のビルドにおいて、このダイナミック・リンク・ライブラリを含むセットアップ(拡張子 apk)が作成される。

Eclipse 上で編集する Java プログラムの修正を行った場合には、修正結果を保存する操作により、自動的にビルドが行われ、実行形式（セットアップファイル）が更新される。しかし、ダイナミック・リンク・ライブラリを修正しても、全体のリビルドは自動的には行われない。そこで、C 言語で NDK のソースを修正した場合には、NDK での再ビルドを行った後、Java 側のリソースファイルに含まれているリテラル文字列を格納した strings.xml の中の app_name を修正し保存することで、全体の再ビルドを起動する方法を採った。

⑤プロジェクトのディレクトリ構成

WorkSpace として、Eclipse 起動時に指定するディレクトリの配下に、プロジェクト毎のサブディレクトリが置かれ、その中に関連するソースコード等が整理される。VC-3M の場合には､VC-3M というサブディレクトリの下に､以下のサブディレクトリを置いている。

/bin　セットアップ(拡張子.apk)を生成する。

/gen　R.java

/jni　ネイティブコードを NDK で生成するための C 言語のソースコード群

/libs　NDK でビルドしたダイナミック・リンク・ライブラリ(libVirtualConverter.so)

/obj　NDK でコンパイルした結果のオブジェクト群(C ソース名.o)

/res　リソースファイル群（アプリの起動用アイコン、layout 配下の各画面のウィジェット構成(xml)、values 配下のリテラル文字列(xml)）

/src　クラス毎の定義や動作を記述した Java 言語のソースコード群

Java 言語で書かれたソースコードは、src 配下にディレクトリでグループ分けして格納してある。

リスト 4-5-11 VC-3M アプリを構成するクラス一覧

```
クラス名と概略機能　　クラス名(日付)　機能(画面構成リソース名)

(root)
Nativelib(140108)
VirtualConverterActivity(140107) メイン(R.layout.main)

(adapter)
```

GpsValue(140107,1)GPS センサ値の構造
ImageItem(140107,1)画像リストの構造
InfoFileAccessor(140107,4)ModelIndex.txt の読み込み
ListItem(140107,2)モデル一覧の構造
OrientationValue(140108,1)方位センサ値の構造(131227:RPY->Quat)
SelectImageAdapter(140107,5)再生表示用縮小画像表示(R.layout.image_list)
SelectModelAdapter(140107,2)モデル選択用リスト表示(R.layout.model_list)

(base)
BaseRenderer(140107,3)リアルタイム、再生表示の画面状態
（初期、描画可能、描画、描画終了(MTerm)、それ以外は待機ループ）

(realtime)
ModelRenderer(140729,5)モデルの描画
Quat(140107,7)Quat 算術
RealtimeActivity(141003,41)リアルタイム(R.layout.realtime)

(replay)
ReplayActivity(140912,9)再生表示(R.layout.replay)
SceneRenderer(140107,3)シーン(再生画面)の描画
SelectImageActivity(140107,11)画像選択(R.layout.selectimage)

(selectmodel)
SelectModelActivity(140107,8)モデル選択「見たい場所」(R.layout.selectmodel)

(util)
DateUtility(140107,1)
DialogUtility(140107,3)
FileUtility(140107,6)
VirtualConverterErrorDialog(140107,3)

新たな Windows マシンの上に開発環境を移植する場合、プロジェクト毎のディレクトリは、workspace ディレクトリ配下のディレクトリにコピーされただけでは、処理対象とすることはできない。workspace/.metadata/.pugins/.org.eclipse.core.resources/.projects フォルダの配下に、開発環境に登録されているプロジェクト単位のサブフォルダがある。このサブフォルダ群が統合開発環境のプロジェクト一覧に表示される。

その一覧の中の VC-3M サブフォルダには、.indexes、org.ecipse.jdt.cor 等のフォルダと並んで、.location（拡張子なし）という名称の、utf-8 形式のテキストファイルが存在し、実際のフォルダの場所を示す URI が「URI//file:c:/・・・」の書式で記述されている。

新たにセットアップした開発環境に、従前の開発環境から一つのプロジェクトを移植するためには、既存のプロジェクトをインポートする操作として、[ファイル][インポート][既存のプロジェクト]で従前の workspace フォルダを参照し、その配下にあるプロジェクトの一覧を表示する。「プロジェクトをワークスペースにコピー」というチェックを入れることにより、従前のプロジェクトを保持したまま、新たな環境にプロジェクトを移植できる。

⑥Android 端末エミュレータ (AVD=Android Virtual Device)

Windows 等の PC 上で、Android 端末の画面を一つのウィンドウとして表示し、その動作を模擬的に実行(エミュレーション)する。更に、デバッガ DDMS により、端末内部でのプログラムの実行状況をデバッグすることができる。

Eclipse 統合開発環境の上にセットアップし、作成した実行形式を直ちにロードして実行することができる。但し、携帯端末の位置や姿勢を変化させてリアルタイムで表示に反映

させるようなテストには使用できない。しかし、現場でのテストに移行する前の、コンパイラの動作の確認や、画面遷移の確認をラボで行える価値は高い。

Eclipse の Java パースペクティブのビューにおけるメニュー「ウィンドウ」の下にある「仮想デバイス」を選択することにより、Android 仮想デバイス・マネージャの編集画面が開く。ここで、端末エミュレータの CPU 種別、プラットフォーム(AndroidOS のバージョン)、画面解像度、内部 SD カードの容量などを設定し、端末名を付けて登録することができる。

[AVD 詳細ボタン]で表示されるフォルダ（例えば、users/ユーザ名/.android/avd/端末名.avd)に config.ini などの各種設定ファイルが保存されている。このフォルダのパスは、このエミュレータの設定ファイル「64bit44.ini」が指定されている。セットアップ先/.android/avd/端末名.ini ファイルに、このディレクトリが絶対パスおよび相対パスで指定されている。Eclipse のバージョンによっては、このパスの途中に日本語のフォルダ名が存在すると、仮想デバイスが起動できない場合があった。

図 4-5-5 WindowsPC 上での、Android 端末エミュレータ

詳細はヘルプなどに譲るが、仮想デバイスの設定と起動の作業の概要を解説するとメニューの[実行]からプルダウンした[実行構成]で開く編集画面において、[ターゲット]タブで実行に使用する端末エミュレータを選択する。この画面のマネージャボタンからも上記の「仮想デバイス・マネージャ」画面が開く。

次に統合開発環境から[実行]を選択すると、端末エミュレータ(セットアップ先/sdk/tools/emulator-arm.exe)が起動する。デバッガ DDMS を用いてデバッグする場合には、同様にメニューの[実行]の下の[デバッグ構成]で端末を選択する。

PC の画面に表示された仮想的な携帯端末の内部の記憶装置（SD カード）は、一種の仮想ディスクとして実装されており、Windows のファイルシステムでは一つの長いバイナリファイルである。上記の例におけるファイルは、

セットアップ先/sdk/system-images/android-19/armbeae-v7a/userdata.img （200MB）

である。この中に仮想化されているディレクトリやファイルは、Windows のファイルマネージャ（エクスプローラやコマンドプロンプト）から直接編集・操作することはできない。統合開発環境のプルダウンメニュー[パースペクティブ]から DDMS ビューを選択し、[ファイル]のタブを開くと、端末内部のファイル構成がツリー表示される。対象とするフォルダを選択した上で、操作画面右上にある[Pull a file from device]アイコンをクリックし、ファイルを取り出して名前を付け、Windows のファイルシステム上に保存することができる。また、逆に[Push a file onto device]アイコンをクリックし、Windows 上のファイルを端末に送り込むことができる。

エミュレータ上の端末において、SD カードは、/storage/sdcard にある。この直下に、[＋]アイコン（新規フォルダ作成）をクリックして VirtualConverter ディレクトリを作成し、その下に各種ファイルを送り込むことにより、実行環境を実現する。

走行させるアプリ―ションのセットアップは、統合開発環境における「実行」指示により自動的に行われる。

このエミュレータの起動には、数十秒を要するため、起動したままの状態でデータファイルやメタファイルの修正、あるいは処理系自体のソースコードの修正と再ビルドを行い、再実行する方法が能率的である。

⑦APK ファイルの内部構造

クロス開発環境と、それが最終的に生成するセットアップファイル（拡張子 APK）は、自動的に開発環境が生成するものを利用する限り、ブラックボックスであって構わない。但し、異なるデータファイルやメタファイルを用いて作成した居住環境の記録を WEB 配信などにより配布する際には、著作権の表示などが問題となる可能性があるため、簡単に触れておく。

APK ファイルは、ZIP 形式で圧縮されたファイルに.APK の拡張子を付したものである。よって、拡張子を.zip に変更した上で、Windows 上のエクスプローラなどを用いて内部を参照することができる。

リスト 4-5-12 VC-3M.apk の内容

```
lib
 armeabi
  libVertualConverter.so
META-INF
 CERT.RSA
 CERT.SF
 MANIFEST.MF
res
-drawable-hdpi
--ic_launcher.png（アイコンの画像ファイル）
-drawable-ldpi
--ic_launcher.png（アイコンの画像ファイル）
-drawable-mdpi
--ic_launcher.png（アイコンの画像ファイル）
-layout
--image_list.xml（画像選択画面レイアウトを記述したリソース）
--main.xml（起動画面レイアウトを記述したリソース）
```

--model_list.xml（モデル一覧画面レイアウトを記述したリソース）
--realtime.xml（リアルタイム画面レイアウトを記述したリソース）
--replay.xml（記録生成画面レイアウトを記述したリソース）
--selectimage.xml（画像選択画面レイアウトを記述したリソース）
--selectmodel.xml（モデル選択画面レイアウトを記述したリソース）
AndroidManifest.xml（プログラムのマニフェスト）
classes.dex(dalvic 仮想マシン上の実行形式、逆コンパイル可能)
resources.arsc（文字列リテラルなどのリソース）

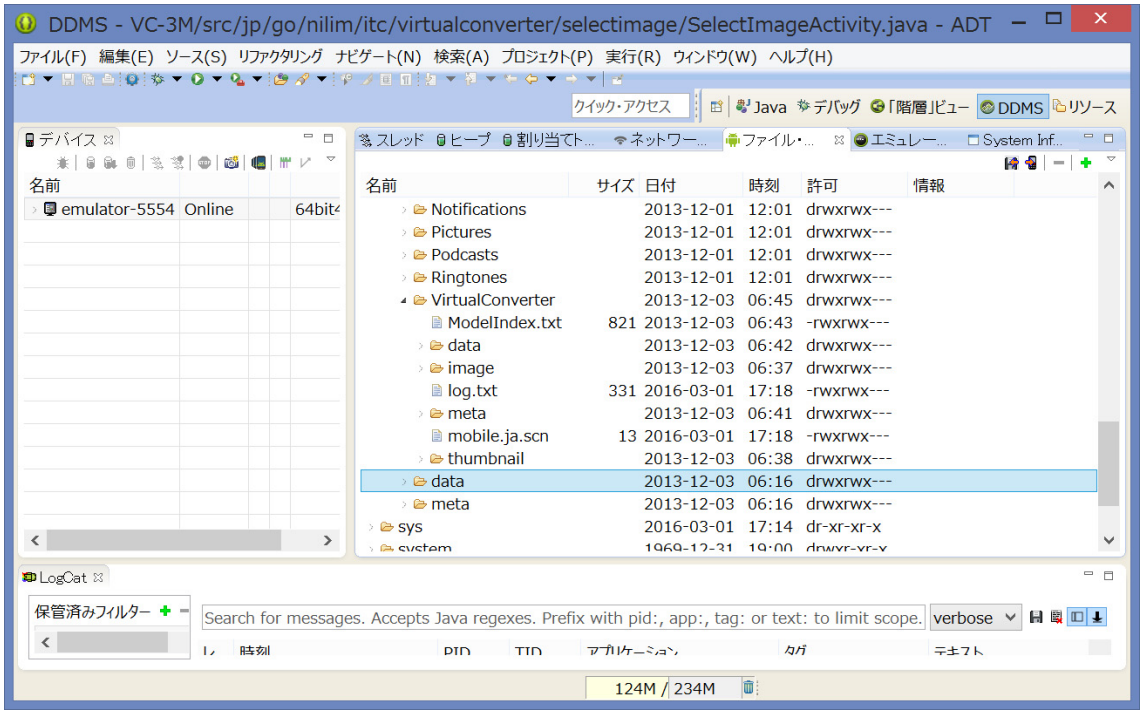


図 4-5-6 端末エミュレータ内部のファイル構成を DDMS で見る

⑧ 開発環境のテストのためのサンプルプログラム

異なる開発環境にプログラムを移植するにあたり、最終的に移植するプログラム自体を使用すると、問題の切り分けが複雑となる。そこで、いくつかのシンプルな、開発環境テスト用のプログラムを用意することにより、環境変化に関する問題を切り分けることが容易となる。Android 開発環境のセットアップに同梱して提供されている、以下のサンプルプログラムが、新たなマシン上に移植した開発環境の動作確認と障害の原因の特定のために有効であった。

1) HelloWorld

Java のみで、画面に”Hello World”と表示する。

2) HelloJni

“Hello World”と表示する C 言語のダイナミック・リンク・ライブラリを起動する。

3) San-angeles

C 言語のダイナミック・リンク・ライブラリで OpenGL を用いた表示を行うアプリ

4) SensorEx

携帯端末の各種センサの計測値を文字表示するテスト用アプリケーション（Java のみ）。

⑨ 文字コードとデバッグ用メッセージ

Windows 用アプリケーションの開発環境である VS2005 においては、たとえソースコードの日本語文字列が UNICODE で記述され、utf-8 形式でファイル保存されていても、プロジェクトのプロパティシートにおいて UNICODE ではなく「マルチバイト文字列を使用する」と設定することにより、プログラム中で画面やファイルに出力するリテラル文字列の中の日本語は Shift-JIS 形式で出力される。一方、NDK では、ソースコードを保存した際に使用した形式のまま、文字列として出力される。従って、プログラム中に埋め込まれた日本語メッセージ等に関しては、ソースコードを utf-8 形式で保存することにより、同じソースコードを使用しつつ、Windows 上の実行形式では Shift-JIS 形式で、また携帯端末においては UNICODE で表示することにより、プログラム中にデバッグ用に挿入したメッセージ等を端末上で読むことが可能となる。

⑩ 開発環境のハングアップ対策

端末エミュレータでプログラムをテスト中にハングアップ状態になった際に、最終的な手段として Windows 側のタスク・マネージャを起動して Eclipse 開発環境を強制終了させると、開発環境自体が再起動できない（エラーとなる）状態になる場合があった。恐らく、終了時点の作業状態を保存し、次回起動時にその状態から作業再開する便宜のための機能が悪影響を及ぼしているものであろう。この場合、統合開発環境のアンインストール、再インストールを行うと時間損失が大きい。workspace/…/workbench.xml を削除すると、Eclipse が再起動するようになる。

ADT が何らかの原因で破損した場合、再セットアップする際に、以前のダウンロード(zip 形式)が保存されていないと、新たなバージョンしかダウンロードできず、Eclipse とバージョンが整合する新たな Eclipse のセットアップから再作業が求められる。バージョンが不整合の場合、「現在表示されているページに無効な値が・・・」というメッセージが表示されて、Eclipse が起動せず、作業できない状態に陥る。

このような場合、eclipse.ini の最後の行を削除するというやや乱暴な方法がある。

⑪ 携帯端末実機へのプログラム導入と実行

実機にアプリケーションのセットアップのためのファイル（拡張子.apk）を送付する方法として、外付け SD カードの抜き差しを行う方法と、USB 接続で携帯端末内部のファイルシステムに、WindowsPC の外部記憶装置のようにアクセス（例えば、エクスプローラを使用）する方法を採ることができる。実機においてセットアップを実行するためには、実機の側でファイルを指定して実行するためのアプリケーション（例えばファイルマネージャ）が利用できなければならない。これは機種によりプレインストールされている場合と、WEB から無償でダウンロードして使用する場合があった。

次に、アプリケーションを実行するために必要なディレクトリ構成と各種ファイルを転送してセットアップが完了する。

開発を終えた後に、レンタルで短期間調達した様々な機器でのセットアップとテスト、デモを行った結果、最も容易と考えられる手順について、「２－２．指導員のための手引き」の中で解説した。

ソースコードの中で記述する、VC-3M に関連するファイルを格納したディレクトリの名称「/mnt/sdcard/」は、外付けではなく内蔵の SD カードを指しており、通常は抜き差しすることができず、あたかも PC の C ドライブのような役割を果たしている。

アプリケーションを実行した結果としてこのディレクトリに作成されるログファイル等を実機上で（つまり現場で）確認するためには、テキストビューワが必要である。このためには HTML ビューワや、簡単なワードプロセッサを使用することができる。

⑫ 実機の画面サイズ、センサ計測値等の確認

プログラムで取得した GPS センサ等の計測値が正しいかを確認するために、GPS アプリケーションである「GPS status」を利用して、計測値を比較した。

更に、計測値を直ちに文字表示、ファイルに記録する機能を有する小さなアプリケーション(SensorEx)を作成して、テストに使用した。この開発のヒストリーに関しては、３－４で解説した。

最初に動作を確認した段階では、機種を特定し、画面解像度などの様々なパラメータは定数としてリソースの中に埋め込んでこれを使用した。一方、Android の各バージョンを搭載した携帯端末には様々な機種があり、急速に多様化しつつある。従って、様々なマシンで使用できるようにするためには、機器毎の性能値などを API 関数で取得し、これを用いて処理を行う必要がある。本稿までの段階で対応した主な点は、以下の通りである。

１）背面カメラの解像度

背面カメラの解像度は、複数選択可能であることから、サポートされている全ての解像度について、表示画面の解像度に最も近い解像度を選択するようにプログラムで設定した。RealtimeActivity クラスの getOptimalSize 関数において取得し、surfaceChanged 関数において設定している。

２）画面の縦横サイズと、自動切り替え機能への対応

多くの機種において、携帯端末の画面を垂直にした状態で、長辺を縦にするか、横にするかで画面の縦横を自動的に切り替える機能を有している。初期の開発に使用した機種の場合には、以下のようになっている。

リスト 4-5-13 携帯端末における姿勢情報の特性

画面を横長（LANDSCAPE）に構えたときの、右を W 軸、上を V 軸、画面上方を U 軸とする。 地面に固定した座標系を考え、東を X 軸、北を Y 軸、上方を Z 軸とする。 ロール角（ツイスト角）を R、ピッチ角（俯角）を P、ヨー角（方位角）を Y とする。 端末の U 軸の傾きが４５°を超えると、画面が垂直に構えられたモードとなり縦横判定と画面の縦横自動切り替えが行われるようになる。それよりも小さい場合には、画面が水平に構えられたモードとなり、画面の縦横自動切り替えは行われない。画面が下向きの場合には、垂直に構えた場合と同様のモードである。 画面が垂直に構えられたことは、P（ピッチ）値が、－１３５°～－４５°の範囲であることによって判定される。

画面が垂直のモードの場合、
Ｐ値は、Ｖ軸のＺ軸に対する角度である。画面が水平で０°、画面を下方に向けると１８０°
　端末の画面が上の場合には、－９０°よりも大きくなり、最大－４５°
　端末の画面が下の場合には、－９０°よりも小さくなり、最大－１３５°
　ロールがかかっていると、－９０°になることはない。
　（例えば、－６０°から、－１２０°にジャンプする）
　また、端末をＸＹ平面に対して直角に立て、ツイストを行うと、
　ＲとＰの値が同時に変化する。Ｐが－４５°になった地点で、Ｙがジャンプする。
Ｒ値は、Ｗ軸のＹＺ平面に対する角度、つまりＷ軸のＺ軸に対する角度を９０°から引いたものである。
　ランドスケープに構えた時に、Ｒ値は、０°である。
　時計回り、右下がりに傾けた時－４５°まで減少する。
（それを超えると、意味が変わり、－７５°にジャンプする）
　反時計回り、左下がりに傾けた時、４５°まで増加する。
（それを超えると、意味が変わり、７５°にジャンプする）
Ｙ値は、Ｗ軸の水平投影面のＸ軸に対する角度である。
　そこで、API 関数から得られる RPY 値から携帯端末の姿勢を求めるためには以下の計算を行う。
（１）まず、Ｐ値が-135°～-45°の範囲にある場合には、画面が垂直のモードで、姿勢計算を行う。
（２）それ以外の場合については、画面が水平に近いものとして姿勢計算を行う。
　但し、画面が下向き（注視が上）の場合には、Ux,Uy の符号を反転する。

以上の知見から、以下のように処理することで正しく表示することが可能となった。

リスト 4-5-14 ロール・ピッチ・ヨー値と注視ベクトル、上方ベクトル

センサが返すロール値をＲ、ピッチ値をＰ、ヨー値をＹとすると、
注視点のベクトルＶは、
Vx = cosYsinR - sinYsinPcosR
Vy = sinYsinR + cosYsinPcosR
Vz = -cosPcosR
上方のベクトルをＵは、
Ux = -sinYcosP
Uy = cosYcosP
Uz = -sinP

画面サイズの取得関数は、縦横切り替え後の数値を返している。このため、解像度が大きい方を横とする絶対的な画面サイズを使用するように工夫した。また、背面カメラの取得する画像の縦横サイズと表示画面の縦横サイズは比例しない。表示における位置ズレを防ぐためには、背面カメラの画像を主体とし、表示画面の余剰部分は黒く残して表示に使用している。

画面サイズの取得の処理は、RealtimeActivityクラスのinitSurfaceView関数において、以下のように実行している。

リスト4-5-15 携帯端末における画面サイズの取得

```
FrameLayout frame = (FrameLayout) findViewById(R.id.cameraframe);
/* 131206修正前：簡単にリソースファイルに書き込んだ文字列から取得していた
int width = Integer.valueOf(getString(R.string.preview_width)).intValue();
int height = Integer.valueOf(getString(R.string.preview_height)).intValue();
*/
WindowManager wm = (WindowManager)getSystemService(Context.WINDOW_SERVICE);
Display disp = wm.getDefaultDisplay();
int width = disp.getWidth();
int height = disp.getHeight();
frame.addView(mSurfaceView, width, height);
```

３）センサの座標軸の自動切り替えへの対応

初期の試作段階では API 関数により取得するロール・ピッチ・ヨー（アジマス）の角度を用いて表示を行っていたが、上記の画面縦横切り替えに伴い、取得値が非連続で変化するため、機種の多様性に対応するために、三次元表示においては最終的に縦横自動切り替えの影響を受けない三軸センサ計測値（加速度および磁気）からプログラムで姿勢を直接

算出する方法とし、角度を使用せずに回転を表現することができる四元数を Java プログラムの側で生成し、ダイナミック・リンク・ライブラリの関数もこれを受けて表示に必要な視点・注視点等のパラメータを生成するように修正した。

⑬ 多重起動した複数のプロセスの干渉のテスト

一つの VC-3M プロセスが起動され、終了しないまま休眠状態になっている時に別の VC-3M プロセスが起動し、シャッター操作が行われ、終了して記録データが保管された後、休眠状態になっていた最初のプロセスが再度アクティブになり終了すると、二番目のプロセスが保存した記録データが上書きされて失われてしまう。

Android のアプリケーションの画面には OS 自体の初期画面も含め、←（戻る）、↑（ホーム）、□（マルチタスク）のボタンがシステムにより標準的に表示されている。

←ボタンでは、アプリケーションの内部においては、ある画面を終了して一つ上の階層の画面に戻る。アプリのトップ画面の場合にはアプリを終了する。

↑ボタンでは、アプリケーションのどのような階層にあっても、ホーム画面に戻る。このとき、アプリケーションは終了せず、単に表示が消えただけの状態となっている。

□ボタンでは、動作中のアプリと、近過去に動作したアプリが同じように表示される。アプリを選択し、表示されている状態で←ボタンで明確に終了するか、設定(アプリ)を開き、[アプリケーション][アプリケーションの管理]でアプリを選択し、強制停止を行う。

VC-3M の場合、例えば現場でのリアルタイム表示を行っている状態で↑ボタンが操作されると、表示からは画面が消えるが、写真合成のプロセスは動いたまま、OS の画面に戻る。

この状態で□ボタンが押され、動作中の VC-3M が選択されると、リアルタイム表示画面が再び表示されるため、問題は生じない。現場での作業は続行される。

一方、この状態で新たなプロセスが開始してしまうと、上記のような問題が生じうる。

VC-3M のアイコンがタップされると、表示から消えていた動作中のアプリが再び表示されるならば問題は生じない。

同じアプリケーションが多重に起動しないように設定するためには、AndroidManifest の中で、activity タグの中の属性 android:launchMode を singleTask に設定する。

リスト 4-5-16 AndroidManifest.xml の内容

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest    android:versionName="1.0"    android:versionCode="1"    package="jp.go.nilim.itc.virtualconverter"
xmlns:android="http://schemas.android.com/apk/res/android">
<uses-sdk android:minSdkVersion="13"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.ACCESS_SURFACE_FLINGER"/>
<uses-permission android:name="android.permission.READ_FRAME_BUFFER"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_MOCK_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<application
 android:label="@string/app_name"
 android:icon="@drawable/ic_launcher">
   <activity
        android:name=".VirtualConverterActivity"
        android:label="@string/app_name"
```

```
        android:launchMode=”singleInstance”
        android:screenOrientation="landscape">
    <intent-filter>
      <action android:name="android.intent.action.MAIN"/>
      <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
  </activity>
  <activity android:name=".realtime.RealtimeActivity" android:screenOrientation="landscape">
    </activity>
  <activity android:name=".replay.ReplayActivity" android:screenOrientation="landscape">
    </activity>
  <activity android:name=".selectimage.SelectImageActivity" android:screenOrientation="landscape">
   </activity>
  <activity android:name=".selectmodel.SelectModelActivity" android:screenOrientation="landscape">
   </activity>
</application>
</manifest>
```

Android OS 上のアプリケーションが起動している状態はプロセスと呼ばれる。それぞれの画面に対応する単位がアクティビティと呼ばれる。VC-3M アプリにおいては、

１）メイン画面 VirtualConverterActivity

２）モデル選択画面 SelectModelActivity

３）リアルタイム表示 RealtimeActivity

４）縮小画像選択画面 SelectImageActivity

５）再生表示 ReplayActivity

の５画面がアクティビティを有している。各アクティビティには、「インテント」と称する関連オブジェクトへのメッセージの送付により、センサ設定計測開始や画面表示ユーザ指示待ちの準備を要求する段階と、計測値変化やユーザ操作を受けたコールバック（メソッド）がプログラムされている。

インスタンスとは、クラスという型を任意個数メモリブロックとして実体化したものである。背景画像とその縮小画像に、視点情報や時刻合わせたシーンというクラスを、具体的なシャッター操作の回数だけ具体化したものがインスタンスであり、各インスタンスに属する縮小画像をマトリクス状に並べることにより画像選択画面が構成されている。

⑭ 画像ファイルのロード

VC-3M においては、古写真から復原した住宅等の立体に関して、写真から切り出した画像の一部をテクスチャとして適用している。このため、テクスチャファイルとして、α値（１から透明度を引いた値を 0-255 の整数値として表現）を定義することができるＳＧＩ形式を使用することとし、これをロードする処理を追加した。ファイルの読み込み自体は、景観シミュレーションシステムのためのライブラリ関数のソースコード（C 言語）を活用して NDK のビルドに加え、Android 上でもプログラムの修正なく処理することができた。但し、読み込み後の画像のサイズ最適化に OpenGL の処理を利用しているため、MLoad 関数の中でモデルの読み込みと同時に実行すると、機種や OS のバージョンにより正常に表示が行われない場合があることが判明した。そこで、この段階では読み込みを保留しておき、最初に視点移動が行われ MMove 関数が呼び出された時点で OpenGL が有効になってから読み込むように修正した。２度目以降の MMove 関数呼び出し時には、テクスチャ画像は既

にロードされていることを確認するのみで何もせずに表示処理に進む。

（7）サーバと SQL データベースの開発環境

VC-4D は、メタファイルから生成したコンバータ実行形式がデータを解読した結果呼び出すライブラリ関数群が、SQL サーバ上にテーブルを構築する。処理が完了した段階では、テーブル群が作成される。

この状態では、保存媒体と形式が変換されたのみであって、利活用の目的はまだ達成されていない。また、データベースを物理的に保存しているファイルとその記録形式に関しても長期保存性が保証されているわけではない。この処理系の特徴は、データベース上に展開された保存データに対して、第二のメタファイルを適用することにより、次の段階での利活用が自由に行える状態を準備したことにある。その具体的な例として、テーブル群に順次アクセスして、利活用時点で必要とされる、保存ファイルとは異なる、他の任意形式を定義するメタファイルを指定することにより、第二のファイル形式として再出力することを可能とした。

この第二のメタファイルが呼び出すことのできる、データベースへのアクセスのためのライブラリ関数群を増補し、同じ実行形式である VC-4D.exe を、保存時に作成され添付された解読のためのメタファイルと、利活用段階で必要となる形式を指定するデータアクセスのためのメタファイルを引数として２回起動することによりサーバ上で解読処理と利活用処理の２工程が行えるようにした点に VC-4D の特徴がある。

但し、この利活用のためのライブラリ関数群は、データの解読指示書としてのメタファイルのコンパイラの目的の外にあるものであって、利活用目的によっては将来修正され更に増補されても、本来の長期保存という役割に影響することはない。

実運用においては、この処理全体の進行を Windows2012Server 上にセットアップされた java 言語で記述されたサーバ側のアプリケーションが管理し、コンパイラ・インタプリタの処理を行うコンソールアプリ VC-4D.exe がデータファイルとメタファイルと SQL サーバの間のデータ処理を行っている。開発に際しては、コア部分の VC-4D.exe を作成するために、Window 上での小規模な SQL サーバを用いた開発環境で段階的に進めた。

①SQL サーバ

VC-4D は、実運用に際しては、セキュリティの管理が求められるが、初期のアルゴリズム開発段階では簡単のためにデータベースエンジンである SQL サーバとして SQLEXPRESS 2012 を使用した。このサーバは、開発環境をセットアップした OS である Windows 8 にプレインストールされていたものである。

開発成果を運用する WEB サーバにおいては、SQL2012 Server を使用しており、その設定方法などについては、２－６で解説した。

1) アクセス方法

データベースが ODBC(Open Database Connectivity, SQL サーバを含むデータベースの

みならず、テキストファイルや表計算ソフトの保存ファイル等にもアクセスするための共通インターフェース)に登録されている状態で、コマンドライン(cmd.exe、または PowerShell.exe)から、sqlcmd.exe を用いてアクセスし、SQL コマンドを実行することができる。SQL2012 へのアクセスになお、windows 2000 までは同様のツールとして低レベルのドライバを用いた、isql.exe、ODBC ドライバを使用した osql.exe が使用されていた。OS として Windows 8, 2012 server においても、osql.exe はまだ利用可能である。sqlcmd は当初 OLE DB(Object Linking and Enbedding Database)をインターフェースに用いていたが Windows 2012 から ODBC に回帰した。

sqlcmd -S サーバ名-E で存在とアクセスが確認できる。

接続するインスタンス名の先頭に明示的にアクセス方法（３種類の内１）を示す場合：

TCP アクセス:sqlcmd –S tcp:127.0.0.1,1433 -E

共有メモリアクセス（ローカル プロシジャー コール）：

sqlcmd -S lpc:.¥SQLEXPRESS -E

名前付きパイプアクセス：

sqlcmd -S np:¥¥.¥pipe¥MSSQL$SQLEXPRESS¥sql¥query -E

インスタンス名(“eqlexpress”)のみによるアクセス方法:sqlcmd –S .¥SQLEXPRESS -E

(この場合、SQL Server 構成マネージャで指定したアクセス方法を使用する)

アクセスが成功すると、１＞のプロンプトが表示され、SQL コマンドを受け付けるようになる。ここで、リスト 4-5-17 に示した SQL 文を実行することにより、現在のアクセス方法を確認することができる。

リスト 4-5-17 アクセス方法を確認するための SQL 文

```
１＞SELECT net_transport FROM sys.dm_exec_connections WHERE session_id=@@SPID;
２＞GO
（結果１：共有メモリの場合）
net_transport
------------------------------------
Shared memory
（名前付きパイプの場合の結果）
net_transport
------------------------------------
Named pipe
（名前付きパイプの場合の結果）
net_transport
------------------------------------
Named pipe
```

2)ログインアカウントの設定

まず、リスト 4-5-18 の手順で SQL 認証を行えるようにする。

リスト 4-5-18 SQL 認証とする手順

```
SQL Server Management Studio において、エンジンを選択し、SQL 認証に変更
SQL サーバを一度再起動する
```

次に、コマンドラインからリスト 4-5-19 に示した SQL 文を実行することにより、パスワード付きの sa でログインできるようになる。

リスト 4-5-19 パスワード付き sa の有効化

```
>ALTER LOGIN sa ENABLE
>GO
>ALTER LOGIN sa WITH PASSWORD='pasword'
>GO
```

3)実行形式 VC-4D.exe へのパラメータの受け渡し

VC-4D においては、WEB アプリケーションから、スタンドアロンの Windows アプリとして、VC-4D.exe を起動する。その際に、メタファイル名、データファイル名、データベース名、入出力フラグ、ログファイル名を引数として渡し、データベースのログインなどに用いるパラメータは環境変数として渡している。

起動時に main 関数への引数として以下を渡している。

リスト 4-5-20 VC-4D.exe の main 関数への引数リスト

```
argv[1]:メタファイル
argv[2]:データファイル
argv[3]:データベース名
argv[4]:0-2 の整数パラメータ(I/O)
argv[5]:ログファイル名
```

また、環境変数として以下を設定している。

```
env S=su, U=sa, P=si, D=SQL_D
```

なお、env 変数は他に多数あり、実行環境により異なるため、全てをキーで検索して代入している。

slog.txt と result.txt の両方が作成される。

パスワードについては､明示的な文字列を使用せずに､呼び出し元で codesave 関数により、exe.bin ファイル（バイナリ）に保存しておき､VC-4D.exe の側では､codeload 関数により、exe.bin ファイルから暗号化された SQL サーバパラメータを取得する｡実運用においては、設定内容を反映した exe.bin ファイルを VC-4D.exe と共にサーバ上にセットアップする。

開発環境においては、Windows アプリから VC-4D.exe を起動することで、サーバ上での動作を模擬的に再現し、様々のテストとデバッグを行った。このテスト環境においては、呼び出し元の VC-4DDlg.cpp がユーザーインターフェースを提供し、ここからプロセスとして起動され実際に変換処理を行う実行形式の VC-4D.exe が変換処理を行う。

VC-4D.cpp が変換処理を行う。

sqllib.cpp が SQL サーバとの入出力処理を行う。

cci ソース群をビルドに含める

実行形式は、VC-4D.exe

4) C/C++言語からのデータベースへのアクセス

a. ODBC のドライバを直接使用する方法

#include <odbcinst.h>

#include <odbcss.h>

（VC/PlatformSDK/include ディレクトリにある）

にて関数プロトタイプを取得してプログラムを作成し、

odbc32.lib, odbcbcp.lib, odbccp32.lib をリンクする。

（VC/PlatformSDK/Lib ディレクトリにある）

b. ADO(Active Data Object)を使用する方法(VC-4D で使用した方法)

リスト 4-5-21 ADO を利用する場合の#import 宣言

```
#import "C:¥Program Files¥Common Files¥System¥ADO¥msado15.dll" ¥
     no_namespace rename("EOF", "EndOfFile")
//Appdata/temp にヘッダー.tlh が作成される Precordset の定義はここにある
```

これにより、msado15.tlh というヘッダーファイルで定義された、

リスト 4-5-22 データベースエンジンへの接続方法

```
//接続に用いるポインタの宣言
_ConnectionPtr m_con;
_RecordsetPtr m_rs[hTAIL];
```

```
//接続の手順
hr = pConnection.CreateInstance(__uuidof(Connection));
hr = pConnection->Open(接続文字列,ユーザ名,パスワード,adConnectUnspecified);
```

```
//接続文字列のいくつかの例
"Provider=sqloledb;Data Source=" + su + ";Initial Catalog=" + pDBname;
"Provider=MSDASQL;DRIVER=sql server;SERVER="+su+";DATABASE="+pDBname;
```

```
//SQL文の実行
m_rs[JT] = m_con->Execute(sql文,NULL,adCmdUnknown);
```

を用いて、データオブジェクトの一種として SQL server2012 にアクセスする。

使用するドライバやデータベースについては、接続文字列により識別する。

なお、プロバイダのMSDASQL(Microsoft OLE DB Provider for ODBC) は、「Provider=・・・」が省略されると、デフォルトで使用される。

接続が確立した後、SQL文を文字列引数として実行し、結果をレコードセットとして受け取り、内容に順次アクセスする。

結果的に、C＋＋言語で記述した VC-4D からは、ADO(Active Data Object)という抽象化された一種のライブラリを用い、MSDASQL という ADO のプロバイダを介して、ODBC（OpenDataBaseConnection）という抽象化されたデータソースに登録された SQL Express 2012 に対して、共有メモリという接続方法を用いて接続している。BASIC 言語を用いて様々なデータオブジェクト（例えばワープロ原稿やメールデータ等）を SQL データベースと同時にアクセスするような処理プログラム作成のためには便利なプログラミング環境であるが、VC-4D 処理系のような特定目的での SQL データベース使用の処理系においては、いわば SQL データが４階にあって、１～３階は全て空室であり、３階から４階へは階段が３あるが、常にその内の一つしか使用しないような使用方法であり、処理プログラムでは、SQL データベースにアクセスする以前の１階から４階まで登るためのコードと設定等の準備作業が求められ、多くの資料を閲覧して 1990 年代の最新抽象概念を復習する必要が求められ、エラー処理を記述しなければならない。しかも、下の階の方が新しく追加されたものであり、今後短期間の内にアップグレードされる可能性が高く、その場合、対応しなければ本処理系が利用できなくなる可能性が高い。SQL データベースが平屋としてアクセスできた時代と比較すると、単純作業のために高度な万能工具を用いる観がある。

一方、本処理系においては、cci_sql.c によるライブラリ関数の実装において、printf のよ

うに使用できる Q 関数を使用して、例えばリスト 4-5-23 のよう SQL 文を発行する。

リスト 4-5-23 Q 関数を用いた SQL 文の実行

```
Q("SELECT * FROM %s WHERE id=%d",tablename, id);
```

この方法は、１階から３階の部分が改変されても、そのインターフェース部分の修正は sqllib.cpp というソースコードの中で対応でき、メタファイルの処理結果を SQL 文に変換する cci_sql.c の中でのライブラリ関数の実装自体は、SQL 言語が改変されない限り再利用できる可能性が高くなるように考慮したものである。引数に含まれる SQL 文字列等に含まれる不適切な要素については Q 関数の中で検査などを行う。

5) VC-4D.DLL

景観シミュレータのためのプラグイン DLL として、DML<=>SQL の交換を行う。データベース関連パラメータは、メニュー項目の中のサーバ設定において設定する。

ダイアログで設定変更されると、codesave 関数が exe.bin という暗号化したファイルに設定内容を保存する。プラグインの起動に際して、codeload 関数が、ファイルからサーバ設定パラメータを読み込む。このファイルの作成場所は、VC-4D.DLL の場合には kdbms.set において E3_ENV_SNAPSHOT の項目に指定された場所とした。テスト環境として使用した VC-4D.exe の場合には、このファイルの作成場所は実行形式が置かれたディレクトリである。VC-4D.DLL のビルドを構成するソースコードをリスト 4-5-24 に示す。コンパイラ処理系を含まないため、３種類のみとなっている。

リスト 4-5-24 VC-4D.dll のビルド構成

```
4Dwnd.cpp がユーザーインターフェース画面の処理と変換処理を行う
VC-4D.cpp が DLL の起動と終了を行う
SQLDB.cpp が SQL との入出力処理を実行する
```

SQL サーバ上での三次元データの入出力処理を行う SQLDB.cpp が完成した後も引き続き、VC-4D.exe が生成したデータベースをこの DLL で読み込んで直ちに表示し、異常がないか検査するツールとして多用した。

6) コンソールアプリ VC4D.exe と起動テスト環境 VC-4D.exe

VC4D.exe は、必要なパラメータを受け取って、メタファイルをコンパイルした上で、データファイルを解読し SQL サーバに格納し、SQL サーバからデータファイルを生成する。

実運用ではサーバ上の VC4D Java アプレットから起動するが、プログラム修正に係る一回のトライアンドエラーにかかる時間が長いため、開発テスト環境として、ウィンドウアプリケーション VC-4D.exe を用意して、ここからプロセスとしてコンソールアプリを起動し、テストとデバッグを行った。テスト環境から、コンソールアプリを起動する際には、spawn 関数を使用した。このテスト環境のビルドを構成するソースコードはリスト 4-5-25 に示す。

リスト 4-5-25 VC-4D(win)のビルド構成

メタファイル名などのパラメータ設定と実行ボタンを有し、VC4D.exe を起動する Windows アプリの実行形式は VC-4D.exe であり、VC-4DDlg.cpp に主要なコールバック処理を記述する。

コンソールアプリである VC4D.exe が実際に仮想コンバータ関連の変換処理を行う。main 関数は VC-4D_exe.cpp に置く。sqllib.cpp が SQL との入出力を行う機能を集約する。コンパイラ・インタプリタ関連の cci ソース群は、VC4D.exe のビルドに含める

② cci_dml, cci_sql におけるログファイルの処理

（改良前の状態）コンパイラやインタープリタが出力するエラーメッセージや、メタファイル中の logf 関数、printf 関数などは、ksim/temp/CClog.geo に全て出力されていた。

一方、cci_sql, cci_dml においては、データベースやメモリ空間内のオブジェクトを外部ファイルとして出力するための機能を備えているため、最終的な成果である出力ファイルと、デバッグ用の情報を格納するログファイルが分離されている方が便利である。

そこで、cci_code　においてファイルハンドルとして、

FILE *outfile, *stberr

の２本を用意し、二つの出力先として使い分けることとした（sql, dml 共通)。ここで、logf 関数は、常に stberr に対して出力する。一方、printf 関数は、もし outfile が開いていれば、outfile に、NULL ならば stberr に出力する。

stberr は、C2VWnd::OnBnClickedOk()コールバックでコンパイルと実行処理を始めるのに先立って、常にオープンされる。一方、outfile は、チェックボックスによるユーザの選択において、処理が出力である場合に限りオープンされ、それ以外は NULL である。

よって、メタファイルとデータファイルを入力する処理において、ファイル出力系のコマンドや printf コマンドが実行された場合においては、データファイルに損傷を与えることなく、メタファイルの実行結果を出力する。

このコールバックを記述しているソースコード 2Vwnd.cpp においては、cci_export.h をインクルードしている。この cci_export.h の中で、

```
extern"C"{
  extern FILE *outfile;
  extern FILE* stberr;
}
```

として宣言を行っている。

③VC4D.exe の起動におけるパラメータの引き渡しについて

１）コマンドライン引数では、引用符「""」なしで、以下の引数文字列を渡す。

第一：メタファイル名

第二：データファイル名

第三：データベース名

第四：入出力フラグ（０：SQL=>file　１：file=>SQL)

第五：ログファイル名

２）環境変数

環境変数は、他にも多くの既存の設定があり、順番では指定できないため、ラベル＝値という形式を使う。VS2005 では、main 関数の第三引数として char*型の環境変数の配列を受け取ることができる。個数は不定で、終端は NULL であることで示される。

```
int main(int argc, _TCHAR *argv[], char*env[]);
```

U=ユーザ名

P=パスワード

S=サーバ名

のパターンを全ての環境変数から検索し、「=」の次以降を文字列型の値として取得する。

コンソール・アプリケーション VC-4D.exe をプロセスとして起動する開発・テストのための Windows アプリケーション VC-4D（win）からは、必要な環境変数を putenv 関数で設定し、引数（上記の main 関数が argv[]として受け取る値）を char*v[7]に格納する。このパラメータ構成は、リスト 4-5-20 に示した。これを引数（配列ｖ）として、spawn 関数を用いて vc-4d.exe をプロセスとして起動する。

リスト 4-5-26 spawn 関数による VC-4D.exe の起動

```
Pid = _spawnv(_P_NOWAIT, “vc-4d.exe”, v);
```

④VC-4D(win)を用いた SQL サーバのテスト

OS や SQL サーバのバージョン更新に伴い、新たなサーバにセットアップする際に、データベースへのアクセス等に障害が生じた場合に、原因の絞り込みを行うツールが存在すると便利である。Windows アプリである VC-4D.exe には、そのようなツールを組み込んで、実際に Windows2012 への移行に際して活用した。具体的には、VC-4D(win)が起動しメイン画面が開く前に、接続確認を行う画面 SqlConnector を用意し、テキスト入力欄に様々な接続文字列を入力して、接続テストを行う。成功した接続文字列を記憶しておいて、これを以後の SQL サーバのアクセスに使用する。この実行形式を、セットアップ先ないし移植先のサーバで起動することにより、サーバの設定を調整し、コンソールアプリの起動条件を設定することができる。

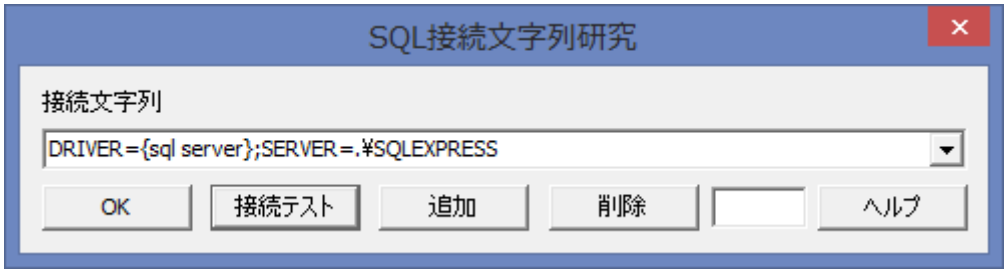


図 4-5-7 SQL サーバへの接続文字列テストツール画面

⑤VC-4D(win)を用いた、コンソール・アプリケーション VC4D.exe のデバッグ

プロセスとして起動する SQL 処理の中で障害が生じた場合には、それを再現し原因箇所

を特定する手段が限られている。このような場合に、VC-4D から、プロセスを起動することなくスタンドアロンでライブラリ関数の内部処理を追跡できると便利である。

VC-4DDlg.cpp の中で、メタファイルとデータファイルを指定してアップロード、ダウンロードの処理の開始を指示している。アップロードは、同じソース中の upload 関数、ダウロードは download 関数で処理を行っている。_spawnv 関数を用いて起動しようとする実行形式 vc-4d.exe がファイルとして存在しない場合には、変換処理を同じライブラリ関数のソースコードを用いて内部で実行することにより、デバッガで処理を追跡できる。これは、開発の初期段階の機能に回帰することに相当する。

VC-4D.exe は、VC-4D_exe.cpp という main 関数を有するソースコードから、コンパイラ等の処理を起動している。この関数以下を、VC-4D のビルドに含めても、この main 関数が呼び出される事はない。画面の[開始]ボタンの操作に対するコールバックの中で、VC-4D_exe.cpp の中の upload, download 関数を呼び出すことにより、コンソールアプリと同じ処理をトレースしデバッグすることができる。コンソールアプリは、デバッグ後のソースコードを用いて再ビルドし、サーバ上で実際の動作を確認することができる。

（８）おわりに

このように過去３０年間、システム開発の生産性を上げるための様々な手段が利用可能になった。建築工事に準えるならば、初期のアセンブラは鑿や鋸のような手工具である。C 言語は、手押し鉋や丸鋸に喩えられるであろう。JAVA や C#は、気の利いたノックダウン（プレファブ）建築である。スクリプト言語は、日曜大工向けの万能工具である。開発する処理系に応じて、また開発チームの規模に応じて使い分けられる。うまく行かなかった時、バグの原因が特定できないとき、プログラムのバグか、コンパイラのバグか、あるいは最悪ハードウェアのバグか、追い詰めなければならない。CPU のバグであれば、直すのは困難であり、特定の機械語を使わないように回避しなければならない。そのような修正ができることも必要であるように思う。2000 年代、Java が全盛の頃、書店の計算機のコーナーには C 言語に関する書籍はほとんどなくなった。スマホやタブレットが普及し始めてから、処理速度を求める三次元画像処理などを実現するため、ネイティブコードの実行形式を開発する必要も生じ、再び C 言語に関する書籍が並ぶようになった。

①メタファイル処理系の位置づけ

本処理系の中心を成すメタファイルの文法においても C 言語を参考にしたが、多くの機能を省略した単純なものとなっている。セミコロン「;」をセパレータとする書法は、java、C#にも継承されているスタイルと類似しているが、クラスはおろか構造体も使用していない。メモリも静的に管理するのみである。

１）現在、C 言語スタイルは、レガシーコーディング

２）処理系がシンプルである（コンパイラ自体のソースコード量が少ない）。

３）特定のデータを対象とするコンバータが組めればよい（メモリ管理不要）

（全て有限長の配列で処理可能）

４）少なくともマシンコードよりも可読性が高い。
５）メモリ管理を行わない。特定のデータファイルだけを解読できれば良いという特権。
６）メタファイルでは malloc, calloc, realloc, free 等の処理を行わない。また、ガーベージコレクションも行わない。

このような単純な処理系であるため、C コンパイラが利用できるプラットフォームであれば移植は容易である。

②メタファイルの作成環境の展望

１）テキストエディタが利用できれば、携帯端末の上でもメタファイルを編集し、直ちにコンパイル・実行することができる。。
２）今後の課題として、メタファイルを作成するための、エディタ、コンパイラ、デバッガを一体化した統合開発環境存在していてもよい。メタファイル作成作業におけるビルドのメイクファイルや、デバッグ情報、バージョン管理情報などはデータの保存に際して添付する必要はなく、このような補助的な処理系を用いて完成した小さなメタファイルだけを、データファイルに添付して長期保存する。
３）現在 CAD などに広く使用されている各種データ形式の例示となるテンプレートを用意しておき、このような「雛形」から出発して、実際に添付するデータに合わせて不要な関数を削除し、固定長配列のサイズなどを最小限にするような作業方法が考えられる。
４）メタファイルのコーディング作業の改善のためには、メタファイルを出力するようなコンパイラなどのより高度な言語による処理系を時流 OS の上で別途構築すればよい。最終的に保存するのはメタファイルのみ。javascript を出力するコンパイラなどが存在している。

③必要な人材

プログラマという職業は、様々な分野で基礎を支える人材となっている。丁寧な仕事を行う日本的な職人の仕事の仕方は、世界的なプログラム作品とは別のジャンルを形成しているように考える。大がかりなツールを駆使して小さな作品を短期間にまとめるのではなく、津々浦々の一隅に散在する小さなシステムを長期にわたりメンテナンスするような関わり方である。

４－６．筑波移転機関の移転前の記録

平成 25(2013)年に、昭和 38(1963)年 9 月の筑波研究学園都市に関する閣議了解から 50 年を記念して各種の記念行事が行われた。本研究においては、従来の事業記録や各機関の社史で余り詳しく掲載していない移転前の研究機関の空間構成、跡地の利用、および個別機関の検討段階での移転計画案を記録した図面などの図形資料とそれらを用いた三次元的な復元記録データの作成に取り組んだ。まず、国総研に移転前後の資料が残されている、新宿百人町からつくば市立原に移転した建設省建築研究所に関して資料整理と試行的な復元を行い、その他の機関に関しても旧所在地と移転後の空間構成の変化を資料調査した。

（１）新宿百人町

この区域には、終戦後から 1970 年代まで、多くの研究機関が立地していた[註１]。この内、建設省建築研究所、東京教育大学光学研究所、資源科学研究所（後の国立科学博物館新宿分室）は 1980 年以降、筑波研究学園都市に移転した。また、移転しなかった機関として、都立衛生研究所、蚕糸科学研究所等があった（図 4-6-1）。この他、消防署、警察署、引き揚げ者住宅が立地し、一部は 1965 年までに中層の都営住宅に建替わっている。

図 4-6-1 筑波移転が決定された頃の百人町地区の状況（1965 年）[註２]

明治初期には「戸山が原」と呼ばれる林地があり、大正４(1915)年に設置された陸軍科学研究所の施設が置かれ、大正 12(1923)年関東大震災の後に建設された建物[註３]が加わっ

た様子が地形図に描かれている。1980 年代まで存在していた。昭和 16(1941)年に同研究所は改組され、新宿百人町地区は陸軍第六研究所（化学）と第七研究所（物理）となった。

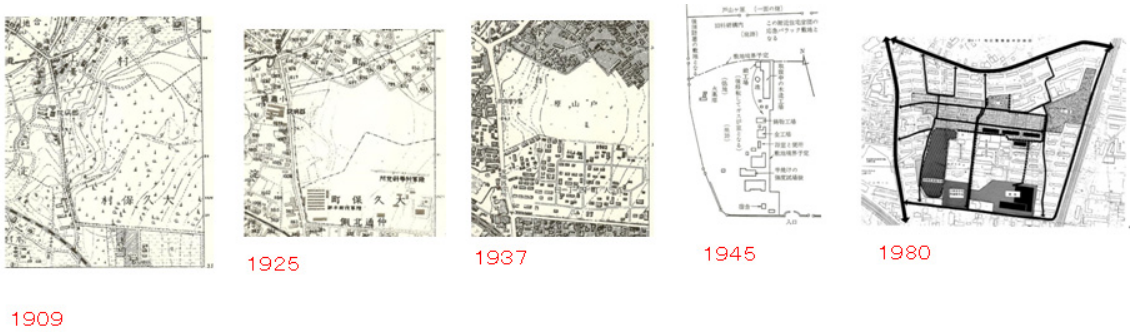
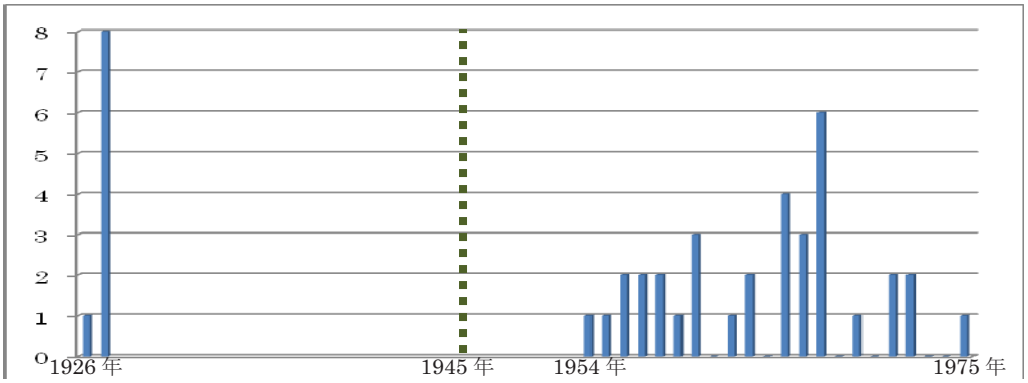


図 4-6-2　２万５千分の１地形図等にみる百人町地区の状態遷移[註 4]

（1945 年は 20 年史掲載図、1980 年は跡地利用報告書掲載図による）

終戦の前には、第七研究所は長野と金沢に疎開しており、大型施設のみ残っていたが、3 月 10 日の空襲で多くを焼失した[註５]。

これらの機関が廃止された後に、戦後にいくつかの研究機関が設置された。このとき、戦前から存在していた建物が新設機関の庁舎として再利用された。この内、筑波移転に至るまで建築研究所の本館として継続使用されていた建物は、関東大震災で罹災した後建設された旧第七研究所の建物（本館）であり、移転前に作成された実測図には大正 15 年３月と記入されている（表 4-6-1 の図面番号１）。翌昭和２年と記録されている建物としては強度試験場（同、２）、施工試験室として使われていた「別館」（同、３）、金工場（同、４）、鍛工場（同、５）、車庫（同、11）、倉庫（同、12）、電気溶接室（同、14）、油庫（同、15）として使われていた建物群があった。昭和 3～20 年の建物は存在しない。言い換えると、終戦時点の第七研究所の施設の内、古く関東大震災直後に陸軍科学研究所第一部が設立された時代(1925)の復興建物のみが戦時下の空襲に堪えて残っており、戦後の建築研究所の施設として再利用された。これに加えて 1954 年以降に建築研究所が整備した建物があった。



グラフ 4-6-1：筑波移転時の建物の年齢構成（表 4-1-6 より作成）

これらは、研究機関の筑波移転後全て解体除却され現存していない。跡地利用については、昭和 55(1980)年に検討されている[*65]。23 区内の 18 地区が検討対象とされ、特に新宿百人町地区と川口駅前地区についてはモデル的に詳細な検討が行われ、百人町地区に関して

は、病院施設を光学研究所跡地とする案と建築研究所跡地とする案の２案が提示された。前者の案の通りにおおむね実施され、建築研究所跡地には集合住宅（民間マンションおよび都営住宅）と防災公園が、また光学研究所跡地には、社会保険中央総合病院が建設されている。

（２）建設省建築研究所について

①成立

昭和 21(1946)年 4 月に戦災復興院の技術研究所として新たに設立された。「建築研究所２０年のあゆみ[*66]」をはじめ、10 年毎に作成された機関史はこの年を起点としている。

年表 4-6-1 に、同書第 1 章「建築研究所の概要」に記されている創立の頃の経緯を基にその他の史料から得られた事項を追記したものを示す。

年表 4-6-1 建設省建築研究所創立の頃の経緯

年月日	事項
昭和 14(1939)年 7 月 3 日	内務省防空研究所設立（世田谷区等々力。現、都営住宅、公園）
昭和 16(1941)年	内務省防空局設置　（計画局からの改組、後の建設省、自治省等の場所）
昭和 17(1942)年 12 月	大蔵省大臣官房営繕課に建築研究室を設置（掛から昇格、霞が関大蔵省地下，4 階）
昭和 18(1943)年 7 月 15 日	内務省防空研究所彙報[*62]
昭和 19(1944)年（月日未詳）	建築研究室が山梨県に疎開（相興村尋常小学校）
昭和 19(1944)年（月日未詳）	陸軍の研究所が長野と金沢に分散して疎開
昭和 19(1944)年 1 月	沼津に海軍施設本部野外実験所を設置（駿東郡清水村、現沼津河川国道事務所等）
昭和 20(1945)年 8 月	内務省国土局分室（防空研究所を前身とする）
昭和 20(1945)年 8 月 20 日	海軍第一技術廠(空技廠)が横須賀田浦を引き払う。この日まで営繕の疎開先である等々力の園芸学校の講堂裏等に機材を運搬（現在の追浜工業団地）
昭和 20(1945)年 8 月 26 日	海軍施設本部は、運輸建設本部となる
昭和 20(1945)年 9 月 8～12 日	GHQ の接収により、大蔵省建築試験室は庁舎を明け渡し、資機材を兜町に一時避難
昭和 20(1945)年 9 月中旬	竹山、沼津の海軍施設本部研究所を訪れ、山田所長に施設提供を断られる。
昭和 20(1945)年 9 月	運輸建設本部技術員養成所を設置（沼津、元の海軍施設本部野外実験所）
昭和 20(1945)年 9 月	竹山、目黒の海軍技術研究所を訪問（交渉不成立） （一時機械技術研究所が了承を得るが、オーストラリア軍が接収、現自衛隊）
昭和 20(1945)年 9 月 20 日	竹山、陸軍造兵廠の工員宿舎（現、大山寮）を訪問、研究所用施設として不適切
昭和 20(1945)年 9 月 23～24 日	竹山・久田、陸軍第七技術研究所を訪問、建築研究所の開設場所とする 24 日、「大蔵省営繕局研究所」の看板を掲げた
昭和 20(1945)年 10 月	旧第七陸軍技術研究所跡に建築研究室設営の事務を開始
昭和 20(1945)年 11 月	戦災復興院設置、技術工員養成所を大山に設置（板橋区板橋町四丁目 128 番地）
昭和 20(1945)年末	建築研究室の疎開先山梨県相興村から新宿百人町への移転を完了
昭和 21(1946)年 4 月	旧内務省防空研究所の一部と合併し、戦災復興院技術研究所として発足
昭和 21(1946)年 9 月現在	人員 67 名、うち研究員 33 名 研究部の下に都市計画研究科（含む、建設経済）、材料研究科、構造研究科、施工研究科、設計計画研究科、防火研究科
昭和 21(1946)年 11 月	技術工員養成所（大山）が等々力に移転（２年目）
昭和 22(1947)年 7 月	戦災復興院技術研究所報告第 1 号
昭和 22(1947)年 12 月	内務省廃止、戦災復興院と旧内務省国土局を統合
昭和 23(1948)年 1 月	建設院設置、建設院第二技術研究所と改名 この時、内務省土木試験所は第一技術研究所となる
昭和 23(1948)年 2 月	建設月報に、企画調査課、第一研究部、第二研究部の近況報告
昭和 23(1948)年 7 月	建設院が運輸省運輸建設本部を吸収して独立の省に昇格、 第二技術研究所は建設省建築研究所と改名 沼津の技術員養成所は、建設省建設工事本部の配下となる
昭和 24(1949)年 7 月	建設工事本部技術員養成所（沼津）は、建設省土木研究所技術員養成所となる このとき技術員養成所の建築分野は、建築研究所が第四研究部に吸収 ２課（総務＋企画調査）＋５研究部＋技術員養成所（等々力）の組織となる
昭和 24(1949)年 8 月	沼津より木工場、木工機械、宿舎を新宿百人町に移設
昭和 26(1951)月 3 月	技術員養成所（等々力）を解散（７期）

図 4-6-3 は、初期の研究者の出身機関の解説図である[註6]。

建築研究所の主な母体は戦前の大蔵省建築研究室（霞が関）であり、廃止された内務省防空研究所（世田谷区等々力）等の戦前研究機関からも研究者が集まっている。開設地に戦前存在していた陸軍技研（新宿百人町）からの出身者は少ないが、「金工場 etc.」と記されているように、研究機器・試験体等の制作を担当した職人は建築研究所に引き続き採用された。第七技所の廃止時点の幹部名簿に、図中にある藤井以外の名前は見られない。海軍航空技術廠は、横須賀田浦（追浜）にあった。運輸建設本部は小石川にあり野外実験所が沼津近郊にあった（後述）。

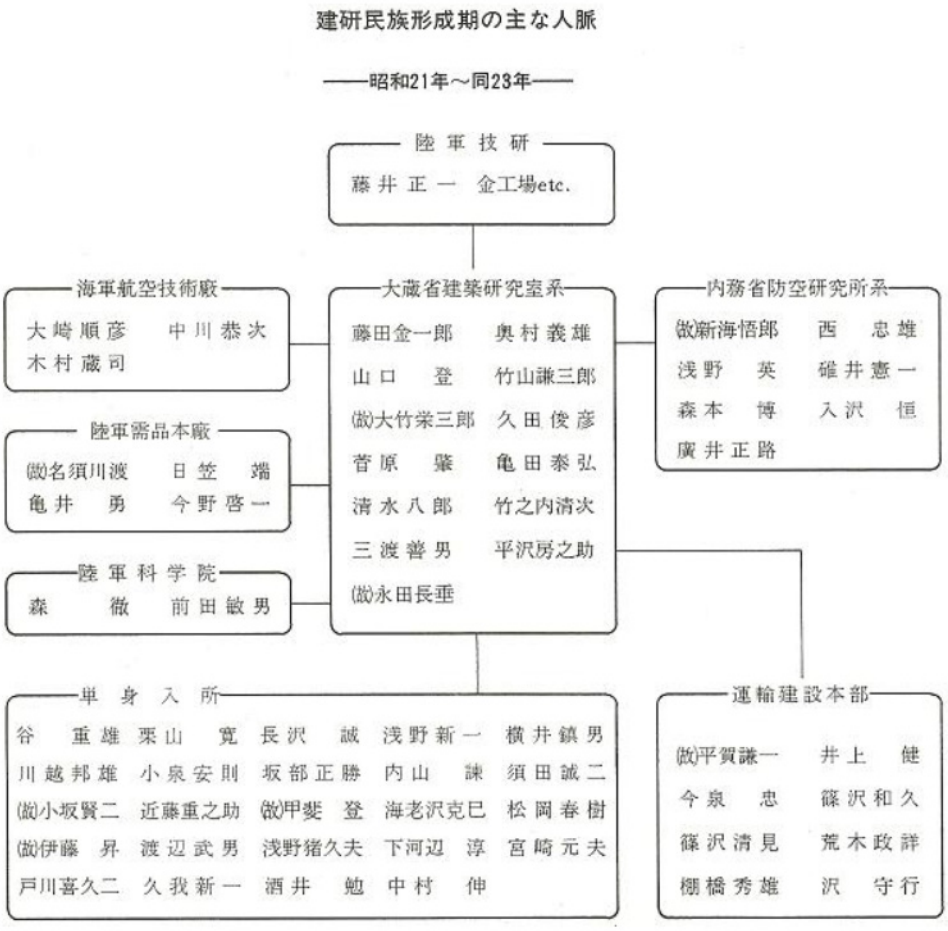


図 4-6-3　建築研究所初期の職員の出身元

②大蔵省建築研究室

東京都千代田区霞ヶ関での国会議事堂建設(年表 4-6-2)における石材の試験がその源流とされている[註7]。この工事がほぼ完成した昭和 10(1935)年頃に、建築研究所を設立する構想があったが[註8]、内務省防空研究所の構想と競合したため成立せず[註9]、昭和 17(1942)年、大蔵省営繕管財局に建築研究室が設置された[註7]。この頃、大蔵省の旧庁舎（大手町）に代わる新庁舎が霞が関に建設されていたが、資材不足の時代であり、工事は９年間にわ

たった[註 10]。建築試験室は昭和 18 年に竣工した霞ヶ関庁舎の４階にあり、試験機は地下にあった。昭和 19(1944)年にアムスラー社の 100 トン試験機などが山梨県相興村相興尋常小学校(2018 年現在の笛吹市一宮北小学校)に疎開していた[註 11]。また、大蔵省営繕課は等々力の園芸学校に疎開していたという記述もある[註 12](現在の都立園芸高等学校。当時等々力にあった内務省防空研究所からは北北東に 1.6km)。

しかし、工作機械や恒温槽などがまだ霞ヶ関に残されており、昭和 20(1945)年９月 8～9 日の GHQ 接収に際して多く廃棄され、電話機などわずかな機材を持って 12 日夕方までに兜町の証券取引所に退避したという[註 13]。このため、戦後の活動を再開するためには新たな場所を探す必要があり、竹山・久田らが９月中旬から沼津の海軍施設部研究所跡、目黒の海軍技術研究所、池袋の元陸軍造兵廠（大蔵省営繕課の大山寮）を調査した後に、9 月 23～24 日に大久保（百人町）の元陸軍科学研究所にたどり着き、この場所を移転先に選択するに至った。

年表 4-6-2 国会議事堂建設主要イベント

明治 20(1887)年	位置の決定
明治 41(1908)年 8 月	地質調査
明治 42(1909)年	国内木材・石材調査
明治 43(1910)年 7 月 22 日	議院建築準備委員会決議第 7 号の１ *「材料ハ本邦品ヲ資用シ」*[*106]
大正 7(1918)年 6 月 10 日	大蔵省臨時議院建築局発足、競技設計の結果１等渡邊福三案
大正 9(1920)年 1 月 30 日	地鎮祭、同年６月 26 日鍬入式
大正 10(1921)年	本館基盤工事開始
大正 11(1922)年 2 月 16 日	本館鉄骨組み立て開始
大正 12(1923)年 8 月 19 日	外装用石材（広島県倉橋島産）購入契約
大正 12(1923)年 9 月 5 日	関東大震災、現場は被害僅か、本省の設計図書等の資料を焼失
大正 14(1925)年 5 月 25 日	大蔵省営繕管財局発足、臨時議院建築局は廃止
昭和 2(1927)年 4 月 7 日	上棟式
昭和 11(1936)年 11 月 4 日	落成式典（７日間）

［長谷川：2000*[106]による］

③内務省防空研究所

世田谷区等々力には、昭和 6(1931)年に目黒蒲田電鉄が開設したゴルフ場「等々力ゴルフリンクス」)があった。この土地を買収して昭和 14(1939)年７月３日に防空研究所が設立された[註 14]。予算獲得は、大蔵省の建築研究所設立案と競合したとされている。初代所長は菱田厚介であった[註 15]。

昭和 14(1939)年 10 月 25 日には、内務省防空研究所長から、土木試験所に対して化学に関する依頼試験「擬装用着色剤に関する件」が記録されている*[61]

昭和 16(1941)年に、内務省計画局の改組により内務省防空局が成立した。

昭和 18(1943)7 月に内務省防空研究所彙報*[62]が出版されている。奥付には、*「東京都世田谷区玉川野毛町一〇〇三番地 電話田園調布 4001,4002,4003 番、玉川 370 番」*と当時の住所・電話番号がある。当時の所長は中沢誠一郎であった。

昭和 21(1946)年 4 月に戦災復興院官房技術研究所が百人町に設立された際に一部が合流した。このとき等々力から搬入された機材も百人町には存在した[註 4]。昭和 21(1946)年８月に*旧内務省防空研究所等々力分室が廃止され本所と統合*された（20 年史年表*[66]）。この「本

所」と想定される、防空研究所を前身とする内務省国土局分室が存在していた*108。

同年１１月には、前年板橋区大山に開設された戦災復興院技術員養成所[註 15]が移転し、２年目以降はこの等々力の防空研究所跡地で運営された(1946-1950)。この技術員養成所は昭和 26 年 3 月まで 7 期継続され解散した[註 16]。

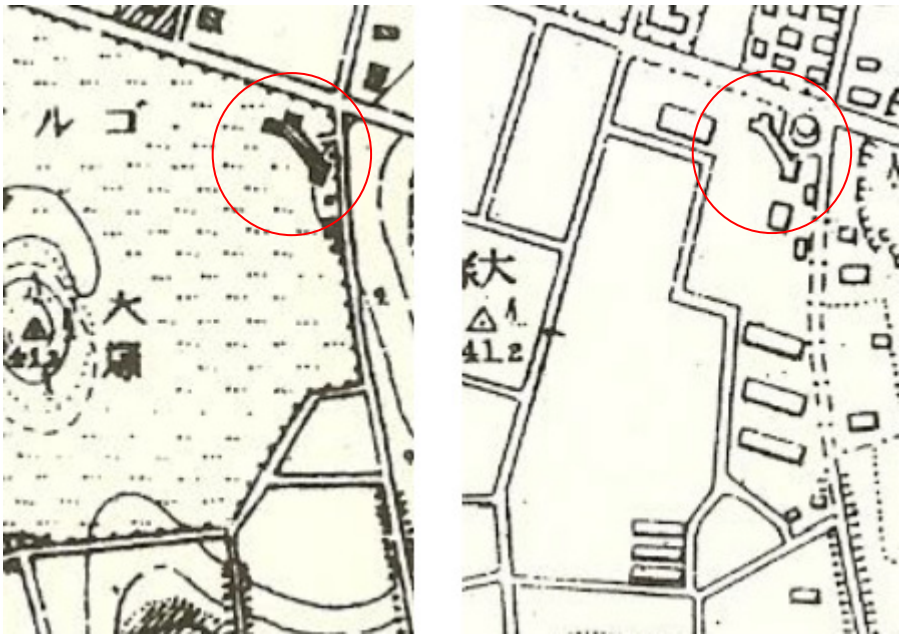


図 4-6-4：クラブハウスが描かれた地形図：昭和 12 年（左）　昭和 20 年（右）

写真 4-6-1：等々力の技術員養成所の玄関（元、クラブハウスの建物）[註 17]

その頃、敷地東北隅の入口には、ゴルフ場時代のクラブハウスが、防空研究所時代を経

てまだ存続しており、管理に使う他食堂などもここにあったという（写真 4-6-1)。敷地内部には防空研究所時代の木造の庁舎が残されており、防空研究所の残党も住んでいたという。空地では野菜やサツマイモを栽培していた[註 19]。この技術員養成所においては、１年間のコースで主に木造住宅再建に必要な大工技術の指導が行われ、コース終了後は戦災復興工事に従事することが義務づけられていた。

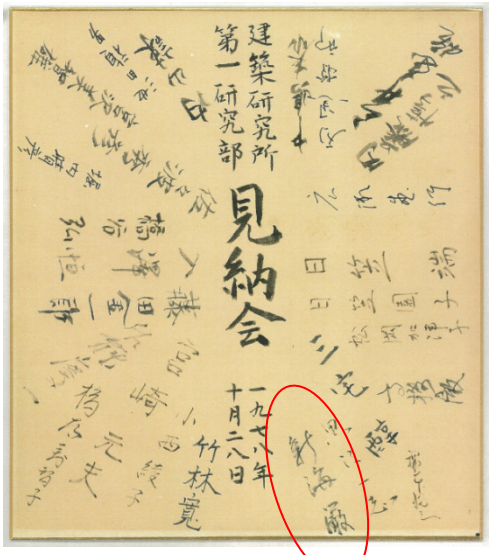
跡地には建設省官舎が建設された他、玉川野毛町公園グランドが整備された(図 4-6-5)。

図 4-6-5 等々力付近の状況：昭和 12 年（左）昭和 20 年（中）昭和 38 年（右）

防空研究所研究彙報 No.1 が国立国会図書館、No.2、3 が東大土木学科図書室に残されている[註 20]。この研究所においては木造住宅や市街地の防火対策のほか、疎開や外観偽装等の戦後復興都市計画の下地となる研究が行われており、戦災復興院技術研究所の研究部都市計画科を継承した建設省建築研究所第一研究部の人材を提供した。

戦災復興院技術研究所には、研究部の下に都市計画研究科（含む建設経済)、材料研究科、構造研究科、施工研究科、設計計画研究科、防火研究科が設けられ（９月時点)、防空研究所の出身者の多くは都市計画研究科に所属し、戦後の都市計画の基礎を築いた。城谷豊追悼文「新海さんのこと」*82に記された新海氏の略歴(年表 4-6-3)によると、昭和 22 年５月には第一研究部長、建設院第二技術研究所においては、第一研究部長となり、昭和 25(1950)年に建築研究所に研究部制が導入されると第一研究部に引き継がれた。都市計画研究はこの蓄積を継承している。新海悟郎は、在職中に殉職したため多くの研究資料を残した（新海文庫)。この中には戦後の全国各地の貴重な記録写真が含まれている。また、筑波移転を前にした 1978 年 10 月 28 日の第一研究部見納会には草創期の研究者と並んで夫人新海巌の署名が見える（図４－６－６)。

終戦から１９５０年代までに、都市計画研究科（後の第一研究部）を拠点とした都市計画研究連絡会の活動を通じて海外の制度などの吸収が行われており、戦後の都市計画は戦中の疎開を中心とした防空都市計画から復興・新都市建設のスキーム形成に向けて大きく展開した。これらについては最近研究されている[註 21]。



年表 4-6-3　新海悟郎略歴

明治 40(1907).12.9: 東京都世田谷区生
大正 14(1925).3:私立開成中学校卒
昭和 4(1929).3:松本高等学校理甲卒
昭和 8(1933).3:京大建築科卒
昭和 8(1933).4:警視庁保安部勤務
昭和 12(1937).11:内務省計画部勤務
昭和 14(1939).7:内務省防空研究所兼務
昭和 18(1943).11:内務省業務局勤務
昭和 19(1944).2:軍需相総動員局勤務
昭和 20(1945).11:戦災復興院計画局勤務
昭和 21(1946).5:戦災復興院総裁官房技術研究所兼戦災復興院建築局監督課勤務
昭和 22(1947).5:総理府戦災復興院技術研究所第一研究部長
昭和 23(1948).1:建設院第二技術研究所第一研究部長
昭和 23(1948).7:建設省建築研究所第一研究部長
昭和 31(1956).5:日本建築学会賞(都市木造住宅の老朽化傾向と防止対策)
昭和 33(1958).4:工学博士
昭和 34(1959).11:首都高速道路公団保障審議委員会委嘱
昭和 35(1960).9: 1 固定資産評価制度調査臨時委員任命
昭和 37(1962).4.29:永眠 (享年 54)

図 4-6-6 百人町見納会の参加者（第一研究部）

④技術員養成所（沼津）

昭和 24(1949)年 7 月に建築研究所（第四研究部）に併合された平賀謙一らの研究者[註 22]と、そこに勤務していた建設本部技術員養成所は、戦時中の海軍施設本部野外実験場と沼津海軍工作学校の組織を前身としていた。建築分野では、日本一を誇る木工機械工場を有しており、併合に伴って木工場と木工機械を新宿百人町に移設している。土木分野では建設機械化施工等を課題としていた。終戦直後、戦災復興に向けて運輸省の運輸建設本部技術員養成所となり、昭和 23(1948)年に建設省の設置に伴い、運輸建設本部は、建設工事本部となっていた[註 23]。

このとき技術員養成所の土木部門は建設省土木研究所に合流し、昭和 28(1953)年から沼津支所として機械施工の研究・研修を担当した。昭和 32(1957)年には、建設研修所が成立したため、機械科が土木研究所から建設研修所に移管された。研究室は昭和 35(1960)年に千葉支所に移転するまで存続した。建設研修所が建設大学校になると沼津分校として機械施工の研修に使用され、昭和 48(1973)年に廃止されている[註 24]。

26 千坪の公有地の一部が沼津工事事務所に所属替えとなり、事務所がここに移転した。残りは、中部財務局を通じて清水町に売却され、町営施設が建設された。

写真 4-6-2, 3 左：土木研究所沼津支所の本館[*95]　右：建設大学校沼津分校の本館[*98]

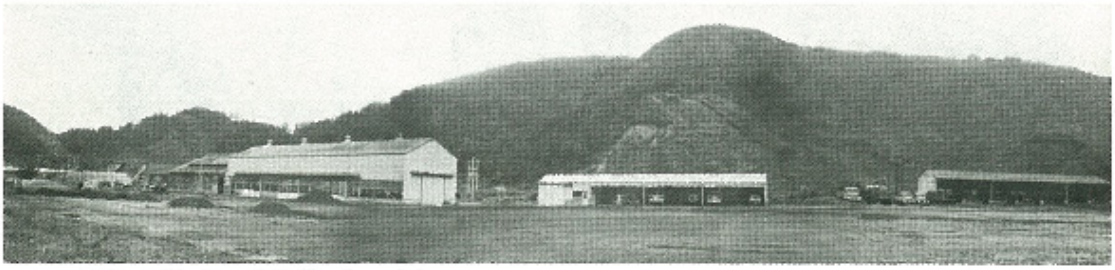


写真 4-6-4 建設大学校沼津分校の全景：左から組立実習工場、車両庫、第１車庫[*98]

本館跡付近には 2018 年時点で、清水町営住宅外原団地（RC 造４階建２棟 32 戸）がある。

なお、付近には終戦時、音響研究所（沼津市下香貫木の宮 888）[註 25]、海軍工廠（沼津市神田町）などが存在しており、後者には工員養成所が附属していた。沼津市内である。

この沼津市郊外にあたる駿東郡清水町徳倉の「海軍施設本部野外実験場」は、建築研究所の設立に際して候補地の一つとして昭和 20 年９月に竹山が沼津を探訪した際の記録に残されている場所「沼津にある海軍施設部の研究所」である可能性が高いと考える[註 26]。

⑤百人町における施設整備

前述のように、百人町に戦後の研究所が成立した当時の建物は、関東大震災後の復興期大正 15～昭和２年の、陸軍科学研究所の時代の既存建物を転用したものであった。

戦後の建設省建築研究所時代の建築年を有する最初の建物は昭和 29 年の実大強度試験室（図面番号 23）であり、戦前建物の最後である昭和２年以降それまでの間の建築年を有する建物は実測図には含まれていない。この間（昭和 3～28 年）は営繕工事としての本格的な施設整備はなかったが、昭和 20 年に防空壕の支保工の廃材を利用した宿舎の整備[註 4]や、昭和 24 年の沼津からの木工場や宿舎の移築[註 23]といった臨機応変的な施設確保が行われていた。

戦後の建物の建設時期は昭和 29～50 年であるが、昭和 44(1969)年までの実験棟などの施設整備は年報に記録されている。最後の整備は、本館と国際地震工学部のペントハウス増築である。またこの年、奥多摩に地震観測所が開設されている。

昭和 45(1970)年以降は、この地区での大きな施設整備は年報に記載されていない。一方、この年の曝露試験場を嚆矢として移転予定先の筑波郡大穂町に施設が整備された。但し、実験装置等は、筑波への搬送を前提として引き続き百人町地区で整備されていた。

但し建物の実測図には、年報に記載されない建物として、昭和 50(1975)年３月に完成した倉庫など数棟が記載されている（表 4-6-1 の図面 46, 47, 48, 49, 51）。また、建築年は未詳の工作物として、門、塀、ストロングルームなどの図面が残されている。

昭和 23 年８月 14 日　建築研究所分課規定の第九条では技術員養成所、第十条では大阪市及び札幌市に支所を置く、とされている（文献：５０年、出典・建設大臣官房広報課編集：「建設省要覧」昭和 24 年）。大阪支所は、日本建築総合試験所、札幌支所は、北海道々立コンクリート・ブロック指導所の形で実現した[註 27]。

昭和 47(1972)年には敷地外の奥多摩に地震観測所の観測機器が整備された[註 28]。

図 4-6-7 奥多摩の地震観測実習所の位置図*68(p. 61)

⑥筑波への移転

昭和 45(1970)年に、茨城県筑波郡大穂町立原に暴露試験場が開設された。昭和 55 年までに主要な施設の整備を終え、移転を完了した。この経緯は移転時の責任者であった棚橋氏の特別寄稿に詳しい。

この時に整備された建物の多くは現在まで継続している。移転直後の記録写真と比較すると、樹木が大きく成長している。移転後に、展示館、画像情報棟、新館等が整備された。

平成 13(2001)年 1 月に、省庁再編に伴い国土交通省建築研究所と改名され、同年４月に大半が独立行政法人建築研究所となり、一部は新設された国土交通省国土技術政策総合研究所の母体となった。

平成 28(2016)年４月に、独立行政法人建築研究所が、国立研究開発法人建築研究所となった。

（３）移転直前の百人町の敷地の復原

移転後に、旧施設は全て解体除却されたため、建物遺構から当時の状態を復原すること

はできない。

空間構成に関して以下の図形資料を利用した。

①航空写真地図帳、住宅地図

国総研の住宅都市資料室には、1960~2000 年代の、紙地図のいわゆる住宅地図、即ち建物外形に居住者の氏名を記入した形式の地図が所蔵されている。2003 年以降は、ゼンリンを出版元とする地図であるが、それ以前のものにはいくつかの変遷がある。

a.住宅協会[註 29]発行、全住宅精密地図帳（1963,65）

b.公共施設地図航空株式会社発行、全住宅案内地図帳(1967,70)

c.同社発行、全航空住宅地図帳（1972）

d.同社発行、航空住宅地図帳(1975～8)

e.ゼンリン発行、住宅地図(2003～)

②所史掲載写真

旧建設省建築研究所の場合、昭和 21(1946)年を創立として、20周年、30 周年、40 周年、50 周年、60 周年に機関史が出版されている。これに加えて 30 周年には、別冊として写真集が編年体で作成されている。50 周年には、別冊として年表形式の資料が作成されており、欄外に小さな写真が掲載されている。

同研究所内に設置された国際地震工学部に関しては、昭和 47(1972)年に 10 年史が発行されており、建物の写真が掲載されている。

③実測記録

移転前に実測図が作成されている。縮尺は 1:100（小さい建物）または 1:200（大きな建物）で、各階平面図と立面図２面が、A2 サイズのトレーシングペーパーに鉛筆で描かれており、これの第二原図と白焼き２部が、現在の建築研究所情報技術課の図面保管庫に保管されている。これに加えて、1:600 の全体配置図が作成されている。

図面は、当時の企画室に在職した益田氏（九州在住）が作成したとの証言が得られた。表 4-6-1 に一覧を示す。移転当時の実測図の建物名称に付された番号は、概ね建築年の順になっている。但し、増築の場合には、同じ図面に異なる建物番号が併記されている。また、既に取り壊された建物が欠番となっているようである。

昭和 47(1972)年３月までに完成していた建物は、移転後の計画案と共に移転時に作成された「昭和 48 年度特定国有財産整備計画要求書」に資産評価表の形で掲載されている。この表に掲載された建物番号と建築年は基本的に実測図の記載と一致している。資産評価表に掲載された建築面積と延床面積を、表 4-6-1 に追記した。

表 4-6-1 実測図一覧

図面番号	名称	建築年月	建築面積 平米	延床面積 平米	入力データ番号(リスト4-6-3)
1	本館	大正 15 (1926)年 3 月	640.09	2289.07	1
2	強度試験場	昭和 2(1927)年 3 月 増築昭和 42(1967)年	905.92	905.92	4
3	別館	昭和 2(1927)年 3 月	630.34	1228.56	5
4	金工場	昭和 2(1927)年 7 月	235.20	235.20	

5	鍛工場	昭和 2(1927)年 7 月	137.45	137.45	
6	火災防火実験室	昭和 42(1967)年 3 月	486.85	486.85	１１
8	施工実験室（３の増築）	昭和 31(1956)年 12 月	282.84	508.99	5
9	会議室	昭和 30(1955)年 2 月	77.75	77.75	
１０	コンクリート養生室	昭和 37(1962)年 12 月	60.09	60.09	
１１	車庫	昭和 2(1927)年 3 月	140.82	140.82	
１２	倉庫	昭和 2(1927)年 10 月	17.85	17.85	
１４	電気熔接室	昭和 2(1927)年	21.81	21.81	
１５	油庫	昭和 2(1927)年 7 月	16.36	16.36	
１７	木工場	昭和 31(1956)年 12 月	243.73	243.73	
１８	アイソトープ室	昭和 33(1958)年 3 月	235.63	235.63	7
１９	変電室	昭和 33(1958)年 9 月	21.05	21.05	
２０	設備実験室	昭和 35(1960)年 12 月	381.75	1111.04	6
２３	実大強度試験室	昭和 29(1954)年 4 月	523.90	523.90	3
２４	音響実験室	昭和 35(1960)年 12 月	44.95	92.95	１０
２５ ２６ ５０	渡り廊下	昭和 32(1957)年 9 月 昭和 32(1957)年 9 月 昭和 48(1963)年 3 月	190.38 79.86 (未詳)	190.38 79.86 (未詳)	
２７	音響性能標準試験室	昭和 34(1959)年 12 月	149.38	173.45	8
２８	室内気候実験室	昭和 35(1960)年 1 月	110.18	200.99	9
２９ ３０ ３１	国際地震工学部	昭和 38(1963)年 11 月	431.39	1583.59	直営入力
３２	地震観測室	昭和 40 年(1965)12 月	102.46	102.46	
３３	宿舎	昭和 40(1965)年 3 月	34.67	75.71	１３
３４ ４１ ４２	宿舎	昭和 40(1965)年 11 月 昭和 42(1967)年 2 月	42.74 42.74 42.74	85.48 85.48 85.48	
３６	土圧実験室	昭和 40(1965)年 12 月	235.40	235.40	2
３７	変電室	昭和 41(1966)年 3 月	27.76	27.76	
３８	宿舎	昭和 41(1966)年 2 月	108.67	434.62	１４
３９ ４０	宿舎	昭和 42 年(1967)2 月 昭和 42 年(1967)2 月	42.74 42.74	85.48 85.48	１５
—	倉庫（１・６）	-	-	-	
４３	実大排煙実験室	昭和 42(1967)年 3 月	40.00	175.30	１６
４４	試験室	昭和 44(1969)年 3 月	349.96	822.98	１２
４５	倉庫	昭和 41(1966)年 11 月	14.58	14.58	
４６	倉庫（総務）	昭和 46(1971)年	19.44	19.44	
４７	運転手控室	昭和 46(1971)年 12 月	19.44	19.44	
４８	簡易無響防音室	昭和 47(1972)年 3 月	58.40	58.40	
４９	風洞実験室	昭和 47(1972)年 8 月	-	-	
５１	倉庫（企画）	昭和 50(1975)年 3 月	-	-	

表 4-6-1 には、「宿舎」も含まれている（No. 33, 34, 38, 39, 40, 41, 42）。掲載された国有財産として戦前から継承した建物、戦後に官庁営繕により整備された建物の他に、全体配置図には、実験装置として整備された建物、住宅公団により整備された建物も敷地内に描かれている。「宿舎」として建てられた建物の図面を見ると、後の「簡二」型公営住宅として建てられた建物に類似しているものがあり、実験的な性格がうかがわれる。またリスト外の「実験住宅」として建てられた建物も、実際に宿舎として利用されていたという。

④UNESCO 報告書掲載図

国際地震工学研修（IISEE）は昭和 35(1960) 年に日本国政府と国連の共同事業として東京大学に開講された後、昭和 37(1962)年に建設省がホストとなって建築研究所に国際地震工学部が設置され、百人町の敷地の一部に建設された施設を用いて第 3 回以後の研究コー

スが開講された。

UNDP プログラムの予算を執行した UNESCO が作成した英文報告書が日本政府に対して提出されており*88、その中に建物の各階平面図、４方位の立面図と写真が掲載されている。この写真と図面が描く建物は３階建の庁舎であった（図 4-6-8）。

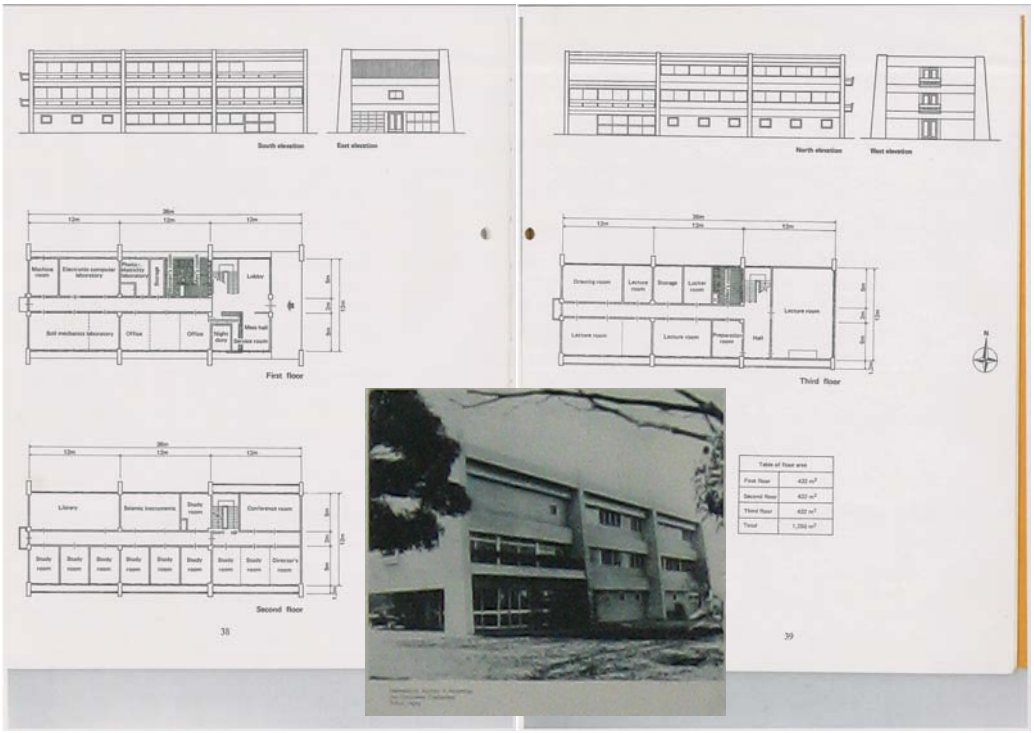


図 4-6-8 報告書に掲載された平面図、立面図と外観写真

この報告書のタイムスタンプは「19__」と未記入のままになっているが、1 階奥の electric computer room は、1962 年に整備された NEAC2101 と考えられ、1967 年に三階に整備された TOSBAK3400 （国地 10 年史*68 に写真掲載）はまだないことからそれ以前の状態を描いており、恐らく 1962 年時点で、第３回の研修が初めてこの庁舎で実施された後、第４回以降(1963～)の研修に先立つ時期の状態を描いていると推定する。

なお、この建物に関しては、1961 年当初の２階建の建物[建研 50 年史年表]と、移転前の 1969 年に[国地１０年あゆみ口絵]の４階ペントハウスが追加された後の建物の写真も残されている。

⑤年報

筑波移転が動き始めた時期にあたる昭和 41(1966)年から年報が出版され、その中に各種の施設整備が記録されている。移転前最後の整備として昭和 45(1970)年には本館と国際地震工学部のペントハウスが増築されている。また、筑波移転先の暴露試験場が整備された。

昭和 45(1970)年からはスタイルが変化し、研究成果に関する報告が中心となった。移転先の施設整備計画がほぼ固まり、移転後の施設を活用した具体的な研究計画の検討に関心が移ったことが考えられる。年報は以後現在に至るまで継続している。

⑥筑波移転に関する企画室資料

筑波移転先（現在地：つくば市立原１）の施設計画の検討過程で作成された資料である。この中には、少数であるが、移転前の研究施設に関する資料が含まれている。

昭和 47(1972)年に作成された「昭和 48 年度　特定国有財産整備計画要求書」には、当時の施設を構成する建物の一覧が掲載されている。これは、実測図に対応するものである。

（４）旧、建設省建築研究所に関する空間情報のアーカイブ

建設省建築研究所に関して、以下の作業を行った

①基盤地図情報から、関係するエリアの切り出し

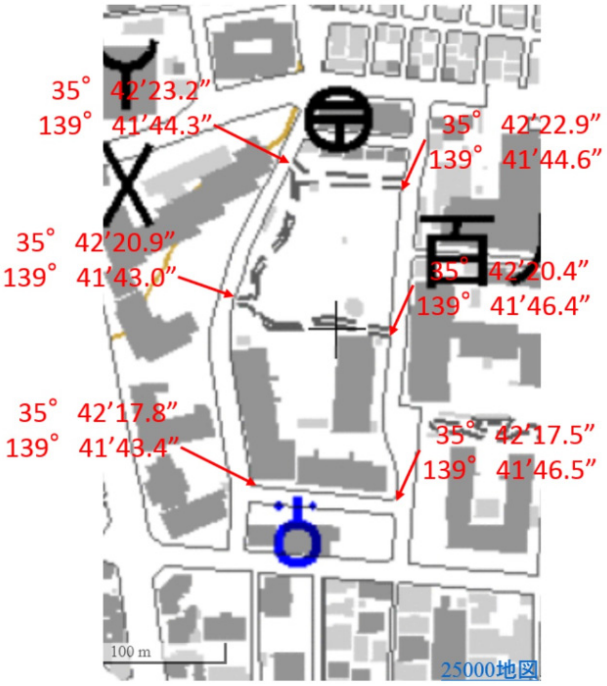


図 4-6-9 百人町付近の現況

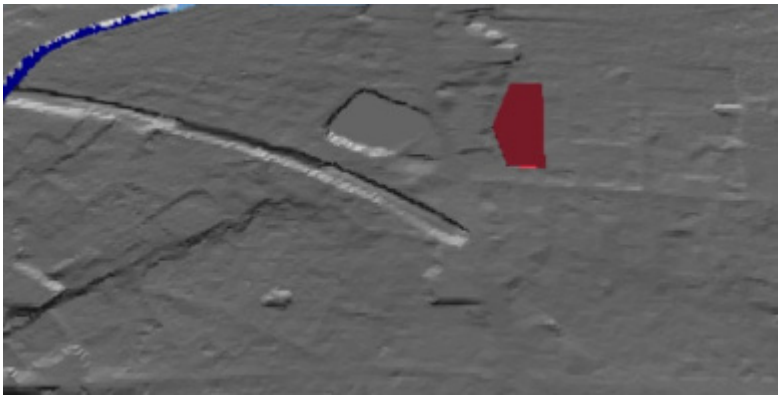


図 4-6-10 百人町付近の地形（赤はほぼ旧建築研究所敷地）

②鳥瞰写真(1966 年)からの敷地全体のモデリング

図 4-6-11 鳥瞰図の位置と建物構成

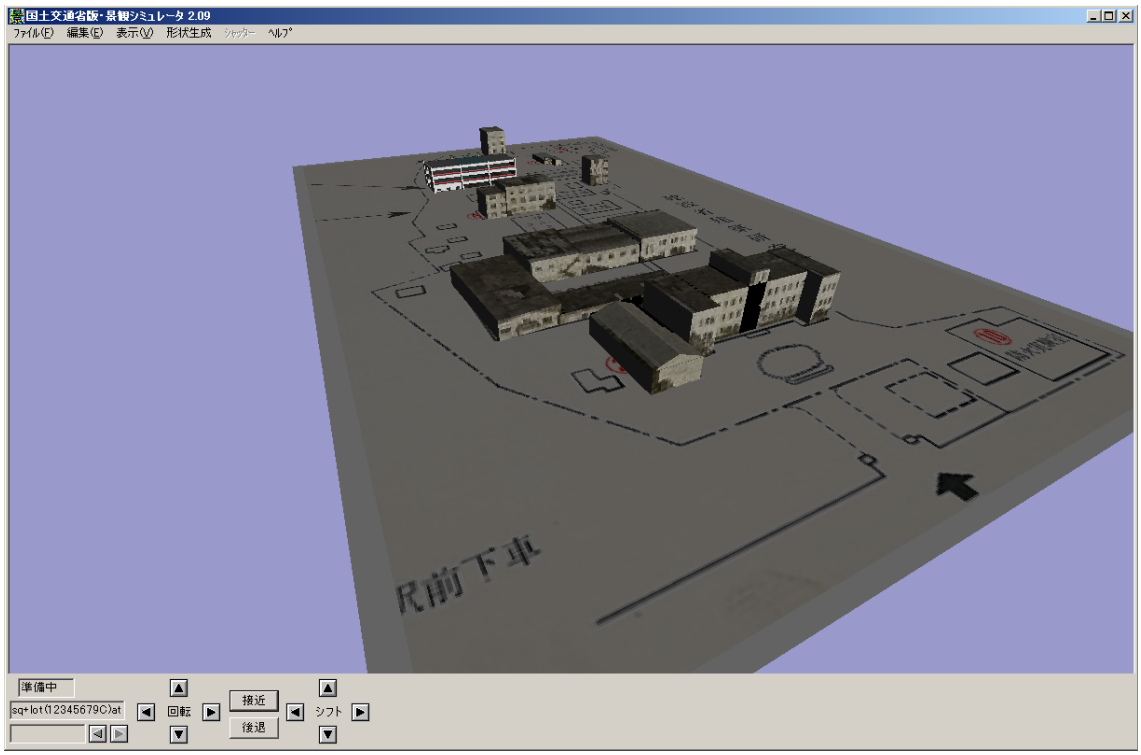


図 4-6-12 敷地全体（主に 1966 年鳥瞰写真からのモデリング）

③本館の地上写真からのモデリング

本館に関しては、最も多くの写真が残されているが、印刷されたものは全てモノクロである。いくつかの写真からリアル・モデラーを用いてモデリングしたデータを図 4-6-10～13 に示す。

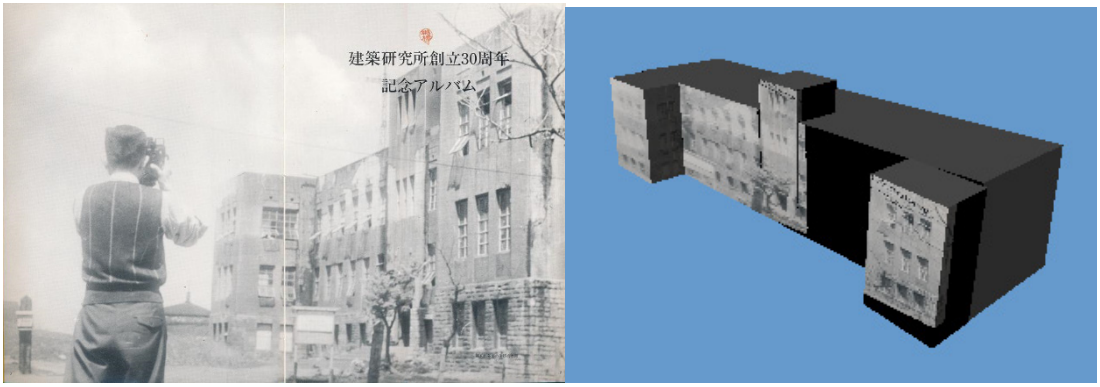


図 4-6-13　1946 年の本館 (p1s 全体.geo 前のロータリー、植栽などはまだない)

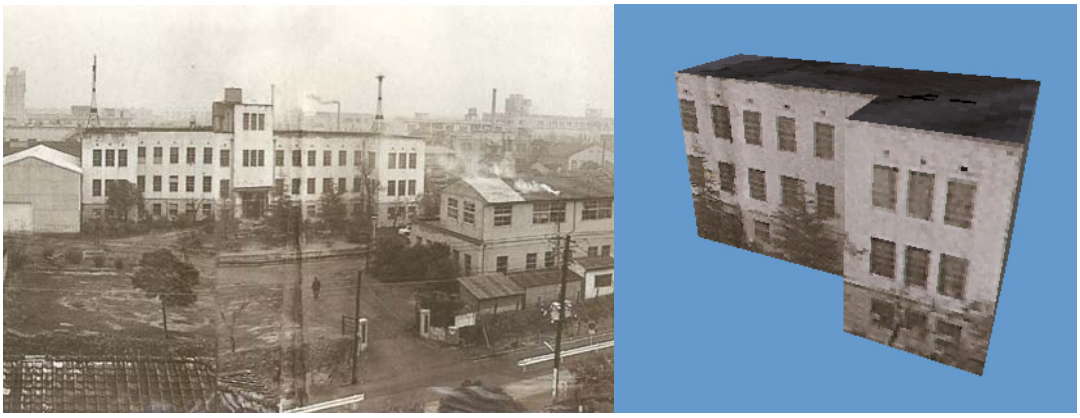


図 4-6-14　1965 年の本館(50-01(cov)本体東半分+東出.geo 、土圧実験室を西側に増築した後)

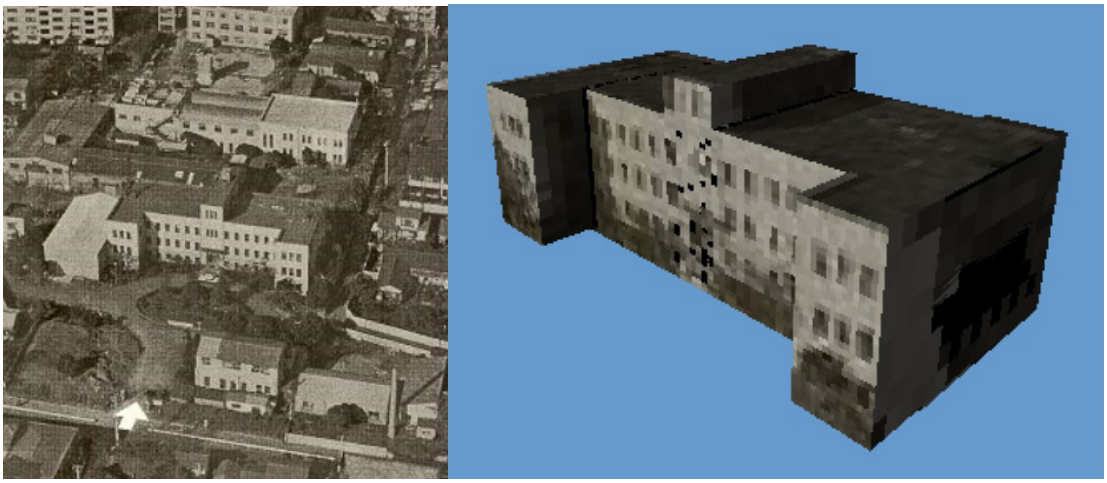


図 4-6-15　1966 年鳥瞰写真からモデリングした本館(33s.jpg,鳥瞰.geo)

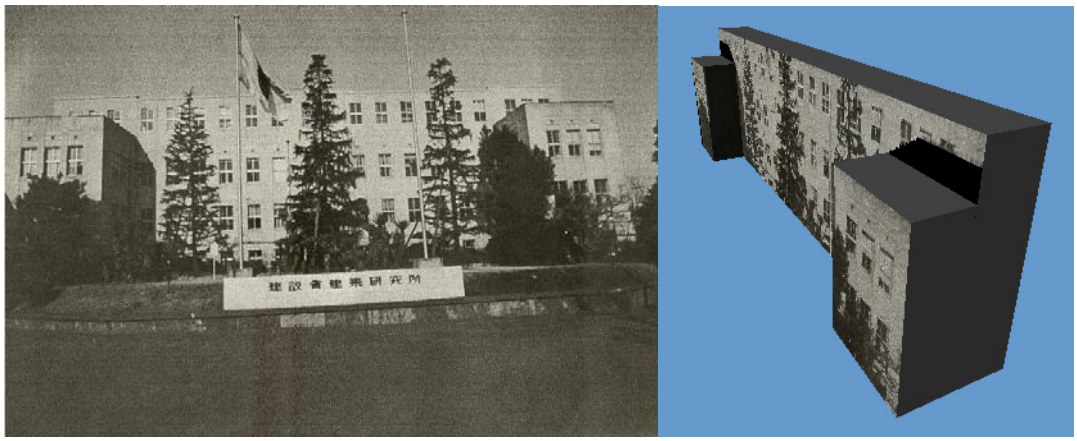


図 4-6-16　1970 年ころ 4 階を追加した後の本館(34s 全体.geo)

④国際地震工学部の報告書掲載図からのモデリング

図 4-6-17　国際地震工学部 (traicen1.geo)

UNESCO に提出された英文報告書（上記３④）に平面図と四面の立面図および外観写真が掲載されていた。景観シミュレータのモデリング機能を用いて、図面から三次元データを作成した。この建物には当時最新の大型計算機が設置されていた。

⑤移転前の実測図からの主要建物の CAD 入力（外注, 1502 納品）

残されていた実測図面は、平面図に加えて、２面の立面図が掲載されているが、残る２面については立面図が存在しない。従って、窓や出入り口の平面的な位置は判明するが、それらの高さに関する寸法（矩計）は、立面図が残されている側の高さ寸法から推定する他ない。一部の建物については記録写真から確認できる場合もあるが一般的ではないため、外注における業務仕様には、記録写真との照合は含めていない。

全部で約 50 棟の記録地面の内、16 棟に関して入力作業を外注し、dxf 形式および IFC 形式により納品された。建物を構成する面は、三角形に分割する形式で記述し、穴あき図形は使用していない。

入力データの性格を示すために、この入力作業に関する一般競争入札および契約に際して使用したデータ入力の仕様を、リスト 4-6-1 に掲載する。

リスト 4-6-1 データ入力の仕様

1　　業務目的

本業務は、過去に除却され現存しない建物の除却前実測結果を記録した紙図面から、当該建物が存在していた位置に立体復原表示を行うための三次元データを作成するものである。

2　　業務構成

(1)　実測記録の整合性と入力方法の確認
(2)　三次元形状の入力
(3)　データファイルの作成
(4)　検索用画像の作成
(5)　データ形式の解説資料作成
(6)　報告書の作成

3　　業務内容

(1)　実測記録の整合性と入力方法の確認

建物の解体除却に伴って行われた実測結果は、平面図及び立面図として記録されている。原図はトレーシングペーパーに、製図板と T 定規を用いて鉛筆を用いて手描きで清書された建築図面であり、図面から各部寸法を計測することが可能である。

平面図の縮尺は 1:100 または 1:200 であり、各階に関して作成されている。

立面図も平面図と同一の縮尺であり、各建物につき２面だけが作成されている。

入力作業を行う建物を別表１に一覧し、図面の様態を示すために縮小した画像ファイルを別添１に示す。

実際に入力作業に使用する図面は、原図から等倍率で光学的に複写した紙図面を、契約後に貸与する。図面間の不整合のある部分、図面だけからは判読できない部分の入力方法ないし省略方法等については随意とするが、受注者からの質問に対しては発注者が回答し、必要があれば担当者間での打ち合わせの上で決定する。

(2)　三次元形状の入力

随意のプログラムを使用して、入力作業を行う。特に支障のない場合には、以下の原則に従う。

①座標軸は、平面図の右方向を第１軸または X 軸、平面図の上方向を第２軸または Y 軸、立面図の上方向を第３軸または Z 軸とする。

②座標値は、m（メートル）を単位とし、0.001m以上を有意とする。

③座標の原点は、建物の平面図における左下端の通り芯の GL レベルとする。

玄関など、張り出した部分がある場合には、座標値が負となっても構わない。

④建物およびこれに付属する小庇、霧除け、タラップ、鉄骨階段、ダクト、煙突等の形状を、面の集合体として表現する。

⑤柱、梁、壁などは、それぞれの単位（意味あるまとまり）として表現しても構わない。

⑥外壁等に曲面部分がある建物の部分は、平面分割を行い表現する。分割数は円周を３２分割する程度とする。円弧、楕円等でパラメトリックに表現しても構わない。

⑦平面図に、寸法記入がある場合には、その寸法に従う。寸法記入がない部分の寸法は、図面から計測する。寸法線、寸法値を形状ないし属性として入力する必要はない。

⑧床に関しては、高さに関する情報しか得られないため、記入された高さの、厚さの無い平面として表現する。

⑨階段は、平面図、立面図から蹴上、踏面、幅を求め、垂直と水平の長方形により構成する。外階段などで、立体形状が判明する場合には、立体として作成する。格子状の手摺は、面（例えば平行四辺形）により近似する。踊り場の高さは、図面から判読される段数と各階の階高から補間計算する。

⑩外壁は、図から計測される厚さを有する立体として表現する。コンクリートパネル、ブロック、外壁石材等の目地は省略して構わない。また、外壁の仕上げが変化する境界線についても、省略して構わない。

⑪窓、入口などの壁の開口部は、外壁の孔として表現し、建具は省略する。開口部の垂直寸法は立面図から判読し、立面図が無い面に関しては、窓か入口かを判定した上で、立面図が存在する面から計測される寸法を適用する。開口部の上に庇がある場合、立面図２面から位置と寸法が判明するものについて入力する。

⑫建物の外部から見えない、建物内部の界壁は省略して構わない。

⑬屋根は、立面図から判読した形状で作成する。陸屋根等、立面図に表れない屋根に関しては、建物の高さと同一高さの水平面として作成する。

⑭面の色彩に関する情報は無いため、省略して構わない。色彩の定義が不可避である場合には、デフォルト値として RGBα値を(0.8, 0.8, 0.8, 1.0)に設定する。

⑮同一形状を有する柱、梁、標準階等を別ファイルとして、あるいは同一ファイル内の部品定義として記述できる入力プログラムの場合には、予め作成した部品を組み合わせる方法で建物を表現しても構わない。

⑯柱、梁、壁等の部材が接合する部分に、隙間があってはならない。

(3)　データファイルの作成

作成した建物別の三次元データを保存するデータ形式は任意とする。

ファイル名称は、建物リストの番号に、データ形式に対応した任意の拡張子を付したものとする。

なお、このデータファイルは、長期保存及び利活用を目的として作成するため、最終成果に関係しない作業上のコメントや仮のデータ等は除去されていることが望ましい。また、データ作成者等が注記できるファイル形式

を用いる場合には、受注者の名称等が記録されていることが望ましい。

(4)　検索用画像の作成

入力したデータを、代表的な視点から透視図またはアイソメトリック表示した画像ファイルを、各建物について最低２の異なる視点から作成する。画像のサイズは、256×256 ピクセル程度とし、保存形式は BMP または PNG 形式とする。

(5)　データ形式の解説資料作成

保存データを入力するプログラムを作成するために十分な情報を含む、データ形式に関する資料を作成する。この資料は、出典を明記した既存の資料であっても構わない。また、コンバータ等に入力するためのプログラム（ソースコード）として表現されていてもよい。

この資料は、入力されたデータに使用されているフォーマット（構文やキーワード）の範囲を含んでいれば十分であり、当該データ形式の全ての定義・仕様を網羅している必要はない。

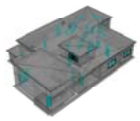
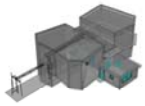
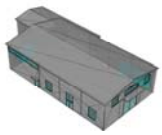
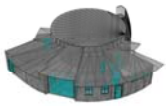
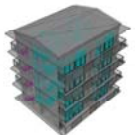
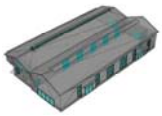
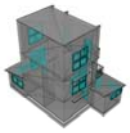
データ形式及び解説資料に関して既存の知的所有権が存在する場合には、解説資料の中に明記する。

上記の長期保存や利活用に際しては、この解説資料に基づいて発注者が業務完了後に作成するメタファイル（データ形式定義ファイル）を上記データファイルに添付する。

リスト 4-6-2 三次元データとして入力した主要建物一覧

番号	名称	建築年次	データサイズ	面の数
1	本館	1926	16,627 KB	172,073
2	土圧試験室	1965	671 KB	4,305
3	実大強度試験室	1954	1,509 KB	9,719
4	強度試験場	1927,1967	1,841 KB	11,736
5	３別館８施工実験室	1927,1956	15,805 KB	100,207
6	設備実験室	1960	6,902 KB	42,642
7	ＲＩ棟	1958	3,181 KB	20,451
8	音響標準性能試験室	1959	857 KB	5,644
9	室内気候実験室	1960	2,369 KB	14,940
１０	音響実験室	1960	1,404 KB	8,863
１１	火災防火実験室	1967	4,126 KB	25,492
１２	試験室	1969	5,205 KB	32,598
１３	宿舎	1965	2,672 KB	16,763
１４	宿舎	1966	5,454 KB	33,872
１５	宿舎	1967	1,866 KB	11,784
１６	実大排煙実験室	1967	1,811 KB	11,496

リスト 4-6-3 アーカイブデータ一覧

ファイル名	イメージ画像	ファイル名	イメージ画像	ファイル名	イメージ画像
01_本館棟.dwg 01_本館棟.dxf 01_本館棟.ifc 01_本館棟_01.png 01_本館棟_02.png		07_アイソトープ室.dwg 07_アイソトープ室.dxf 07_アイソトープ室.ifc 07_アイソトープ室_01.png 07_アイソトープ室_02.png		12_試験室.dwg 12_試験室.dxf 12_試験室.ifc 12_試験室_01.png 12_試験室_02.png	
02_土圧実験室.dwg 02_土圧実験室.dxf 02_土圧実験室.ifc 02_土圧実験室_01.png 02_土圧実験室_02.png		08_音響性能標準試験室.dwg 08_音響性能標準試験室.dxf 08_音響性能標準試験室.ifc 08_音響性能標準試験室_01.png 08_音響性能標準試験室_02.png		13_宿舎.dwg 13_宿舎.dxf 13_宿舎.ifc 13_宿舎_01.png 13_宿舎_02.png	
03_実大強度試験室.dwg 03_実大強度試験室.dxf 03_実大強度試験室.ifc 03_実大強度試験室_01.png 03_実大強度試験室_02.png		09_室内気候実験室.dwg 09_室内気候実験室.dxf 09_室内気候実験室.ifc 09_室内気候実験室_01.png 09_室内気候実験室_02.png		14_宿舎.dwg 14_宿舎.dxf 14_宿舎.ifc 14_宿舎_01.png 14_宿舎_02.png	
04_強度試験場.dwg 04_強度試験場.dxf 04_強度試験場.ifc 04_強度試験場_01.png 04_強度試験場_02.png		10_音響実験室.dwg 10_音響実験室.dxf 10_音響実験室.ifc 10_音響実験室_01.png 10_音響実験室_02.png		15_宿舎.dwg 15_宿舎.dxf 15_宿舎.ifc 15_宿舎_01.png 15_宿舎_02.png	
05_別館8施工実験室.dwg 05_別館8施工実験室.dxf 05_別館8施工実験室.ifc 05_別館8施工実験室_01.png 05_別館8施工実験室_02.png		11_火災防火実験室.dwg 11_火災防火実験室.dxf 11_火災防火実験室.ifc 11_火災防火実験室_01.png 11_火災防火実験室_02.png		16_実大排煙実験室.dwg 16_実大排煙実験室.dxf 16_実大排煙実験室.ifc 16_実大排煙実験室_01.png 16_実大排煙実験室_02.png	
06_設備実験室.dwg 06_設備実験室.dxf 06_設備実験室.ifc 06_設備実験室_01.png 06_設備実験室_02.png					

入力された建物を、敷地に配置したものを、図 4-6-18 に示す。

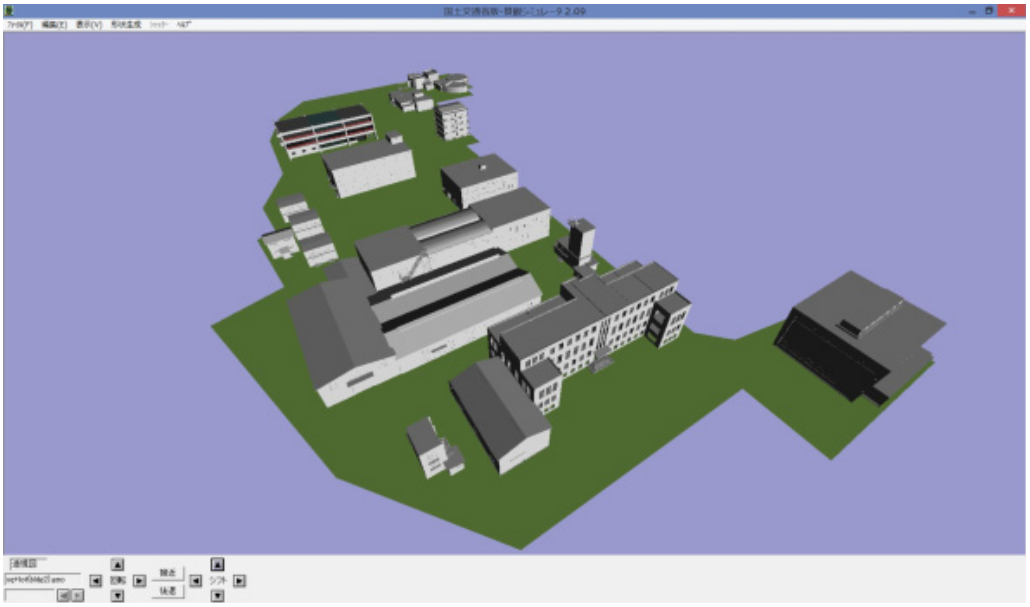


図 4-6-18 敷地全体 (CAD 入力したモデルをそれぞれの位置に配置)

（５）利活用

①タブレットによる表示

鳥瞰写真からモデリングを行ったデータは図形的に単純であるため、軽く表示することができる。景観シミュレータでモデリングを行い保存した LSS-G 形式のデータに、これを解読するためのメタファイルを添付してタブレットに仕込んだ。

実測図面から CAD ソフトを用いてモデリングを行い IFC 形式で保存したデータに関しては、IFC 形式のファイルを解読するメタファイルを添付した。

写真 4-6-1 タブレットによる表示・閲覧風景

② 3D プリンタによる出力

3D プリンタに出力するための入力データとしては、STL 形式が広く用いられている。CAD 入力したデータは、三角形分割により作成されているため、STL 形式に変換するために壁面などの再分割を行う必要はないが、機械的に変換したデータには、以下の問題点があることが判明したため、修正を行う必要が生じた。

a. 面の向き

三角形の面の表裏が不揃いであり、外壁や屋根面などの外側が表側とはなっていない面の部分が混在していた。これは、図面から形状を手入力する際に使用した CAD ソフトウェアにおいて、ソリッドを構成する面の表裏に関する明示的な関心が払われていない（どちらでもよいとして処理している）ためと推定される。

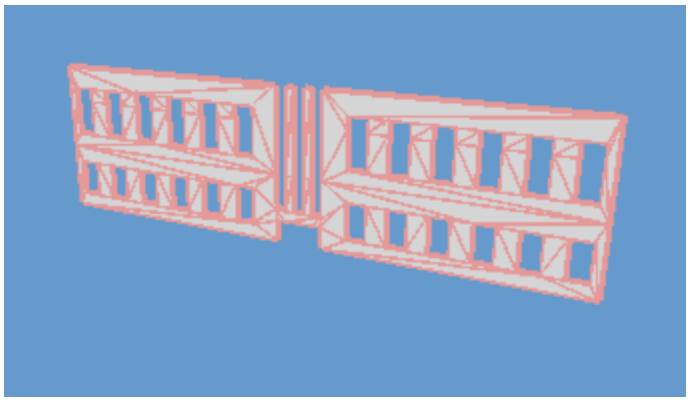


図 4-6-19 三角形の分割状況

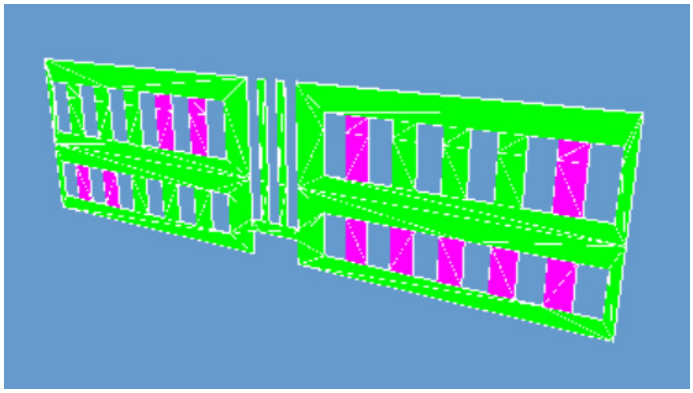


図 4-6-20 修正前の壁面（緑：正、紫：誤）

（カラー表示は、付録 DVD-ROM の pdf ファイルを参照）

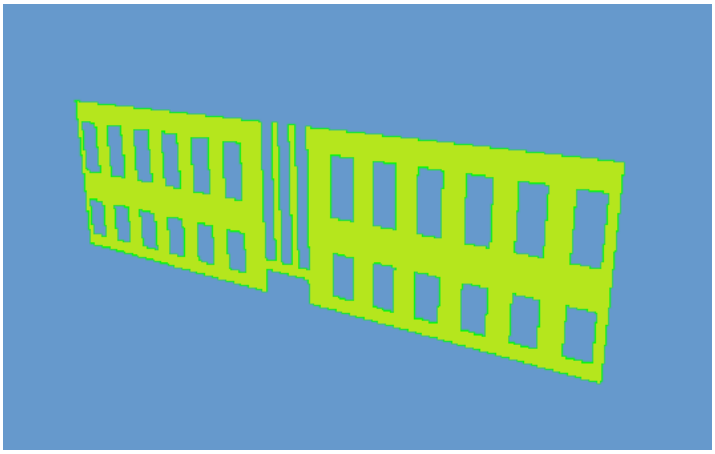


図 4-6-21 修正後の壁面

b. 面の重複

同じ位置・形状の複数の面が重複して入力されている。これはオペレータによる誤操作に起因すると考えられる。単に３D 画面表示を行うだけの目的であれば問題はない（わずかに表示が遅くなる程度）が、３D プリンタ等を駆動しようとすると、重複部分が解釈不能なデータとなる。

c. 面群が、立体として閉じていない

これらの修正作業を能率的に進めるために、flow.dll の地形編集機能を増補した。３D プリンタに出力するためのデータの修正作業は、きわめて特殊な仮設機能であり、汎用性は当面あまりないため、画面右下の「上」のエディットボックスにコマンド（文字列）を手入力した上で、プルダウンメニューの[地形処理][ソリッド化]を選択する方法で、各機能を起動する。機能を指定するキーワードは以下の通りである。

リスト 4-6-4 ソリッド化のための修正作業用コマンド一覧(flow.dll)

```
0. HELP キーワードの一覧を表示する
1. DEBUG　データのソリッド整合性を調べる
2. FLOOR　水平面を薄い板（ソリッド）に変換する
3. EDGE ソリッドとして閉じていない開口部の辺を線に変換して色分け表示する
4. PATCH 開口部を塞ぐ面を追加する
5. COLOR 同一平面上にある構成面を向き（表裏）で色分けする
6. PLANE　同一平面上にある三角形タイルを統合して、一つの穴あきポリゴンとする
7. SELECT 条件式で判別された面を、別グループに移動する
```

（６）その他の筑波研究機関の移転跡地

移転機関の移転前の敷地の所在地と範囲に関しては、検討報告書（特別寄稿の文献 23)、各機関の WEB サイトなどから確認を行った。2001 年の独立行政法人化に伴って、移転当時の研究機関が再編されたり名称変更されたり、現在の各機関の公式 WEB サイトから公開されている各機関の沿革に関する情報は限定されている。しかしながら、各地域の地方史における土地利用の変化に関する情報を公開している WEB サイトが公開されており、容易に検索ができるようになった。

資料収集は継続しており、新たに入手できた資料は上記サイトに増補している。

表 4-6-2 筑波移転機関跡地一覧

No.	画像名	旧研究機関	面積ha	所在地	住宅地図	移転後名	現在住所	住宅地図	跡地
1	01o65 01n02	東京教育大学 光学研究所	2	新宿区百人町4丁目	S40:1218-1288 ,1219-1289	筑波大学	新宿区百人町3丁目	2002:20-21,28-29	高住、公園
2	同上	建築研究所	2.1	新宿区百人町4丁目	S40:1218-1288 ,1219-1289	不変	新宿区百人町3丁目	2002:20-21,28-29	高住、公園
3	03o63 03n02	東京教育大学	6.4	文京区大塚窪	S38:1565-1566	筑波大学	文京区大塚3丁目	2002:10-11,23-24	筑大附小、公園 文京スポーツC
4	04o63 04n02	林業試験場	12.1	目黒区小山台1，2	S38:1307-1377	不変	目黒区下目黒 品川区小山台	2002:38-39,44	公園
5	05o63 05n02	東京教育大学 農学部	6.8	目黒区駒場	S38:1089-1159 1090-1160	筑波大学	目黒区駒場	2002:3-4	区体、入試C、公
6	06o63 06n02	東京工業試験所 目黒分室(第六部)	3.1	目黒区中目黒1丁目	S38:1374	？	目黒区三田	2002:18-23	清掃工場
7	07o63 07n02	国土地理院	1.3	目黒区上目黒7丁目	S38:1162	不変	目黒区東山	2002:8,9	公幼老介
8	08o63 08n01	東京教育大学 祖師谷農場	8.9	世田谷区祖師谷1丁目	S38:460,461	筑波大学	世田谷区上祖師谷4丁目	2001:43-44 ,54-55	公園
9	09o63 09n02	東京教育大学 体育学部	4.3	渋谷区西原1丁目	S38:1155,1156	筑波大学	渋谷区西原1丁目	2002:18-19	渋谷スポーツセンター
10	10o63 10n02	東京工業試験所	3.1	渋谷区本町1丁目	S38:1154,1155	化学技術研究所	渋谷区本町1丁目	2002:6,12	新国立劇場
11	11o63 11n02	蚕糸試験場	4.2	杉並区高円寺2丁目	S38:941	不変	杉並区高円寺	2002:57-58,67-68	蚕糸の森公園
12	12o63 12n02	機械技術研究所	4.6	杉並区矢頭町	S38:584	不変	杉並区井草	20025;	運、区立井草森公園
13	13o63 13n02	気象研究所	1.9	杉並区馬橋4丁目	S39:798-868 ,799-869	不変	杉並区高円寺北4丁目	2002:30,38	公、気象庁住宅
14	14o63 14n02	農業技術研究所 獣疫研究室	3.1	北区西ケ原2丁目	S38:1630,1631	農業技術研究所	北区西ケ原	2002:60,66	公、体、政策研
15	15o63 15n02	公害資源研究所 浮間分室	11.1	北区浮間4丁目	S38:1200,1201	不変	北区浮間4丁目	2002:5,11	下水道局処理場
16	16o63 16n01	東京教育大学寄宿舎	1.8	板橋区常盤台4丁目	S36:1136,1137	筑波大学	板橋区常盤台4丁目	2001:74-75	公園
17	17o63 17n01	計量研究所	1.5	板橋区板橋町6丁目	S38:1418,1419	不変	板橋区加賀1丁目	2001:91	体、公園
18	18n02	公害資源研究所	4.3	川口市	(報告書)	不変	川口市川口3丁目	2002:西部86	再開発
		(以上は跡地利用報告書掲載)							

19	19o68 19n02	蚕糸試験場日野桑園		日野市日野	S43:7314	蚕糸試験場	日野市本町6丁目	日野市2002:9	スポ
20	20n02	農事試験場畑作部		埼玉県北本市			埼玉県北本市荒井5-200	2002北本市:59-60 ,67-68,74	自然学習C
21	21o68 21n02	家畜衛生試験場		小平市	S47:7966-7	不変	小平市上水本町6丁目	2002小平市:74-75 ,80	国分寺市営けやき公園、都立小平南高、国分寺市民ス、周辺都市整備
22	22n02	果樹試験場		平塚市	-	不変	平塚市大原	2002平塚(東)147-8 ,154-5	平塚市総合公園他
23	23o68 23n02	機械技術研究所東村山分室		東村山市富士見町5	S43:7719-20 ,7789-90	不変	東村山市富士見町5	2002:東村山市48-50,5	都立東村山西高、
24	24o68 24n02	工業技術院産田無分室 電子技術総合研究		田無市上向台1059	田無市 S43:8285-8355 ,8286-8356	不変	西東京市向台町5丁目	西東京市2002:49	調整池、運動場、高校、公園 集合住宅(元宿
25	25n02	土木研究所千葉支所		千葉市稲毛区穴川		不変	千葉市稲毛区穴川4	2002千葉市稲毛区 55-56,62-63	放射線医学研究所他
26		土木研究所鹿島試験所		茨城県神栖町		不変	神栖市		
27	27o68 r調査	東京教育大学保谷農場・寮・グラウンド		東京都保谷市東町1丁目	東村山市S43: 7719-20 ,7789-90	筑波大学	西東京市東町1丁目	2002西東京19	文理台公園
28	28n02	東京教育大学板戸農場				筑波大学	鶴ヶ島市千代田1丁目	2002鶴ヶ島市3,10 ,18-19	附属坂戸高校農場、ワカバウォー
		(以上は、高山資料)							
29	29o63 29n02	土木研究所駒込本所		文京区上富士前町26	S38:1633		文京区本駒込2丁目	2002年:6	区立昭和小
30	30o63 30n02	土木研究所赤羽支所		北区志茂五丁目	S38:1481,1482		北区志茂5丁目	2002年:15-16	荒川下流工事事務所
31	31n02	畜産試験場		千葉市			千葉市中央区青葉町	2002:千葉市(中央区)34-35 ,A2-43,A9-51	千葉県立青葉の森公園

移転機関が存在していた頃の住宅地図で S38 1418 と記したものは昭和 38 年の地図の図郭番号 1418 である。収録する分冊は各所在地に従う（例えば「杉並区」）。図郭番号は、最

古の 1963 年から最後の 1988 年まで同じであるが、住居表示などは変化している。

移転後の跡地の状態を示す住宅地図は、例えば No.6（工業試験所）について「2002 18-19」と記したものは、目黒区の 2002 年版の住宅地図の 18-19 頁に掲載されていることを示している。

［補注］ 根拠となる引用（全文）を「*イタリック*」で再掲、出典を略記し詳細は[文献番号]参照

１．新宿百人町に存在した研究機関等の移転時の名称とその履歴等

(1)建設省建築研究所

1946 年戦災復興院技術研究所として設立、建設院第二技術研究所、建設省建築研究所

住所は新宿区百人町 4-394、1971 年 6 月 1 日から〒160 百人町 3-28-8　電話 361-4151

本館建物は、日本近代建築総覧 No.16387, p.122　*94(以下、総覧と略)

(2)東京教育大学光学研究所

1949 年東京文理科大学大久保分室として設立され、1949 年に東京教育大学光学研究所となった。本館は 1921 年。跡地には現在、社会保険中央総合病院がある。

総覧 No.16413, p.123 住所は新宿百人町 3-22-7

なお、建築研究所の開設地を探していた竹山が、昭和 20 年 9 月 23～24 日に陸軍第七研究所の最後の野村所長と面会した際に、「*この時同室に小柄な温厚そうな中年の方がニヤニヤしながら我々の話を聞いて居られた。跡で解ったことだがこれが文理大*(後の東京教育大学)*の藤岡由夫博士で、やはり七研を交渉に来られ*」との竹山謙三郎の回想録にある。また、同回想録には、*「財務局の担当官は種田事務官という人で「大久保の科研の跡には全部研究施設を集め、戸山カ原を緑地帯とする理想境を造ろう」という我々には全く好都合の抱負を持って居られた*」ともある。*66　(建築研究所 20 年のあゆみ、以下 20 年史と略)

(3)国立科学博物館新宿分館

資源科学研究所は、1941 年に高樹町に設立され、1946 年に百人町に移転、1971 年に国立科学博物館新宿分館となり翌年資料館を残し新築。2012 年に筑波に移転した（つくば市天久保 4-1-1)。百人町に移転当時の建物は昭和２年。総覧 No.16414, p.123 住所は百人町 3-23-2

(4)都立衛生研究所

昭和 24 年に設置され、2003 年 4 月に健康安全研究センターに統合され 2012 年に本館が竣工した。〒169-0073 百人町 3-24-1,2。本館は昭和２年頃。総覧 No.16415, p.123

(5)財団法人・蚕糸科学研究所

1940 年 3 月に淀橋区柏木 3 丁目に設立され、昭和 20 年 10 月に百人町に移転した。〒169-0073 百人町 3-25-1。総覧 No.16416, p.123

(6)呉羽化学東京研究所本館

昭和 10 年代。総覧 No.16417, p.123

なお、上記各本館の建築年は、総覧に基づく「建築研究所５０年」(p.330)の記述による[*73]。

２．航空写真地図帳（図 4-6-1）[*92]

この出版時点ですでに５０年が経過しており、パブリックドメインである。

３．本館建物

戦後から筑波移転 1979 年まで建築研究所の本館として使用されていた建物は、終戦時点の第七陸軍技術研究所の本館を転用した建物であった。

遡ると 1941 年における組織改編で従来の陸軍科学研究所と陸軍技術本部を統合して８の研究所とした。同年６月、陸軍科学研究所第一部の業務の大半を継承して陸軍技術本部第七研究所が設置され、さらに翌 1942 年 10 月に第七陸軍技術研究所となった[沢井 2012：p. 286]。[*105]

前身機関である陸軍科学研究所は、当初の第 1 課と第 2 課からなる組織から、1925 年 5 月 1 日に第一部（理学）、第二部（火薬）、第三部（化学）から成る組織に再編されている。この時期は戦後、建築研究所の本館として使用されていた建物の建築年として記録された大正 15(1926)年３月と同年度であり、竣工時点でこの建物が陸軍科学研究所第一部の建物として使用されていたことを示唆している。

昭和 15(1940)年に第一部に採用された藤井は、*「物理関係の基礎研究所で、後に第７陸軍技術研究所と改名」*と記している[註４]。

関東大震災(1923 年)の後、薬師寺主計が欧米視察より帰国し陸軍震災善後委員会特別委員長に就任している(1923 年)ので、設計に関与したと推定できる。同氏は、1926 年に陸軍を退職して倉敷絹織株式会社の取締役となり、1930 年に竣工した大原美術館を設計した。(https://ja.wikipedia.org/wiki/薬師寺主計)

４．図 4-6-2 の各図の出典について

＊1902,1925,1937 年は、国土地理院が発行した地形図 1:25000

1945 年は、「建築研究所 20 年のあゆみ[*66]」に掲載された図。註５に大きく掲載

1980 年は、跡地利用報告書(1980)[文献 2]掲載図

５．移転までに除却、建替された戦前建物

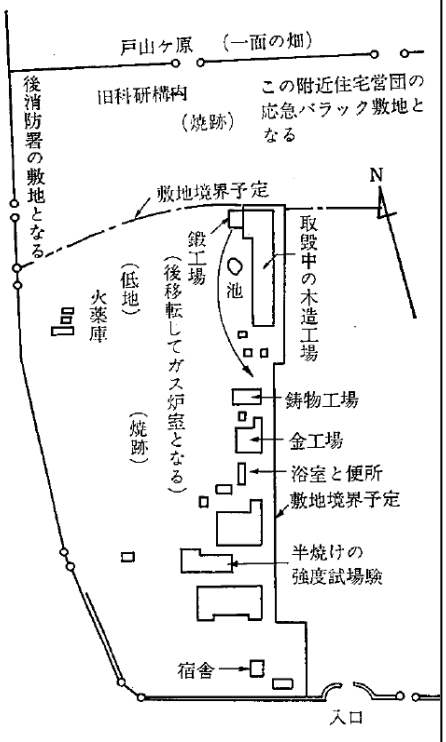
戦前から百人町の研究施設に勤務していた藤井正一によると、昭和 19 年に陸軍の小型の研究施設や計測器類は長野と金沢に疎開していたが、大型の施設のそのまま残されていた。昭和 20 年３月 10 日の空襲で、木造建物は灰燼に帰したが風洞などの大型の施設は残されていた。GHQ により陸軍から継承された残留物は建物を除き処分されたが、戦後に内務省防空研究所や大蔵省管財局から持ち込まれていた物品は properties of BRI として残された。

「私は、昭和 15 年３月、当時の陸軍科学研究所第１部（物理関係の基礎研究所で、後に第７陸軍技術研究所と改名）に研究員として入所以来、終戦まで現在の第４部長室を研究室として、弾丸やロケットの空気力学に関する研究を行っておりました。施設としては現在のコンクリート実験室の位置に風速 100m/s の中型風洞と、強度試験場にマッハ３の風速の出る小型超音速風洞を持っておりました。この陸軍の研究所は昭和 19 年には長野と金沢に分散して疎開しましたので、小型の研究施設や計測器類は、それぞれ分散しましたが、大型

の施設はそのまま残され、私の風洞類も現在の建研の敷地内に残留しました。昭和 20 年 3 月 10 日の空襲で、木造建物は全部灰燼に帰しましたが、鉄筋の建物の大部分は火災をまぬがれ、研究施設も無事でした。」

「昭和 23 年になってから、進駐軍からお達しがあり、陸軍の残留物はすべて処分するというようなことでそのリストを造ることが命ぜられました。何しろ終戦時には陸軍はすべての書類を焼き棄てましたので、何一つ記録もなく、当時の建研内にある残留施設の名称と性能を書き上げるには、現物を一々調べる以外には方法がないのですから大変です。おまけに、防空研究所や大蔵省管財局から持ち込まれた陸軍以外の物品が処分されては困ります。そこでこのような物品にはすべて大急ぎで properties of BRI という表示をし、処分されることを防ぎました。他の物品については、進駐軍からの指示にしたがっていわゆる inventry sheet を作るのですが、２カ月の期限を切って必ず作り上げるようにとの厳命です。（中略）この調査の結果、風洞をはじめ、工場の工作機械類も破棄を命ぜられ、結局陸軍時代のもので建研で利用できたものは、殆ど建物のみということになってしましました。」 藤井正一「陸軍技術研究所から建研へ」（建築研究所３０年のあゆみ，p. 25）[*69]

一方、建築研究所開設地を探していた竹山・久田は、当時のこの場所の様相について、次のように記して説明図も残している。



昭和20年10月現在の敷地状況

「久田君の提案で念のため右手に見える鉄筋コンクリートの荒廃した一群の建物をたづねて見ることとした。この一群は当時の七研であるが、周囲に焼けた木造の建物が多かったために、荒廃の状は著しく、２，３日来の雨に濡れそぼれた外壁は斑に剥げ落ちて、まともに使える建物ともみえなかったのだ。

然し近附いて見ると、外はひどいが中は一応使えそうな鉄筋コンクリート造２棟、焼けては居るが走行クレーンのついた強度試験場１棟、更に奥へ進むと木造の便所と浴室、鉄骨の工場が２棟あって中には工作機械が 10 数台、数人の人達がたき火を囲んで居るのが見えた。この人達が現在の金工場の諸君で、いづれこの間の話がうかがえるものと期待して居る。

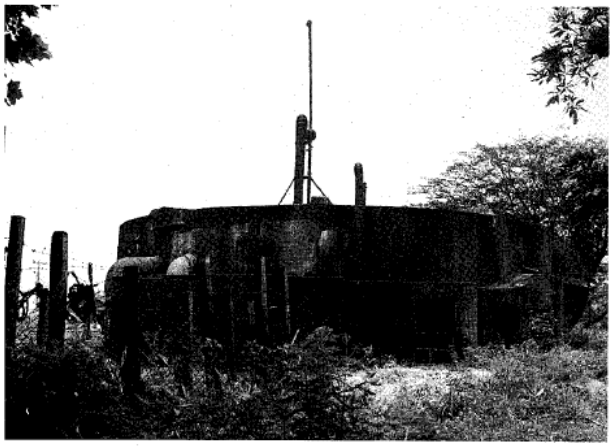
工場の裏は遠く焼け跡の彼方に行けをへだてて現在ガス炉室につかって居る鉄骨造の棟とこれに鍵の手につづいて木造の相当大きな工場が半ば解体され乍ら残って居た。これは現在の野球場の三塁側見物席一体にあるコンクリートの土間の所に建って居たのである。これらの建物の先は再び焼け跡をへだてて戸山カ原につづいて居た。参考にその当時の図を揚げて置こう。（後略）」

［竹山:建研創設時代回想記（20年のあゆみp. 197，回顧録２）[*66]

図 4-6-22 昭和 20 年 10 月の百人町敷地の状況

この 1945 年の回想図には、「取毀中の木造工場」、「鍛工場　移設してガス炉室に」とい

った書き込みがある。また、この図にはないが、30 周年記念アルバムには昭和 24(1969)年に撮影された旧軍時代の爆破井戸の写真が掲載されている(p. 8)。キャプションに記された「現在の音響実験室」は、表 4-6-1 の図面番号 24、昭和 35 年 12 月の建物である。



隣接する第６陸軍技術研究所で化学兵器が研究されていたことから、戦後昭和 31 年から平成 16 年まで環境省により、元関係者の聞き取り調査、跡地の都立衛生研究所等における旧軍関係施設の残留毒物などの調査に加え、毎年周辺地域の地下水汚染調査が行われている。

図 4-6-23 百人町に残されていた旧軍時代の爆破井戸

この、昭和 20(1945)年 8～9 月の状況を撮影した空中写真から戦災復興院が作成した地形図を図に示す。

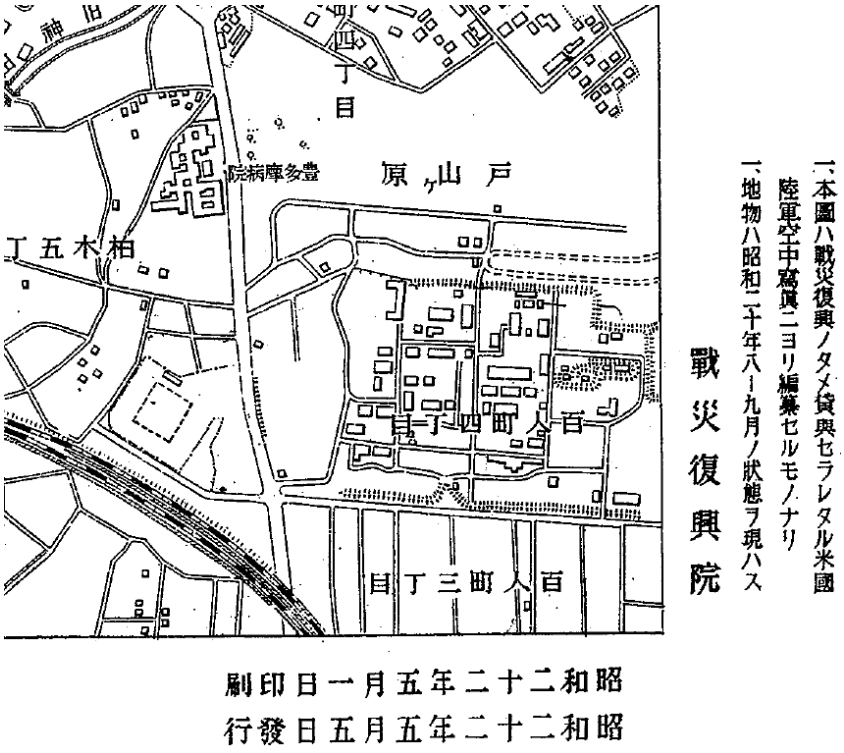
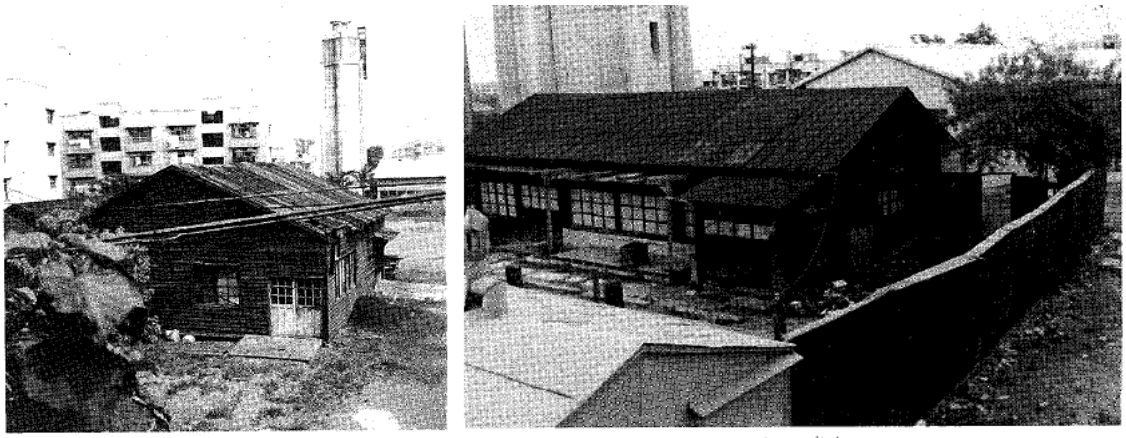


図 4-6-24 百人町付近の戦災復興院地図

「予算もなく、資材も新規に入手することは到底不可能であった。幸い 10 数個の木製ベッドがあったので独身者にはすべてこれを当てることにしたが、困ったのは家族持ちの人たちの処置であった。いろいろ考えた末に山梨の疎開先から大工さんの星、処の両君に一先

ず状況して貰い、吉田老人と３人で構内の防空壕の側壁をはがし、これで現在の財務局の建物（当時６つの小部屋に分かれていた）と本館地階の数室に床を張ることとした。」
と、竹山回想記（20 年のあゆみ）に記されており、防空壕も存在していたことがわかる。

除却前の建物実測図(表 3-6-1)に記されている建物番号には欠番がある。除却され、あるいは建替えられた建物がある。木工場に関しては、20 年史年表の昭和 24(1949)年 8 月に*「沼津より木工場を移設」*とある。30 周年記念アルバムに、「昭和 34 年用途廃止された木工場」の写真があり、このとき移設された建物かも知れない。同じく「昭和 36 年用途廃止された宿舎（篠沢宅)」の写真が残されているが、篠沢氏は沼津から昭和 24 年に建築研究所に合流した職員である。



昭和34年用途廃止された木工場　　昭和36年用途廃止された宿舎（篠沢宅）

図 4-6-25 百人町で廃止された建物

筑波移転前の実測図中には、昭和 31 年 12 月に建築された平屋 243.73 ㎡の木工場がある(表 4-6-1)。用途廃止の前に、新しい木工場が建てられたことを示している。

６．初期の職員の出身機関を示した図

座談会「研究の今昔」[出典／建築研究所 30 年のあゆみ][*69] の中に掲載されている。

なお、建築研究所 50 年[*73]p. 330 には終戦時点における第七陸軍技術研究所幹部名簿（航空本部長隷下、出典：終戦時帝国陸軍全現役将校職務名鑑）が掲載されているが、図 4-6-3 に記載された藤井正一を除き一致する名前はなく、施設だけが継承された。なお継承されたという金工場の職人[例えば手記が 30 年史にある森義夫]の人名は記録されていない。竹山の座談会発言に*「その時七研では所長の野村少将が残務整理で残って居られた。ひどく古風な内法の高い扉の把手を引いて現在の所長室に入り、この研究所を譲り受け度い旨を申し入れた。・・・・七研で交渉の相手をされた方は前記野村所長と総務課長の竹下中佐（現学術会議学術部長)、名前を忘れたがガラさんというアダ名の大尉さんであった。・・」*と記されている[出典／竹山”建研創設時代回想記”20 年のあゆみ], 建築研究所 50 年 p. 326 に再掲。

当時の交渉相手の*「研究所長 少将 野村政彦 長野、中佐 竹下俊雄 技術」*の名前がこの幹部名簿中にある。

７．国会議事堂の現場

「今日の建設省建築研究所は国会議事堂の工事現場から生まれた。すでに大正年代に臨時議院建築局の試験室が議事堂工事場におかれ、主として国産建築材料の試験研究を始めていた。金子堅太郎子爵の建言によって議院はすべて国産材料によってつくられるべきことが決定していたからである。ほんとに下小屋ていどの研究室で研究費も工事費からの支弁という状態であったが、日本産の石材や木材の研究など、学術的にも大きな成果と見られている。古代からの建築学の歴史を見ると、建築家や棟梁個人の叡智の中から、まず学問として分化成立したものが材料の分類学的な学問と、つづいてその力学的研究であった。このことを思い合わせると国会議事堂工事現場に芽を出した研究機関は発生学的に興味のあるものである。

この研究室はその後大蔵省営繕管財局による分課規定による一掛りとして存続し、昭和九年には大手町の大蔵省庁舎に移り、営繕管財局全体の工事用諸材料の試験・検定にあたるようになった。しかし兼任技師一名、専任技手二名ていどのごく小規模なものである。昭和十一年に営繕管財局は本格的な国立建築研究機関の構想を打ち出し、予算案も成立したが、二・二六事件による内閣更迭でついに流産してしまった。昭和五－六年ころからコンクリートの研究を、十四年ころから新しい木構造法の研究を開始している。昭和七年東大卒の竹山謙三郎（後の建築研究所長、現鹿島建設研究所長）が大蔵省に入って、木構造の近代化にとりくんだのもこのころである。昭和十六年にはドイツの木構造計算および施工規格が竹山の手によって紹介され、大張間構造や、いわゆる“新興木構造”の建設・応用にあずかって力があった。

昭和十八年（ママ）にははじめて室制がとられ、萩一郎を室長に人員も強化された。研究室のこうした動きが、昭和九年室戸台風の直後建築学会に設けられた「木構造基準調査委員会」の組織と協力して、できあがったものが「木構造計算基準」である。昭和十九年八月、九月合併の最後の「建築雑誌」に発表されているのも印象的である。明治二十年以来一度も休刊したことのない「建築雑誌」も、今度の戦争では、十九年十月から二十年十月まで休刊を余儀なくされたのである。ともかく、この基準によって木構造もはじめて構造学的な数値計算の対象となり、戦後の発展の土台となった。

終戦後、この研究室は新宿区百人町のもと第七陸軍研究所の施設に入り、二十年十一月には戦災復興院建築研究所（ママ、正しくは戦災復興院官房技術研究所。当時土木試験所はまだ内務省に所属していた。これに先立って 10 月 24 日には「大蔵省営繕局研究所」という看板を掲げたとの竹山発言が３０年史に記録されている）、*二十三年一月には建設院第二技術研究所*（註：第一が土木、第二が建築）*となり、旧内務省防災研究所*（ママ、正しくは防空研究所）*陸海軍の研究者も漸次吸収して建設省発足とともに、その付属機関として「建設省建築研究所」の看板を掲げるにいたったのである。」*[村松 1965，p. 124-6][*102]

８．建築研究所の構想

「昭和 10 年のことだったと思う。大蔵省の営繕管財局（外局）に建築研究所を創設しよう

という話が出て、当時の研究掛長斎藤亀之助技師（故人）などの先輩が中心となって企画し、私共が予算史料の調査を命ぜられ、東京工大の建築材料研究所や商工省東京工業試験所を見学に行ったり試験器メーカーのカタログを調査して予算案を作り上げた。」
（５０年史[*73]p. 323 に再掲　出典／藤田金一郎“創設期の建研・外から見た建研”「建築研究所２０年のあゆみ[*66]」）

９．戦前における建築研究所設立に向けた予算要求等

「昭和 14 年内務省に防空研究所が創設された。この年の予算では建築研究所（営繕管財局）と防空研究所（内務省）とが主計局で競合し、第一次査定で主計局は両者を削った後、復活要求で、時局の緊急性の強い防空研究所が復活したのである。」（建築研究所５０年[*66]p. 323, 出典：藤田金一郎：「創設期の建研・外から見た建研」［建築研究所 20 年のあゆみ[*66]］）

「営繕管財局は、昭和十二年度予算において、建築試験所の設置に関する経費を要求したが、認められるに至らなかった。
注）本経費の要求理由は、「我国土国情に適合する独創的にして且つ経済的建築方策を確立する」ためであった。

なお、昭和十九年度には、「建築工事用資材節約の為 之が代用品の試作及実験、労務節約の為 木工具の機械化並建具各部材の大量生産化に必要なる設備の考案試作、防空施工上未研究の事項に関する調査並研究及木構造の一般的改良、杭の体力増進に関する研究等をなすは時局下緊急を要する」ことを理由として、建築に関する試験研究などに要する経費を要求したが、認められなかった。」
［50 年史[*73]p. 323，出典／大蔵省昭和財政史編集室：「昭和財政史」第８巻 国有財産・営繕 昭和 33 年］

１０．霞が関の大蔵省庁舎４階にあった建築研究室

「竹山と久田が、（疎開先の山梨から）先遣隊として、焼野原の東京に乗り込んだ。ところが、思いもかけない出来ごとが、彼らを待っていたのである。本拠の大蔵省庁舎が、連合軍の命令で接収されることになったのだ。それも、二日間で立ち退けという猛烈な命令である。当時、四階にあった研究室は、机、椅子、書棚、研究機材、研究書類の一さい合財を、窓から中庭に投げ落とすという騒ぎになった。狭い階段は完全に交通マヒ状態である。藤田の記憶によると、研究室の宝ものとして、大切に保管していた国会議事堂の精巧な模型が、このドサクサ騒ぎで完全に行方不明になり、未だに消息がわからないという。

大蔵省は、取りあえず都内各区の国民学校に分散して入ったが、建築研究室はどこにも割り込む余裕がない。・・・」［田中 1985[*103]p. 41～43］

１１．アムスラー社製１００トン試験機

大蔵省建築研究室があった霞ヶ関の大蔵省庁舎の地下には、大正５年にアムスラー社から輸入した 100 トン圧縮試験機ほかが設置されていた。大蔵省庁舎は戦後米軍に接収されたが、機材の一部は戦時中疎開していた。

「戦局の急迫に伴って、まもなく建築研究室は強度試験機の一部を移送して山梨県相興村

の小学校に疎開した。と同時に焼け出された世帯持ちの職員は周りの民家や草堂に、単身者と勤労動員の学生達は近くの禅林・法珠院に寄宿して、結局ここで終戦の日を迎えることとなった。」[文献３⑥;出典菅原肇:実録藤田金一郎先生と私「建築の研究６７号」(1988)]

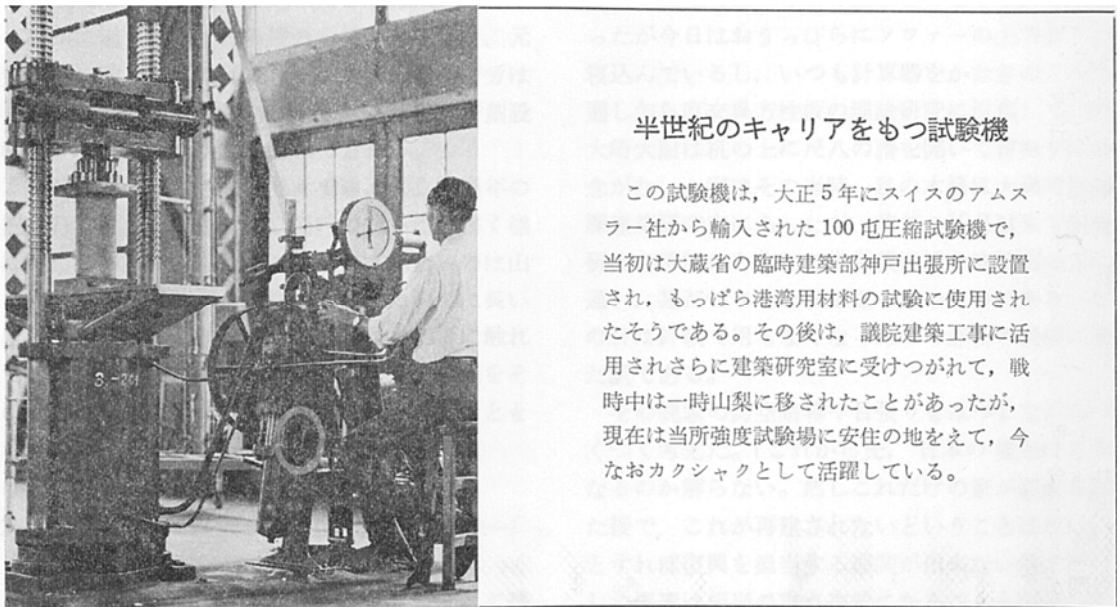


図 4-6-26 20 年史に掲載された試験機の写真

20 年史[*66]p. 195 に試験機の当時の写真が掲載されている(図 4-6-25)。山梨経由で百人町の強度試験場に設置されて稼働していた。

*「疎開して荷解きもされないままの強度試験機は再び東京に送り返された。しかしこれらの試験機は初期の大久保の強度試験場で大いに活躍し、今では筑波の新庁舎に納まって、長閑かな余生を送っている。」*50 年史[*73]p. 328，出典／藤原肇:“実録　藤田金一郎先生と私”「建築の研究」６７号(1988)

写真 4-6-2 つくば市立原の建築研究所展示館における設置状況 (2018 年 7 月 11 日小林撮影)

筑波移転当初（1979 年）は本館玄関ホールに置かれていた。疎開先の相興小学校（現在

の一宮北小学校）は、明治６年に約 500m南にある臨済宗・宝樹院に設立された中尾学校を前身とし、昭和 15 年に現在の場所に中学校校舎を移築した。一宮町誌[110]に、疎開当時の校舎の写真が残されている。但し昭和 36(1961)年に屋根を葺替えた後の姿である。現在の校舎は RC 造であるが、校友会が寄付した「ふれあいポート」に古いコンクリート製のベンチがある（2018 年 8 月撮影)。

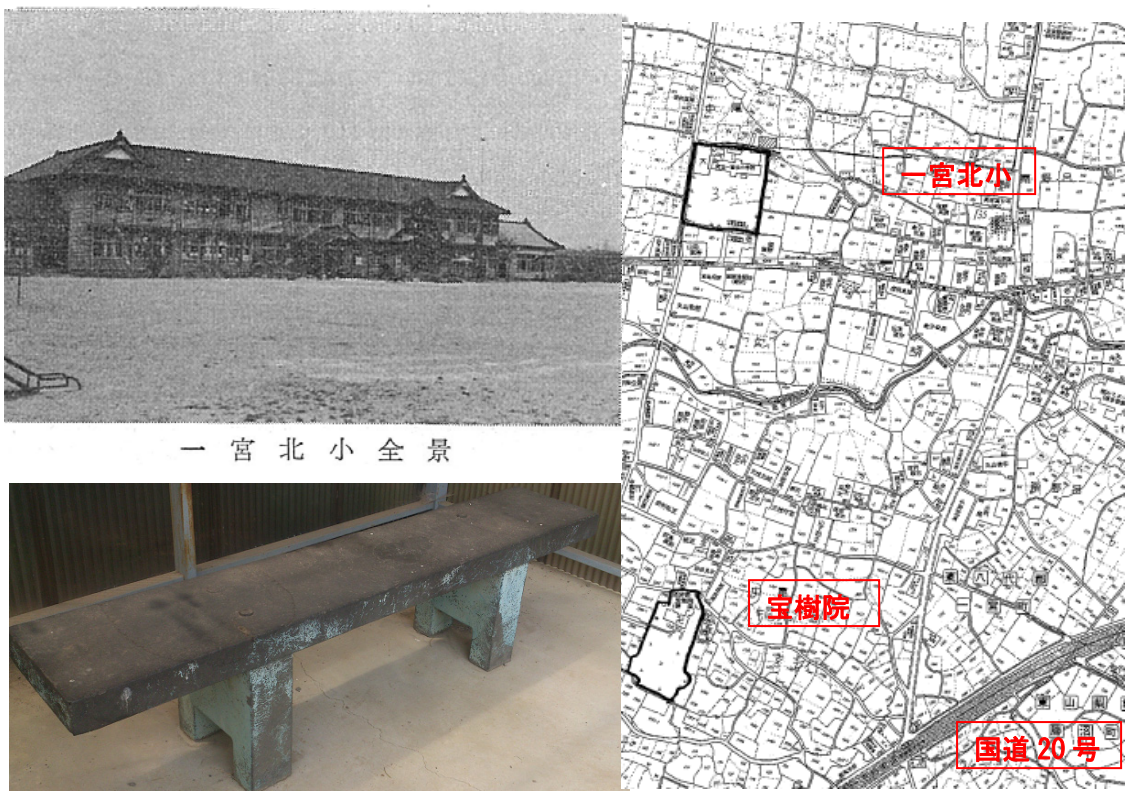


図 4-6-27 相興小学校（現、一宮北小学校）と法樹院の位置

なお、一宮北小学校の昭和 20 年 7 月 13 日の日誌に、大蔵省から運搬のため数人来た記録があるという。また周辺には「法珠院」という名称の禅林はないという（笛吹市文化財課内田裕一氏の教示による)。

１２．空技廠の機材輸送

「私の本籍は大蔵省営繕課建築研究室にあったが、前年の 12 月以来、木製機研究の手伝いと称して横須賀田浦の空技廠の方に通い、甚だ気ままな日を送っていた・・・

（終戦後）空技廠のすばらしい設備、計器、文献等の行方が何となく気になり・・・

空技廠に通うこと３回、・・・この機材運搬は空技廠が田浦を引き払う８月２０日まで続いたが、運んだ荷物の格納にはこれまた頭を悩ました。・・・結局これ等の機材は、私の家の物置と防空壕、今一つ当時大蔵省営繕課の疎開先であった等々力の園芸学校の講堂の控室に押し込んでおいた。」

出典：竹山謙三郎「建研創設時代回想記」、建研 20 年史[*66] 回顧録 2(p. 199)

出典：座談会「研究の今昔」；建築研究所 30 年のあゆみ[*69]

（等々力の園芸学校は現在、都立園芸高等学校として存続している。等々力の内務省防空研究所からは北北東に 1.6km ほどの位置にあたる）

図 4-6-28 東京都立園芸高等学校の位置（昭和 38 年）

１３．ＧＨＱによる接収と兜町への退避→研究所開設地の探索へ

「９月８、９日頃アメリカ軍が東京へ進駐するとほとんど同時に大蔵省にもドヤドヤと入ってきて接収を言い渡し、12 日までに清掃してひき渡すべきことを厳命した。当時大蔵省の建築研究室は山梨県相興村に疎開して居たから、100 トン、50 トン、20 トン、5 トン、等強度試験場にある機械類或いは恒温恒湿ボックス、低温ボックス等は既に取り除いてあったが、まだ４、５台の工作機械、コンクリート恒温養生水槽、その他多数の施設が残って居たし、半身不随の自動車数台、其他多数の資材をバタ屋の様に抱え込んで居たので、当時研究室残留のわずかな人数では如何ともなし難かった。・・・結局資料其の他紙類は中庭で焼却し、吾々の力で動かせる程度の資材は裏庭に持ち出して積み重ね、重量物は止むを得ず室内に残し、手に持てるだけの道具をたずさえて指定の 12 日夕方には一応兜町の証券取引所に避難する事が出来た。」

出典：竹山謙三郎「建研創設時代回想記」、建研 20 年史、回顧録 2[*66]（p. 199）

出典：座談会「研究の今昔」；建研 30 年史[*69]

１４．野毛の地誌における防空研究所

地方史・地誌の範疇に属する情報として、ウィキペディア「野毛（世田谷区）」に、以下の記載がある。［https://ja.m.wikipedia.org/wiki/野毛（世田谷区）］(2018 年 7 月時点)

「現行行政地名は野毛一丁目から野毛三丁目、郵便番号 158-0092」

「

現存しない施設

・等々力ゴルフリンクス

玉川野毛町公園と、そこに隣接する国土交通省住宅、都営住宅の敷地は、戦前、目黒蒲田電鉄の経営する「等々力ゴルフリンクス」というゴルフ場であった。等々力駅前のゴルフ橋はその名残である。1931 年にオープンしたが、戦争の激化により 1939 年に政府に買収

され、内務省防空研究所が置かれた。戦後、一部に引揚者住宅が建設された時期もあったが、環状 8 号線整備による道路拡幅に伴って施設の再配置が実施され、現状のように三分割されて今に至っている。なお国土交通省住宅は 2.8ha（１１棟）の大規模な団地であるが、国会公務員宿舎再編成の対象となり、2016 年６月には世田谷区へ公園用地として売却されることが決定された（一部は保育園となる）。また都営住宅も 2017 年から老朽化により、一部の建替えが開始されている。」

１５．都市計画学会五十年に際しての対談（入沢）

「(対談)

入沢　就職しはじめは、先ほど話が出ました、内務省の防空研究所です。これはまったく偶然に入っちゃったので、それが研究者になるきっかけになりました。・・・入った時に辞令をもらったのが菱田さんでした。辞令をいただいたんですけれども、私、海軍に召集くらいまして、ぜんぜん勤めずに帰ってきて、それで改めて辞令いただいたのが、中沢誠一郎さんです。・・・２３年ころです、大阪市立大学の教授になられました。

高山　あれは戦前じゃないかな、「新都市の構成」というの。ぼくはあれを引き受けて、召集されちゃって、菱田さんが書かれたんですよね。

入沢　当時本がありましたのは、それと石川先生の「国土計画及び都市計画」の２冊しかなくて、あればっかり読んでいました。

高山　等々力のほうに防空研究所というのがあって・・・。

入沢　今、宿舎がありますね。

伊藤　砧の緑地のへんですか。

入沢　いえ、それよりもちょっと。今の第 3 京浜へ入る入口です。今、公務員宿舎ですよ。

木村　防空研究所のことは菱田さんが初代の所長で、ぼくもそこへ、最初できた頃入れられちゃったわけだ。昭和１４年ころでしたね。それから私は支那海の向こうへ飛んでいったわけだが、防空研究所というのは、研究的にはあまりやれなかったんじゃないのでは。実験的なことをやったとしても。

井上　海軍の人もいましたね、あのとき。

入沢　まわりがそういった雰囲気で、都市計画というものが少しわかったんですね。そのうち、先ほども話がでましたように、防空研究所がつぶれて、かわりにそれをどう改組しようか。そうしたら、たまたま大蔵省営繕局の藤田金一郎さんが、建築研究所をつくろうじゃないか、一方、内務省の防空研究所も、内務省はなくなってしまった。どこに行こうか。戦災復興ということでいっしょになろうではないかということで建築研究所の前身の戦災復興院技術研究所ができたのです。その内務省系は都市計画のようなソフトの面を相当持っていたんです。牧野さんとか、新海さんとか。それで、いっしょになった場合に、そのようなソフトの面、住宅問題と都市計画、その２つくらいを１つのセクションの研究部にしようではないかという動きがありまして、一応はできたんです。

木村　そうすると、防空研究所を引き継いだような形かね。
高山　それから、大蔵省営繕局、あれが強かった。だけど、あれは少し縮小したんだよね。それで、技術部門を建研に持ってきた。」
［都市計画学会五十年史[*101]］所収、出典「都市計画」100 号、(1978 年 3 月 20 日)］座談会：国土総合開発㈱顧問木村三郎、東京大学名誉教授高山栄華、東京大学工学部教授井上孝、横浜国立大学工学部教授入沢恒、司会都市計画編集委員長伊藤滋

なお、文中にある初代防空研究所長であった菱田厚介は、石川栄耀、高山英華らと共に同書掲載(p. 2)の日本都市計画学会発会式の記念写真に写っている（1952. 10. 6 早稲田大学大隈講堂)。

「技術員養成所」終戦直後の技術員養成所については、断片的な記述しかないため未詳な点が多いが、二つの異なる「技術員養成所」に関して以下の資料を見出すことができた。16～22 は板橋と等々力の大工学校であり、23~26 は、沼津の施設である。

１６．戦災復興院技術員養成所(大工学校)

附属技術員養成所は、昭和 20 年 11 月に設置された（20 年史年表)。その後昭和 21 年４月には復興院技術研究所が発足した。昭和 21 年７月[復興情報]は、その３ヶ月後の状況を描いている。

*「板橋町四丁目１２８番地、ここに職員共に百二十人の集団する」*とある。*「此の養成所に於いては研究の成果としての新建設技術に関する講習並に伝習に主眼を置いてゐます。・・・差当って木工機械作業を中心とした大工の短期養成を行って、修行の暁には本院の担当工事に腕をふるはうとしてゐます。校長は技術研究所長があたっており」*
として、養成人員：１２０名、期間：１年、養成中は月給４０円、卒業後は２００円以上で２年以上　という条件で募集を行い、３２４名の応募があって１４０名が選ばれ、入所式が４月１３日、寮宿舎に入寮したと記されている。また研究所と労働科学研究所が生徒を使って実験をしていた。[50 年史[*73] 資料 19, p. 340、出典：戦災復興院：「復興情報」昭和 21 年７月号、樋口寛三筆]。

板橋町（いたばしまち）四丁目は、昭和 7(1932)年に板橋区が成立した際に設定された。この板橋町四丁目は、昭和 33(1958)年に大山町（おおやまちょう)、大山西町、大山東町、幸町、南町および大谷口町に分割された。現在の板橋四丁目とは全く区域を異にしている。

昭和 32(1957)年の１万分の１地形図には、板橋町四丁目の区域が描かれている（図 4-6-28)。

この区域を昭和 38(1963)年の全住宅精密図帳の区域図と比較対照すると、概ね大山町、大山西町、幸町を合わせた範囲に対応し、一部が大山東町、大山金井町、南町にかかっている。その６町の内のいずれかであるが、昭和 33(1958)年に地番は振り直されているため、当時の「四丁目１２８番地」の位置を直ちに特定することはできない（板橋区役所戸籍・住民課の教示による)。

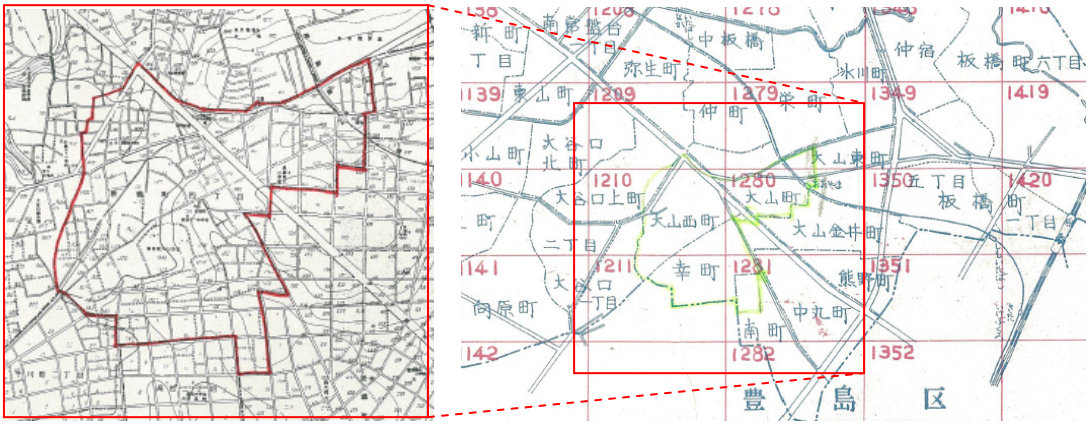


図 4-6-29 板橋町四丁目の範囲

既に「大山町」の地名が成立していた 20 年史[*66](1966 年)には、昭和 20(1945)年 9 月に建築研究室のための研究施設を求め歩いていた竹山が三番目に訪れた場所として *「池袋の現大山寮」* のことが記されている。*「ここは元陸軍造兵廠の工員宿舎で、大蔵省営繕課が終戦直後に職員宿舎用として確保していたものである。私のみた９０月 20 日頃は未だ全くの明き家で、蚤だけば猛烈に跳梁して居たが、もちろん研究所の施設としては全く不適当であることが解った。」*[竹山：建研創設時代回想記、20 年史 p.199]

昭和 20 年８～９月に米国陸軍が撮影した空中写真を用いて戦災復興院が作成した 1 万分の 1 地形図には、板橋町四丁目の記載があり、ここに、寮と見える建物が描かれており、建物の配置は、上記の昭和 38 年地図に長屋状に描かれた「建設省大山宿舎」と一致し、ほかに相応しい地物が周囲に認められないことから、竹山が研究所候補地として訪れた元陸軍造兵廠の工員宿舎が、技術研究所が所管する技術員養成所として修理され、後に（ほぼ戦前からの建て方のまま）建設省の宿舎となり昭和 38 年にはまだ存続していたと推定する。なお、この場所にはその後、中層（３～５階建）の都営幸町アパートが立地している。

現在の番地から遡ることは可能であるため、「幸町４５番地－２」について閉鎖登記簿を閲覧してみると、以下のように推移していることが分かった。

大正 9(1920)年 2 月時点 北豊島郡板橋町大字下板橋字境久保 1290 番地

昭和 7(1932)年 板橋区板橋町四丁目 1290 番ノ二

昭和 33(1958)年 3 月 1 日 板橋区幸町 45 番地

つまり、大字下板橋から板橋町四丁目に変更となった時点では番地は変化していないが、昭和 33 年に幸町に変更になった時点で番地も振り直されている。

調査した土地は昭和 23 年に大蔵省に物納され、昭和 35 年に東京都に払い下げられている。昭和 38 年の住宅精密図には、「建設省大山宿舎」と並んで、「民生住宅」（都営住宅の前身と考えられる）が、終戦直後と同じ建物に注記されている。その後、1986-90 年に中層住宅（３～５F）に建替えられ、さらに 2008-11 年に高層住宅への建替えが進んだ。幸町アパート(45-14)1986-90, 板橋幸町アパート（45-5）2008-2011

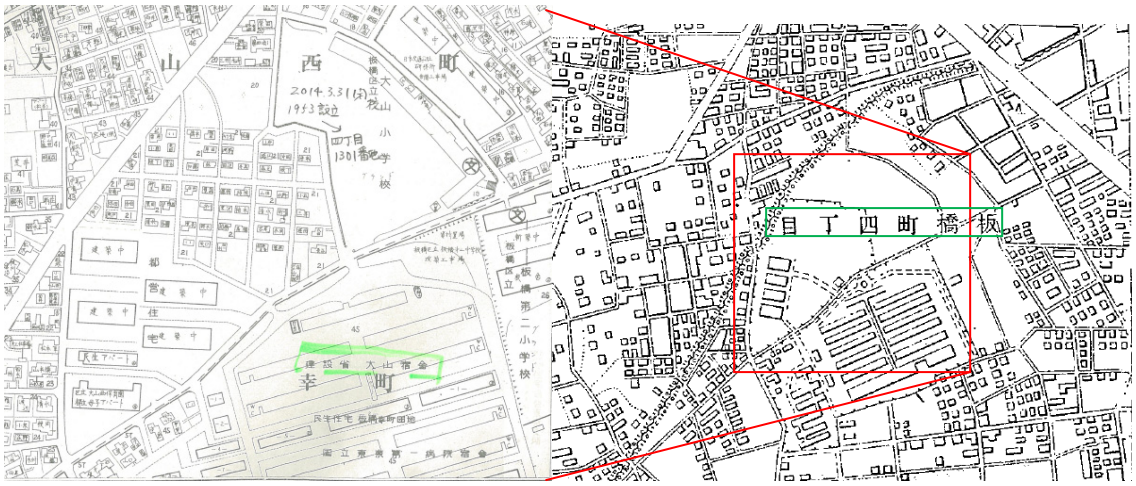


図 4-6-30 建設省大山宿舎（左：昭和 38 年全住宅精密図帳　右：昭和 20 年 1 万 1 地図戦災復興院）

戦災復興院技術員養成所は、昭和 21 年 11 月からは、旧内務省防空研究所があった等々力に場所を移し、建設省建築研究所に継承されて昭和 25 年度まで運営された。

１７．大工養成所の写真

この元ゴルフ場クラブハウスであった建物の写真を掲載した「３０周年記念アルバム」*70は、佐藤慶一氏[19]が編集され、版下に使用された写真は現在もお持ちであった（2018 年 6 月 13 日インタビュー、10 月 11 日スキャン）。木造平屋瓦葺の建物であったという。

１８．大工学校の継続期間について

亀田 *「戦後一番初めにやったことは木造プレハブや大工作業の機械化の研究で、これは業界全部を挙げて非常に熱心に行われました。この時代はなくなられた大竹さんがおられまして、今盛んに使われている手持電動鋸や鉋の研究もやったものです。また、等々力にいた技能員養成工を大久保へつれてきて、海軍から木工機械を持ってきて、大工さんを班長にして仕事をとって、養成工を養いながら研究費を稼いだ時代がありました。それは 23 年ころまでです。竹を利用できないかということで、竹の家をつくったこともありました。*

ところが 23 年に福井地震が起きて、都市の不燃化ということから木造のプレハブに対して批判が出ました。それから旧に鉄筋コンクリートの研究が始まったわけです。そこで木造プレハブの研究はほとんど立ち消えになりました。」［座談会建研の今昔］

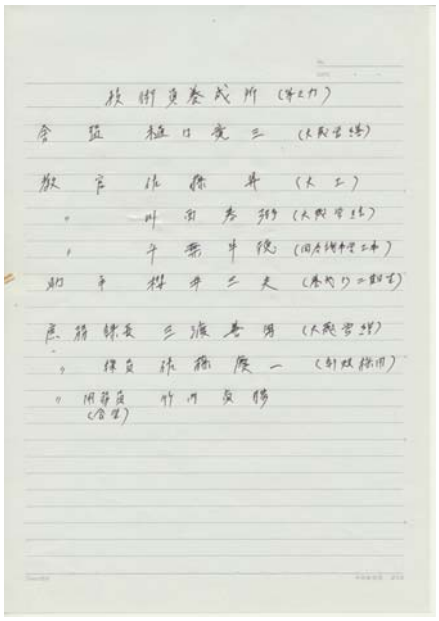
文中、*「海軍から木工機械」*は、20 年史年表で昭和 24 年 8 月 *「沼津より木工場を移設」*した平賀・篠澤らのことをさすものであろうか。同年表では、昭和 26 年３月に *「建設省技術員養成所を解散（７期で修了）」*とある。ある時期から研修期間を半年に短縮したものであろうか？なお、篠沢「創造する人々」には運輸施設本部が東京小石川にあり、沼津との間を頻繁に往復していたとの記述もある。

１９．佐藤慶一氏

大工養成所に採用され、その後建設省建築研究所、建築研究振興協会に勤務された。

2018 年６月 13 日、川越のご自宅に小林と同協会の田中良寿氏が訪問し、お話を伺った。30 周年記念アルバムの編集を担当され、版下に使用された写真を多数お持ちであったため、

同年 10 月 11 日にスキャナーを持参して記録保存用の画像データを取得した。記憶に基づいて作成していただいた当時の養成所の組織を図 4-6-31 に示す。



技術員養成所（等々力）

舎監　樋口 寛三　（大蔵営繕）

教官　佐藤 昇（大工）
〃　　川西　春弼（大蔵営繕）
〃　　千葉　牛徳（国会議事堂工事）
助手　桜井　三夫（養成所二期生）

庶務課長　三渡　善男（大蔵営繕）
〃係員　佐藤　慶一（新規採用）
〃用務員　竹内　貞勝
（食堂）

図 4-6-31 等々力の研修所の職員構成

板橋の養成所に関して上記註 16 に引用した復興情報の筆者である樋口寛三（文中にある樋口監事と同一人物と思われる）が、等々力の舎監となっている。

佐藤氏のご記憶によると、昭和 20～21 年に板橋で二期行われた後、場所を等々力に移して研修員の数を一期２０人に減らし、７期まで継続された。等々力で採用されたため、板橋の研修所跡地を訪れたことはないが、後に昭和 60 年頃に建築研究振興協会の板橋試験所がこの付近に開設された時期があるため、東上線大山駅付近の土地勘はお持ちであった。

２０．防空研究所彙報[*62]

「発刊の辞

内務省防空研究所は昭和十四年七月三日防空に関する研究および講習並に防空資材の検定に関する事務を行ふ為設置せられ、爾来この使命の遂行に任じて来たのであるが、時局下当面せる問題に関し史料の蒐集整備並に調査応用を遂ぐるに急を要するものありし為研究部門に於ては十分系統的な成果を挙ぐるに至らなかった。これ今日まで彙報を刊行するに至らなかった理由である。調査研究を了した事項は其の都度報告書を関係方面へ配布しつゝあつたのであるが、今日機会を得たので敢て当初開設以来の研究事項を取纏め彙報を刊行することゝした。

国民防空技術の調査研究はその内容広汎多岐に亙る為是非共関係研究機関の連絡調整を図ることが肝要であって、曩(さき)に次官会議に於て之が統制連絡に関する申合せが成立したのも斯から主旨である。本彙報が関係方面に於て防空史料として活用せられるのみならず、此の方面への連絡調整への一助ともならば望外の幸せであるとともに今後各方面よりの緊密なる連絡協力を冀(こいねが)ってやまぬものである。

所期の如き内容を充実し得ず編集上不備の点も多々あると考へるが将来巻号を重ねるに従ひ漸次訂正改善を加へ逐次内容の充実を期し度いと考へるものである。

内務省防空研究所長　中沢誠一郎」

引用中、旧字体は意味を損なわない範囲で新字体に直した。なお、奥付の住所・電話番号は、

「東京都世田谷区玉川野毛町一〇〇三番地 電話田園調布 4001,4002,4003 番、玉川 370 番」

とある。

２１．中島論文(2017)[*108]

日笠端の退官講義の講義録「都市計画研究三十五年を顧みて」（1981 年）を資料として、*「1941 年に設立された内務省防空研究所を前身として 1945 年 8 月に発足した内務省国土局分室は、1946 年 4 月に大蔵省官房営繕か建築研究室、陸軍技術研究所等とともに新設の戦災復興院総裁官房技術研究所（所長：藤田金一郎）に統合された。防空研究所出身の新海悟郎、広井正路、内山諫、陸軍に籍を置いていた日笠端（1920 年生、1943 年東京大学工学部建築学科卒、卒業論文は丹下健三の指導による「大都市改造論」）ら建築系出身者と、浅野英、坂部正勝、高木義弥ら土木系出身者で技術研究所内に都市計画研究科を組織し、都市計画と建築経済の研究を開始することになった。その後、日笠とは東大の同級生で、防空研究所を経て技術研究所で防火研究に従事していた入澤恒(1919 年生、1943 年東京大学工学部建築学会卒、卒業論文は武藤清の指導による「耐弾床版に関する研究」)が志願して都市計画研究科に移籍してきた。翌 1947 年には下河辺淳(1923 年生、1947 年東京大学第一工学部建築学科卒、卒寮論文は丹下健三の指導による「都市に於ける工業の地域構造に関する研究」)らが加わり、都市計画研究の体制が整えられていった。」*としている。

２２．建築研究所要報[*64]

建築研究所図書室に所蔵。ガリ版刷り手書原稿である。昭和２１年の第 1 号から昭和 27 年の第 162 号までを採録している。

第 1 号は、戦災復興院技術研究所、第 2 号～第 3 号は、建設院第二技術研究所、第 4 号以降は建設省建築研究所の名称で報告されており、研究所の名称変更を超えて番号は通っている。

２３．技術員養成所②運輸建設本部技術員養成所（元、海軍施設本部野外試験場）

「竹山と久田のねぐら探しがはじまった。ところが、これがなかなかうまくいかないのである。沼津の海軍施設部研究所、目黒の海軍技術研究所、池袋の陸軍造兵廠工員宿舎、九段軍人会館、その斜め向かいの海軍将校クラブと、次から次へと当たって歩いた・・・」

[村松 1965[*102]p.42]

昭和２０年に沼津で開講された。昭和２３年に建設省に統合され、建築部門は建築研究所、土木部門は土木研究所の傘下となったことが以下のように記述されている。

「平賀謙一は、海軍施設本部沼津野外実験場で太平洋戦争の終戦を迎えた。・・・昭和二十年八月二十六日運輸省に移管されて運輸建設本部として発足、戦禍に荒廃した国土の復興

再建に活躍することとなった。・・・昭和十九年(1944)一月に設置された、海軍施設本部沼津野外実験場も、運輸省への移管に伴い「運輸建設本部技術員養成所」と改められた。

沼津市郊外（静岡県駿東郡清水村）にあった技術員養成所は、大きくは土木部門と建築部門に分かれていた。技術員養成所の所長には、土木担当の西村義一がなり、平賀謙一は、建築担当の副所長となった。人員は、土木・建築の研究者・技術者及びその他職員を合わせて四十～五十名程」[篠澤*104p.34～35]

「昭和二十三年(1948)七月、建設省が設置されると、技術員養成所も建設省に移管されて、建設省の「建設工事本部技術員養成所」となった。・・・昭和二十四年(1949)七月、建築研究所に併合された。建築研究所に移ったのは、・・・技術７、事務２、そのとき第四研究部にいた６名と合流し、総勢１３名で建築生産施工技術研究開発のスタートを切った。」
[同書 p.45]。

平賀「私の思い出」（２０年史*66）には、次のように記されており、沼津からは人員のみならず、宿舎、木工場、機械をもってきたことがわかる。

「海軍施設本部を経て運輸省建設本部沼津技術員養成所に入り、建研に移る前はそこの副所長をしていた。この技術員養成所はもと海軍施設本部の実験所でたくさんの研究施設をもっていたが、特にそこにあった木工機械工場は日本一を誇るものであったので、私達はそこで木造プレファブ住宅を盛んに研究し、また実際に沢山の製作をやり、それらの作品ではいくつかの賞を得た。当時建研の所長は藤田先生であったが、同所長の非常なご懇望もあり、また私も沼津技術員養成所の建築部門だけを建研と合併することを望んでいたが、丁度当時は建設相の組織変動期でもあったので、この案は早速当局に受け入れられることになった。それは丁度昭和 24 年 6 月で、私は元研究員井上健君、現総務課今泉忠君、現研究員篠沢和久、篠沢清見両研究員他合せて 10 名をつれて建研にやってきたのであった。・・・私達はしかし宿舎も、木工場その他の施設、機械等沢山もってきていたので漸く心よく了解された。」

建設省三十年史*96によれば、平賀謙一は昭和 24(1949)年 7 月 31 日から昭和 37 年 10 月 16 日まで第四研究部長を務めている。

昭和 25(1950)年 9 月には、研究員平賀謙一と篠沢和久が連名で、建築研究所要報*64No.115 として「火山砂利を使用せる現場打軽量コンクリートの施工」を出版している。

このように沼津の技術員養成所の建築部門は第四研究部となった一方、土木部門は沼津に残り、土木研究所沼津出張所となった。

*「昭和 24 年(1949)、元運輸省運輸建設工事本部技術員養成所を合併する」「昭和 28 年(1953)、技術員養成所を沼津支所に改称」。*千葉支所の設置に伴い、廃止された。*「昭和 35 年(1960)、土木研究所千葉支所設置（沼津支所統合　機械施工部となる）」*
[土木研究所 70 年史*99、p.266, 土木研究所年表(平成 4 年 9 月)]

これらに対応して、昭和 26 年度の土木研究所組織図には技術員養成所があり、その配下に庶務課、指導課、実験課がある。支所設置後の昭和 33 年度の土木研究所組織図には、沼

津支所があり、その配下に庶務課、性能試験研究室、施工研究室が置かれていた。少し時期は隔たるが、昭和 43 年度(1968)年の組織図では、沼津支所はなくなり、千葉支所の配下に機械施工部があり、その下に機械研究室、施工研究室、土質研究室がある。以下の記述がある。

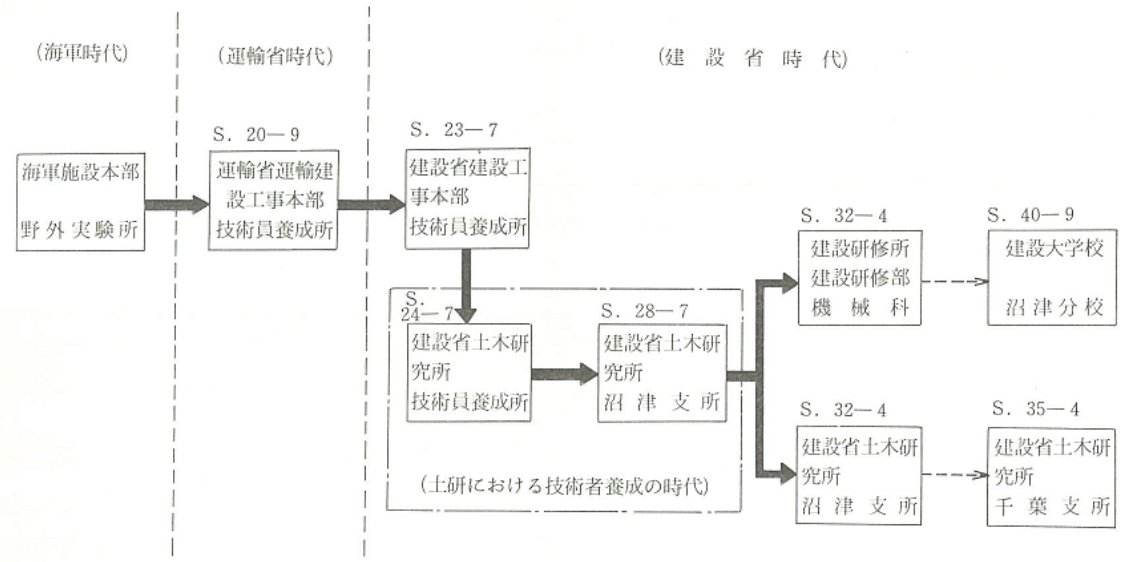


図 4 -6-32 技術者養成機関の変せん[土木研究所五十年史掲載、*図 7－2* p. 102]

「建設省における技術者養成の過程をみると、その創造期には土木研究所沼津支所が重要なる役割を果してきたといえる。その沿革について概要をのべると、図－7．2のごとく昭和 20 年 9 月海軍施設本部野外実験所の施設を引つぎ、運輸省運輸建設工事本部技術員養成所となり、昭和 23 年 7 月建設省土木研究所技術員養成所、昭和 28 年 7 月に建設省土木研究所沼津支所となった。さらに昭和 32 年 4 月建設研修所の発足とともに養成部門は土木研究所からはなれて建設研修所研修部機械科となり、これは現在、建設大学校沼津分校となっている」[土研 50 年史][*95]

建設大学校の組織の変遷に以下の記述がある。

「*(3)旧沼津分校*

昭和 48 年 4 月に廃止された沼津分校の施設は、旧海軍施設本部野外実験所の施設で、当初土木研究所沼津支所が使用していたが、建設研修所が設置されたときにこれを引き継ぎ、建設研修部機械科が当所に置かれた。敷地の面積は 85,950 ㎡（約 26,000 坪）で、それに木造の建物が附属しており、当初これらは大蔵省所管の普通財産であったため、一時使用の形態により使用していたが、昭和 36 年 8 月 9 日建設省所管の行政財産へ所管換の措置が完了した。土木研究所沼津支所から引き継ぎを受けた建物は老朽化が著しく、通常の風雨でも被害のでる状態で、台風時には倒壊、大破あるいは電線の疲労による火災などの被害が発生し、それらの災害復旧工事がかろうじて施設改善の工事といえる程度であった。昭和 42 年にはじめて官庁営繕費が認められ、実習室の新築工事が行われた。

沼津分校において実施された研修は、沿革的に機械施工技術および建設機械の操作に関

するコースが主体となっていたが、昭和 30 年代後半からの、建設工事の機械化施工の推進に関する国の行政施策の転換、研修棟の統合による研修の合理化の指向等により、昭和 48 年４月同行は廃止されることになった。

同校の廃止に伴う国有財産の整理にあたっては、土地 14,198 ㎡並びに当該土地に定着する建物延 198 ㎡、立木竹６本及び工作物一式は沼津工事事務所の用に供するため、中部地方建設局に所属替し、残りの土地 49,883 ㎡並びに当該土地に定着する建物延 2,863 ㎡、立木竹 188 本、工作物一式は用途廃止のうえ、大蔵省（東海財務局）に引き継ぐことになった。

沼津分校の敷地は昭和 36 年 6（ママ）月 9 日大蔵省より所管換を受けたものであるが、当該土地の一部に旧軍未登記（未登記の国有地）の土地が介在しており、国有地への所有権の移転登記が必要であったが、当該事務は完了していなかった。土地の引き継ぎにあたっては、まず所有権の移転登記を完結する必要があるが、公簿上の名義人をめぐる複雑な登記事務が生涯となって当該所有権移転登記は遅々として進まず、昭和 50 年 12 月 26 日に至り、ようやく大蔵省に引き継がれることになった。また中部地方整備局に対する所属替についても翌 51 年１月７日及び２月 24 日に完了した。

」「出典／建大３０年史」*98

この土地に関して、昭和 47 年 11 月に沼津工事事務所から*「国有財産所属替承認申請書」*が出されている。所在地は「静岡県沼津市上香貫山ケ下」となっているが、現在の沼津河川国道事務所（〒410-8567 静岡県沼津市下香貫外原 3244-2）の場所である。以下の理由書が付されている。

「当工事事務所富士海岸出張所は昭和 41 年９月 24 日台風 26 号により高波は最大波高 15m 以上の大波により堤防を越え 100m 以上の松林を抜け背後の住宅を破壊し死者 10 数名の被害を受け昭和 42 年度より国の直轄工事として着手した範囲は田子浦港～新沼津湊間の 14.7km である、昭和 42 年 10 月 23 日富士海岸出張所庁舎が新築され、敷地は、当初から借受てきたが地主より、敷地返済の申出が再々あったが、適当な場所が見当たらず、敷地候補地をさがして来たところ、海岸に津書く地理的条件で最適な県背悦大学校沼津分校が昭和 47 年度中に本校に統合される予定であるので、沼津分校の一部敷地(1495.80 ㎡)を所属替するものである。」

場所は沼津市と清水町(外原)の境界（赤破線）付近にあり、事務所は「沼津市下香貫外原」で案内している。下香貫は沼津市、外原は駿東郡清水町の地名である。

沼津河川国道工事事務所 www.cbr.mlit.go.jp/numazu/jimusyo/gaiyo.html の資料に、

「所在地：沼津市下香貫外原 3244-2 055-934-2001(代)

沿革：

昭和２年８月　狩野川測量員詰所

昭和２年９月　狩野川改修事務所

昭和 18 年 10 月１日　沼津工事事務所（河川・道路）

・・・

昭和 39 年 7 月　狩野川砂防工事事務所を統合

昭和 54 年 11 月　沼津市下香貫外原に事務所を移転

沼津国道維持出張所　長泉町下土狩 1027-1　　」

とある。

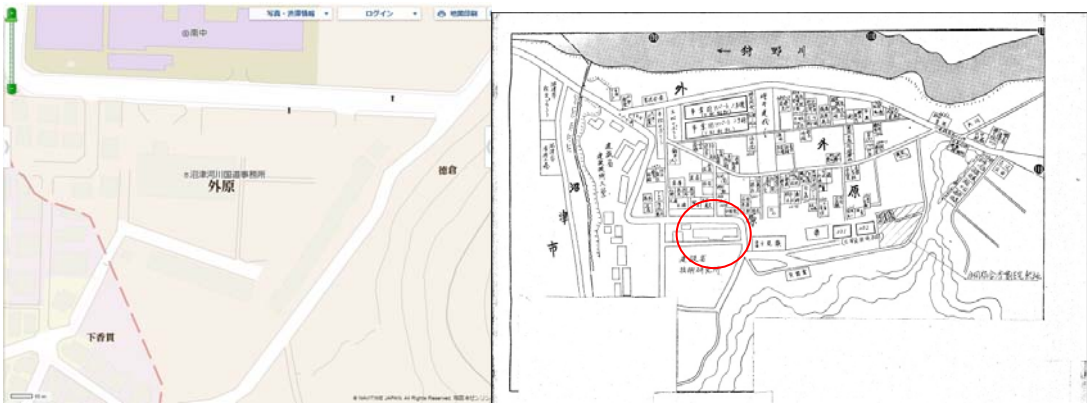


図 4-6-33　建設大学校沼津分校跡地 (左 2018 年現況、右 1972 年当時(○で囲んだ書き込み文字のない建物が、旧本館の位置に相当)

狩野川は、天城峠付近を源流とする、沼津市・三島市・伊豆市・伊豆の国市・富士市・函南町・清水町・長泉町を含む流域を有する。清水町で日１２０万トンの柿田川の湧水が合流する。同事務所の web サイトによると、1960 年代には１３０万トンあったが、1980 年代には１００万トンに減少し、その後１００～１１０万トンで推移している。飲料水に２０万トン、工業用に１０万トンが利用され、残り７０万トンが狩野川に流入している。この豊富な湧水は戦前、東京への給水も検討され、海軍施設用地選定の理由ともなった。海軍用に整備された給水施設は戦後、沼津市に移管された。昭和 50(1975)年３月から駿豆水道により、熱海市への給水に利用されている。

２５．戦前・沼津の海軍関連機関

①沼津海軍工作学校跡の石碑（清水町長沢）

（表）*「沼津海軍　工作学校跡」*

同町のホームページの解説では、「約 56 万平米の敷地に実演場や兵舎などを備えた大日本帝国海軍の学校。1944 年 6 月からわずか 1 年 2 カ月で役目を終えたため「幻の学校」と呼ばれる。1998 年に石碑が、2008 年には構内図看板が設置された」。

055-975-6987 清水町観光協会 https://www.surugawan.net/guide/612.html

場所は、技術員養成所（現、国土交通省沼津河川国道事務所）から北に、狩野川を渡った対岸にある。

②海軍音響技術研究所

なお、終戦時に沼津市内に存在していた海軍音響技術研究所に関しては、

*「1947 年　海軍音響技術研究所の建物を用いて沼津第三中学校開校」*とあり、石碑が残されている。(htpps://ja.m.wikipedia.org/wiki/沼津市第三中学校)

③海軍工廠

金岡護国神社（沼津市神田町）に石碑(1981 年)が残されている。

(表)*「沼津海軍工廠　工員養成所跡地 」*

(裏)*「・・・昭和十八年四月開校、昭和二十年八月閉校、沼廠工養会一同、昭和五十六年六月」*

２６．竹山が訪れた沼津の海軍施設部研究所

この場所は、昭和 20 年９月に竹山が探訪した場所と思われる。

「竹山と久田のねぐら探しがはじまった。ところが、これがなかなかうまくいかないのである。沼津の海軍施設部研究所、目黒の海軍技術研究所、池袋の陸軍造兵廠工員宿舎、九段軍人会館、その斜め向かいの海軍将校クラブと、次から次へと当たって歩いた・・・」

[村松 1965[*102]p.42]

「どこかに新しく施設を求めることが当面の課題になった。入手可能の研究施設として頭に上るのは、陸海軍のものであるが、その中先ず沼津にある海軍施設部の研究所が話題に上った。当時は食糧事情が著しく窮迫して居たのでとうてい東京では生活出来まい。半農半研究の覚悟が必要であろうと考えられた。それで食糧事情の少しでも良い処、というので沼津が先ず話題に上ったと記憶して居る。そこで私が一応の偵察に行ったのが９月の中旬のことであった。殺人的に混み合う汽車にゆられ、一面の焼野原となった沼津駅に降り立ち、狩野川の縁に沿う荒れた遠道をトボトボと歩き乍ら、私の意欲は次第に冷めて、先方で山田所長から「ここを立ち退く意思は全然無い」と聞いたときは寧ろほっとした。」

［20 年史[*66]p. 199］

２７．建築研究所支所の構想

「研究所運営連絡会議で、「建築防災研究機構」の設置計画が提案せられ、住宅局が予算化を図ることとなったが、実らなかった。

又、阪神地区と北海道で建築研究所支所を設置することが提案され、研究所から予算提出したが、デフレ政策の折であったし、既存経費で実施することが検討され、大阪府庁、北海道庁との交渉に入ったが、当事者の熱意はあったが、経費分担問題で進捗せず、停頓した。しかし、その後数年にして、北海道々立コンクリート・ブロック指導所（後ちに寒地建築研究所）が発足し、大阪では昨年、国と府市、業界等の補助と寄附で日本建築総合試験所（千里山）が発足した（後略）」（藤田：草創期の建研、20 年史[*66]p. 128)］、50 年史 p. 335

２８．奥多摩の地震観測研修所

文献３③（国地１０年史）掲載図(p. 61)。当時、青梅線は奥多摩駅（氷川）までであったが、鉱山鉄道として延伸された。

２９．住宅協会（航空写真地図の発行元）

最古の 1963 年版全住宅精密地図帳を発行した住宅協会の住所は、杉並区宿町 204 番地となっており、同地図で所在を確認することができる。日本住宅協会とは無関係の民間企業

である。公共施設地図航空株式会社の住所は、杉並区上荻 4-29-6 となっているが、同図で調べると、上記の宿町 204 番地と同じ位置であり、住所表示と会社名が変更されたのみである。この位置は 1986 年まで不変であるが、1988 年には、約 324m南の同区上荻 4-6-7 に移転している。

2014年３月シンポジウム寄稿文

旧・建設省建築研究所の筑波研究学園都市への移転
－その経緯と移転後の研究の展開－

本稿は、独立した文献として執筆されたものであるため、原著のオリジナリティを尊重して、頁数、図表番号および＊出典番号を原著のまま再掲しています。

２０１４年３月

棚　　橋　　一　　郎

旧・建設省建築研究所の筑波研究学園都市への移転
－その経緯と移転後の研究の展開－

目　　次　　（頁）

旧・建設省建築研究所の筑波研究学園都市への移転

－その経緯と移転後の研究の展開－

元・建設省建築研究所企画室長　棚橋一郎

１．建築研究所の筑波移転の経緯

（１）筑波移転の背景

１）建築研究所の歴史と移転前の状況

昭和１７年（1942 年）１２月、大蔵省大臣官房営繕課に建築研究室が設けられ、我が国の専門的な建築研究機関の発祥となった。

その後、営繕研究室は、太平洋戦争中の昭和１９年に山梨県下に疎開したが、終戦直後の昭和２０年（1945 年）１０月に、旧・第７陸軍研究所跡地（新宿区百人町）に研究所を設営した。そして、昭和２１年４月、戦災復興院の設置に伴い、旧、内務省防空研究所の一部などと合併し、戦災復興院大臣官房技術研究所として発足した。

その後、昭和２３年（1948 年）１月、建設院・第二技術研究所となり、さらに同年７月に建設省建築研究所（建研）となり、建築及び都市・住宅に関する国の唯一の研究機関としての活動を始めた。当時の研究部門は６科の編成で、総員６７名（研究員３３名）であった。(図１－１)

建研の発足から 10 年程度の間は、我が国の建築・都市関係の研究機関は他になく、学会においてもかなり主導的な地位を占め、戦後の建築技術および都市計画の研究の急速な発展に寄与した。

その後、民間や大学の研究活動が盛んとなり、こうした中で建研は、建築・都市計画および住宅に関する、我が国唯一の国立研究機関として、国としての主要な研究や、公共の利益と国民全体の福祉のための研究を行う事を基本とする建設省設置法に定められた活動を展開して来た。

その後、昭和３７年（1962 年）に、地震学および地震工学に関する国際研修を行う「国際地震工学部」が、また昭和４４年（1969 年）には「建築試験室」が設置され、さらに昭和４９年（1974 年）には、第６研究部(都市計画)が設置され、管理部門を増強するなどして、徐々に組織・定員を拡大し、総員１８２名の研究所として活動を展開して来た。(図１－２、図１－３)

この間、我が国の経済復興に伴い、昭和３０年代から始まった、人口、産業の都市集中の波は、昭和４０年代における経済の高度成長と共に、特に大都市圏地域において激化し、住宅問題や交通問題が深刻化するに至った。そして、

（昭和21年９月現在）

人員67名，うち研究員33名（養成所を除く）

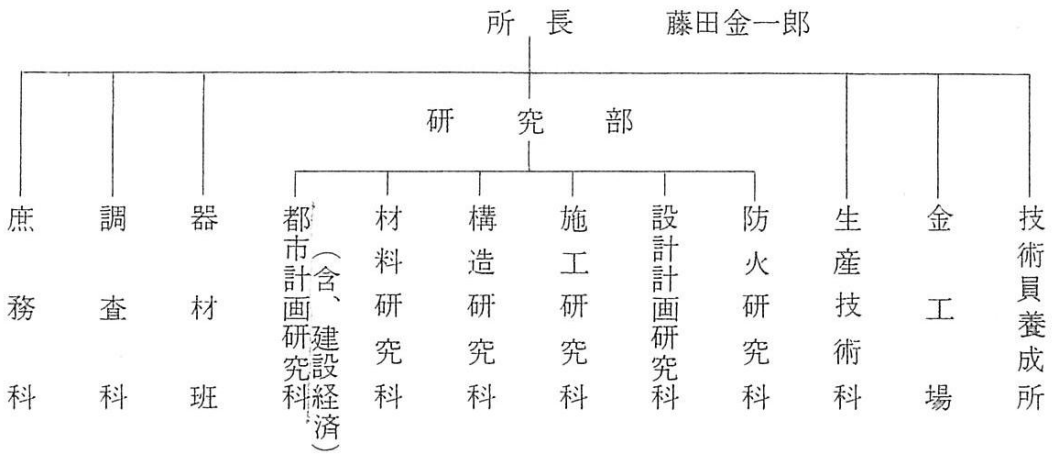


図１－１　戦災復興院技術研究所組織図[*1]

（昭和41年10月現在）

定員182名，うち研究員62名

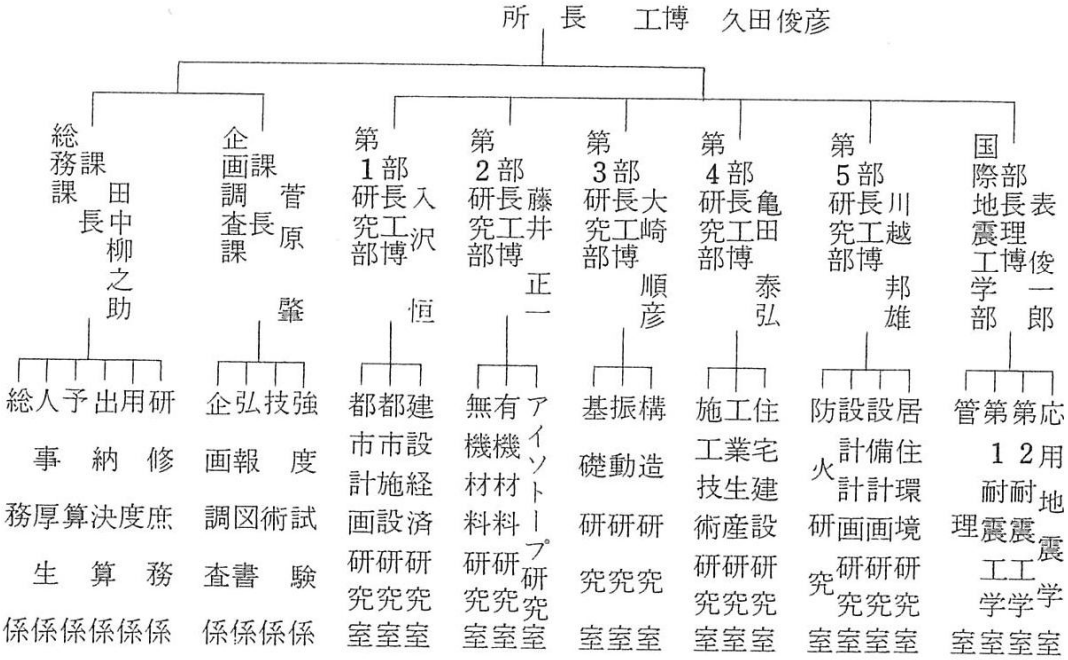


図１－２　建設省建築研究所組織図[*2]

（昭和51年 6 月現在）

定員 182 名，うち研究員82名

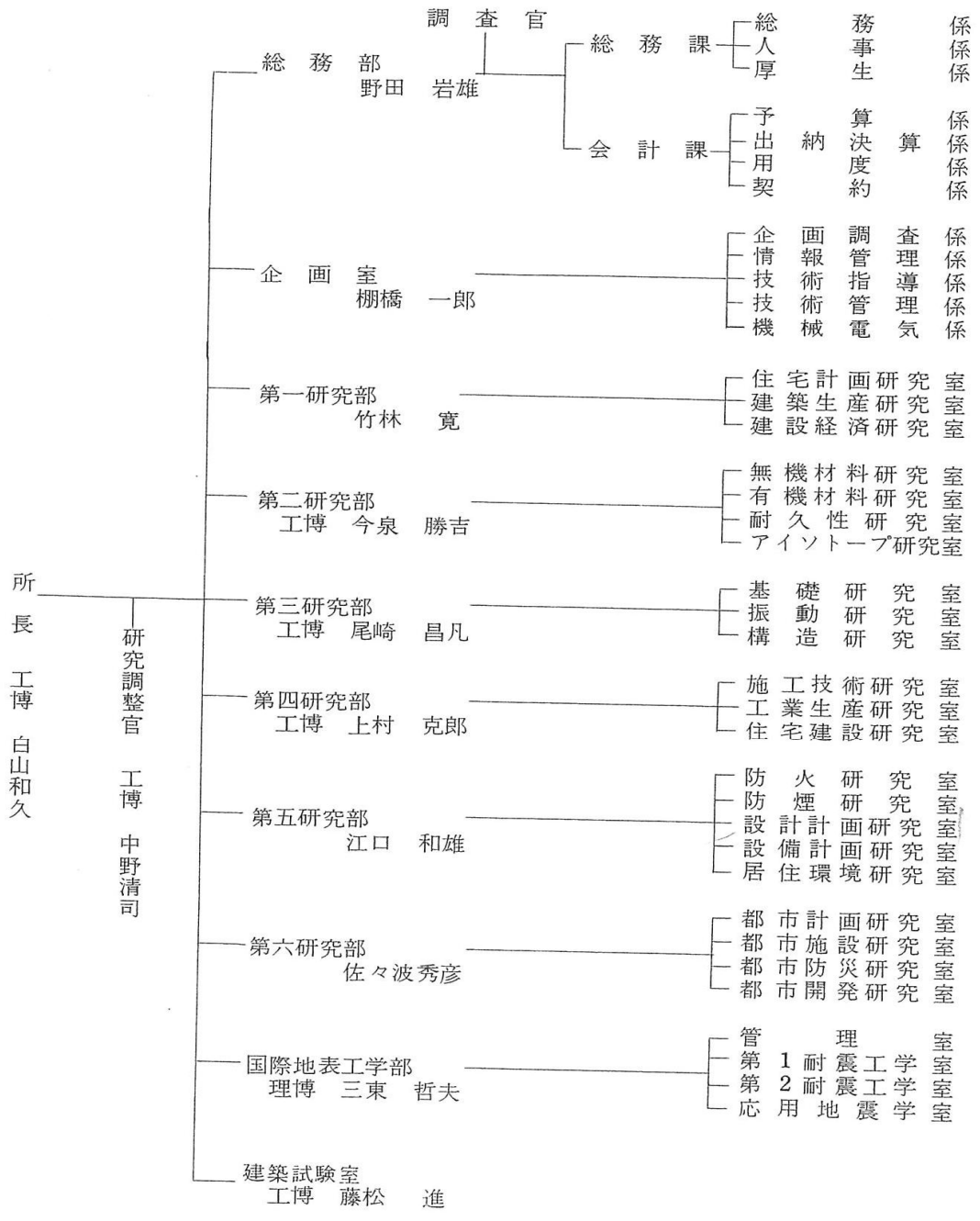


図１－３　建設省建築研究所組織図[*3]

都市環境対策や都市防災対策に関する技術的な問題に、如何に対応するか課題とされると共に、建築物の構造、材料および構法などに関する新技術の開発や、既存ストックの維持・管理技術に関する調査・研究のニーズが高まり、研究施設の整備が着々と進められ、これに対応して特に、昭和４１年～４４年の間には、実験施設や機器の拡充が盛んに行なわれた。

しかし、新宿・百人町における研究施設は老朽化しており、社会的なニーズに対応して、新施設の増設を行う余地はなく、振動実験や野外火災実験に支障を来し、これを如何に打開するかが課題とされていた。こうした中において、わずかに本館４階の増築と国際地震工学部の増設が行なわれた。

この様に、当時の研究所の規模は、欧米の研究所に比して遥かに小さく、英国の建築研究所（ＢＲＳ）：総員６００名、フランス建築研究所（ＣＳＴＢ）：総員３５０名に比して、日本の国力に比して寡少であり、先進国のレベルに劣るものとなっていた。

この間における建研の敷地規模および建屋配置の変遷は、図１－４に示す如く、敷地規模については昭和２０年：3.8 ha、２４年：2.3 ha、２８年：2.2ha ４１年には、2.1ha と漸減したが、増築の結果、建築面積は 1.04ha となり、写真１－１に見られる様に建て詰まりの状況となり、これを如何に打開するかが課題とされていた。

この様に当時の研究所の規模は、英国・建築研究所（ＢＲＳ：総員６００名）、フランス・建築研究所（ＣＳＴＢ：総員３５０名）に比して、定員と施設規模において、これら先進国との差は大きく、国力に比して劣るものであった。

２）筑波研究学園都市への移転要請

こうした中で、首都・東京における過度の人口・産業などの集中を緩和する方策が国により検討された。研究学園都市の開発構想は、昭和 35 年の所得倍増計画を基調とした首都改造計画論議により始まり、当時の日本経済の高度成長に対応する大型プロジェクトとして、東京遷都論、東京湾埋め立て案などが世論を賑わしたが、政府は首都圏整備委員会を中心に、大学や官庁の集団移転を軸とした新都市建設の検討を始めた。

こうして、東京の既成市街地にある政府関係機関の分散策が採られることとなり、昭和３６年には、「官庁の移転について」が閣議決定され、その後、これに対応する新都市建設の目的として、政府関係試験研究機関の首都からの分散（集団移転）により、首都の過大化防止に寄与すると共に、試験研究機関の研究体制を刷新・向上させる事となり、昭和３８年には、「筑波研究学園都市の建設」が閣議で了解され、東京の既成市街地にある、国立の教育・研究機関

などの筑波への移転が決定され、建設省付属の建築研究所、土木研究所および

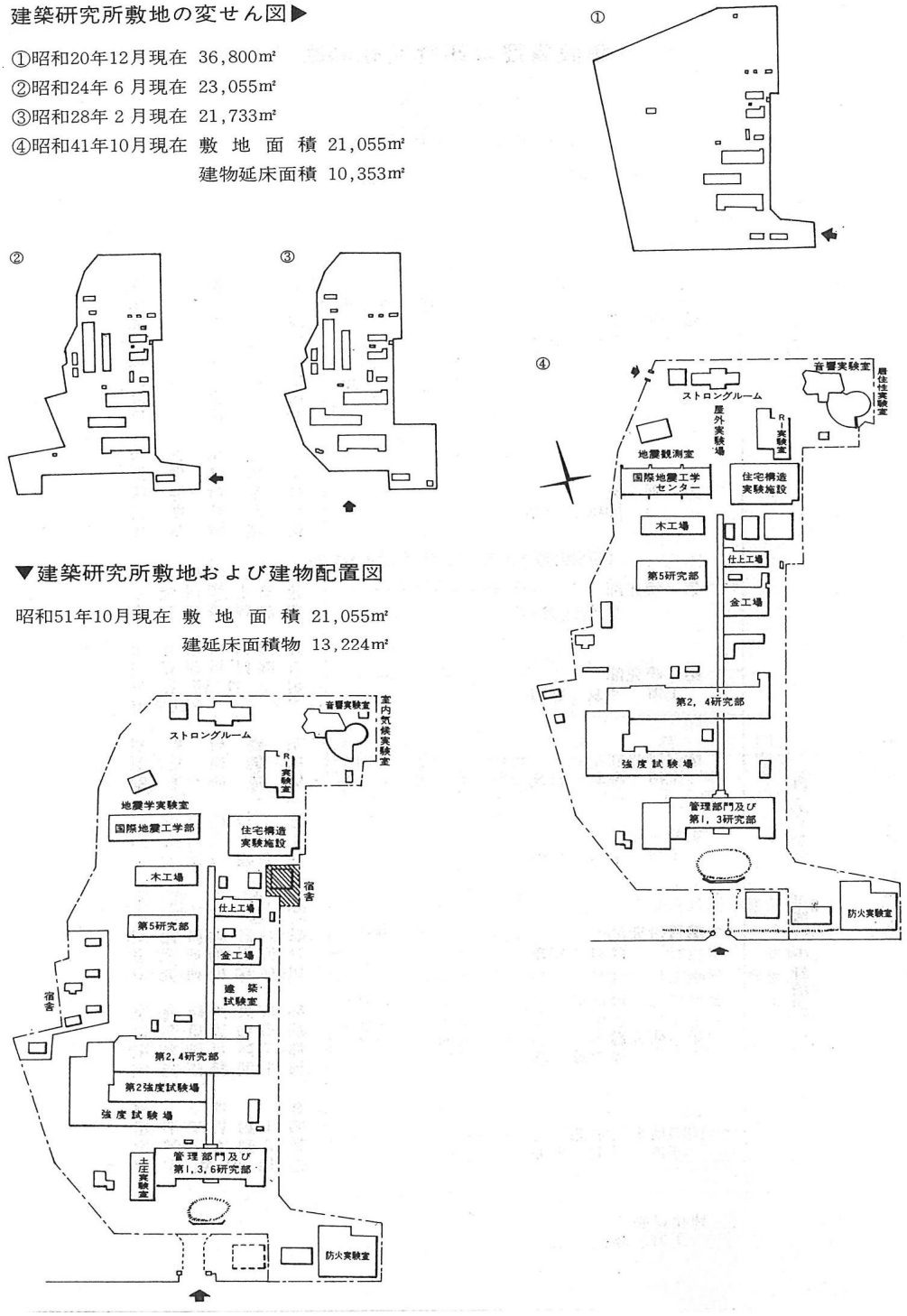
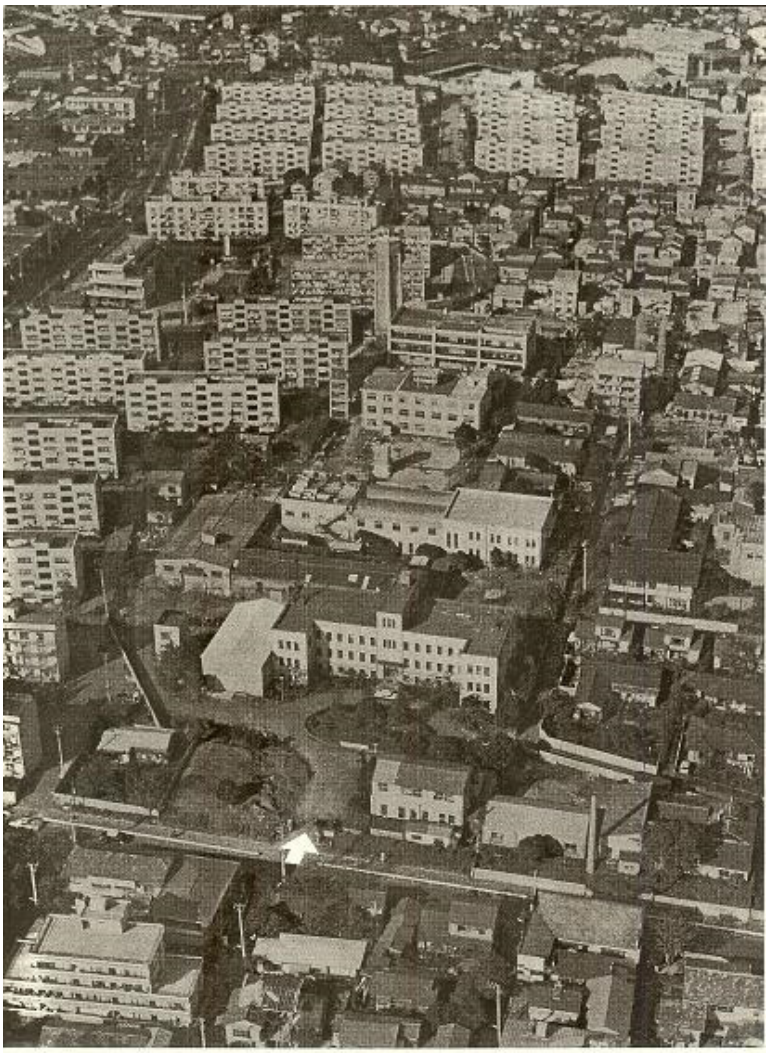


図１－４　建築研究所・敷地の変遷と建物配置[*4]



建築研究所の鳥かん（日刊建設工業新聞社提供，昭和41年11月３日撮影）

写真１－１　建築研究所全景（昭和４１年）[*5]

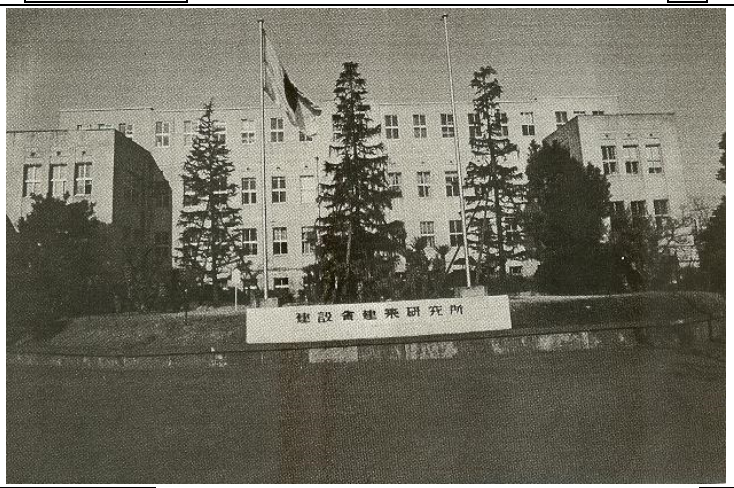


写真１－２　建築研究所本館正面（昭和５１年）[*6]

国土地理院の三機関もこれに含まれることとなり、建研は懸案の組織・定員の拡充を行い、新たな研究施設を建設する絶好の機会を得ることになった。

（２）建研の将来構想と筑波新施設整備の経緯

１）建築系研究部門の飛躍的な拡大と都市計画部門の独立への期待

上記の如く、昭和３８年の閣議了解により、同年には、研究学園都市への移転対象とされる研究機関に対し、「移転に伴う研究所および研究者に関わる一般的な問題」に対する研究所の意向を建設本省に提出した。

昭和３９年には、建研は、研究所の組織体制の将来構想として、総務部、企画室、技術管理部および建設経済、建設材料、建築構造・土質、建設施工、防火・設計計画、建築設備の６研究部門の他、国際地震工学研修所から成り、 西欧先進国の追随を許さない程の、最新鋭かつ大型の構造実験棟や火災実験棟などの施設を擁し、総員４００名から成る建築研究部門の飛躍的な拡大を目指す構想を作成した。

そしてこれとは独立して、８０名を擁する都市計画研究所の創設構想を策定し、昭和４０年に建設省本省技術調査室に提出した。(図１－５)

前述のように、建研は創設以来、都市計画に関する調査・研究を実施して来たが、実験や試験などを行う、建築部門の所謂ハード面を主とする研究とは異なり、都市計画に関する調査や文献などの、ソフト面の情報を研究の基盤としており、それらの情報源である、都市行政や統計部局および学・協会や、専門図書館などとの日常的な接触が不可欠であり、これらの諸機関の所在する都心部へのアプローチが生命線である事から、都心を遠く離れた場所への移転は、研究活動を著しく不利なものとする事になるとの判断から、建研から分離して都内に残留・独立させる構想としたものである。

その後、昭和４２年度には、移転予定の研究機関に対して、移転計画の検討・策定に充てる「移転関連経費」が計上され、４２年８月には２０ヘクタールの敷地要求が行なわれた。これに次いで同年９月には、「研究学園都市の建設について」の閣議了解により、建設省付属三機関を含む３６機関の移転が決定した。

その後、昭和４６年６月には、第一研究部・入澤恒部長の下で、「都市・住宅に関する国立研究機関の設置について」と題する提案が起草され、都市・地域計画部門では、都市経営、基本計画、都市設計、都市施設の４研究部、また、住宅部門では、住宅計画、土地・住宅経済、住宅生産の３研究部からなる、「都市住宅研究所」の設置構想案がまとめられた。(図１－６)

所長	1名
総務部（総務課、会計課）	54名
企画室：研究調整、研究情報、広報活動、技術指導	20名
技術管理部（工務課、共同実験場、電子計算機室）	35名
第1研究部（建設経済関係4研究室）	40名
第2研究部（建設材料関係5研究室）	50名
第3研究部（建築構造　土質関係4研究室）	45名
第4研究部（建築施工関係4研究室）	45名
第5研究部（防火、設計計画関係4研究室）	40名
第6研究部（建築設備関係4研究室）	30名
国際地震工学研修所	30名
（小計	400名）
都市計画研究所	80名
総計	480名

図1－5　建築研究所の組織体制の構想（昭和39年の構想）[*7]

所長・副所長
総務部（総務課、人事厚生課、文書課、会計課）
企画部（企画課、広報課、情報・管理課－資料室、計算室－）
研究部
　都市経営研究部（開発経済、都市経営、都市制度の3研究室）
　基本計画研究部（地域計画、都市基本計画、都市解析、都市環境の4研究室）
　都市設計研究部（都市開発、都市再開発、都市デザイン、都市防災の4研究室）
　都市施設研究部（交通計画、交通施設、交通公害、公園緑地、文教・社会施設、下水道、供給・処理施設の7研究室）
　住宅計画研究部（住宅計画、住宅需給、住居基準、住生活の4研究室）
　土地・住宅経済研究部（住宅経済、宅地経済、住宅経営、土地制度の4研究室）
　住宅生産研究部（住宅産業、生産組織、生産技術、コスト、の4研究室）

職員総計　317名
1研究室の標準組織は、室長1名、研究員3名、研究助手4名の計8名とする。

図1－6　都市・住宅研究所組織（案）（昭和46年6月）[*8]

2）筑波移転と新施設の整備の経緯（年表１－１）

その後、国の筑波研究学園都市建設に向けた計画が進み、昭和４３年５月には、研究学園施設用敷地：１４９８ヘクタール、うち建設系団地の敷地：約１３８ヘクタールが推進本部により決定された。これに基づき、同年８月には、建研移転の年次計画を提出した。

翌４４年６月には、４７年度までに建設系３機関を含む、１１機関の建設工事に着手する旨の閣議決定が行なわれ、いよいよ、筑波移転の日程が決まり、同年１０月には、建研の所内に、具体的な移転実行計画を検討する為に、各部・課の代表１１名による「研究学園都市委員会（ＫＧ委員会）」が発足し、最終的な施設計画などに関する検討が開始された。

翌、昭和４５年には、まず、昭和４３年の提案による、面積：１２ヘクタールの、基本計画第１次案がまとめられ、これに基づき、最初の研究施設として、「ばくろ試験場」が起工された。

これに次いで、昭和４６年には、敷地面積２１ヘクタールの、基本計画第２次案がまとめられ、同年１１月には、以下のような「移転計画作成方針」が示された。

ア）研究の質の低下・阻害をきたさないスムースな移転。

イ）職員のオーバーワークとならない移転。

ウ）研究学園都市公共施設等の整備に合わせた移転。

次いで、昭和４７年前半には、棟別の施設計画を含む基本計画第３次案がまとめられた。（図１－７）

こうした経緯を経て、昭和４８年１月に大蔵省は、建研の新施設の規模を48,000 ㎡とする事を了解した。その結果、建研は同年３月の基本計画第４次案において、懸案の大ストロング棟の建設および、都市研究所の分離案を取り下げた。（図１－８・図１－９）

これを受けて、推進本部は建研用の敷地は２１ｈａとする事を最終的に決定し、昭和４８年１１月に建設工事を発注して工事を開始した。なお、同年度に筑波大学を新構想大学とし、筑波研究学園都市に移転する事を決定した。

建研の筑波施設はその後、昭和５０年１月には、本館、小スト、実大火災実験棟などを起工した。なお本館の完成は、オイルショックによる緊縮財政座の下で、５１年度から５２年度に変更された。その後、５０年８月には先行して建設が進められた「ばくろ試験場」が完成した。

一方、この間、昭和５０年３月には、昭和５３年度に移転機関の移転を行う事が閣議で決定され、建設省付属三機関は、昭和５３年度に概成移転する事となり、最終的な建設計画がまとめられた。（図１－１０）

こうして筑波施設の建設の進められる新たな時代に対応しつつ、昭和５１

年表１－１　筑波移転の経過[*9]

年次（昭和）	政府の動向	建研の動向	社会状況
３６年６月		都市・住宅に関する国立研究機関の創設提案	
９月	閣議決定「官庁の移転について」	(36/6)	
	首都への人口過度集中の防止のため		
	既成市街地に置く事を要しない官庁		
	（付属機関・国立の学校）の集団移転		
	について検討する。		
３８年９月	閣議了解「研究学園都市の建設について」		
	１．筑波地区に４０００ha 規模で建設		
	２．用地の取得・造成は日本住宅公団に		
	行なわせる。	筑波移転に関する一般的問題を本省に提出	
	３．東京教育大学を筑波に移転	筑波移転計画を大蔵省主計局に説明	
３９年３月	茨城県及び６町村が学園都市の建設に協力		
	する事を表明		
１２月	閣議了解「筑波研究学園都市の建設について」	建研の将来構想を策定	
	１．４０年に着手し、１０か年で完成する	建築系６研究部・国地部（４００名）	
	２．総理府に「研究学園都市建設推進本部」	都市計画研究部（８０名）	
	を設置し、新都市の建設、調整、推進に		
	当たらせる。		
４０年			
１２月	閣議決定「推進本部の設置」	建設省・技術調査官に上記の将来構想を提出	

４２年９月	閣議了解「研究学園都市の建設について」	建研敷地要求（１６＋４ha）、 ＫＧ（研究学園）移転対策班を設置(42/8)	
４３年５月	「推進本部決定」 学園用地：１４９８ha 建設系用地：１３８ha	建研施設建設年次計画提出（43/8）	大学紛争
４４年６月	「閣議決定」４７年度までに建設系三機関を含む１１機関の建設に着手する。 建研は現員（１８２名）で移転する。	研究学園都市委員会（ＫＧ委員会）発足 （各部課代表１１名）(44/10)	
４５年	筑波研究学園都市法公布	建研基本計画第１次案、　ばくろ試験場起工	
４６年		建研基本計画第２次案	
４７年		建研基本計画第３次案	
４７年		ＫＧ推進本部設置	
４８年１月	「大蔵省了解」建研施設全体計画（48000 ㎡）		第１次オイルショック
４月	「推進本部決定」建研敷地面積（２１ha）		大型公共事業凍結中止
１１月	建研施設の工事着工		筑波大学開学
５０年３月	「閣議決定」５４年度を目途に移転を行う。 建設三機関は５３年度に概成移転する。		
７月		ばくろ試験場完成	
５１年		建研将来構想を策定 建築系７研究部および国地部（３００名） 都市計画研究部（１００名）	
５４年２月～３月		建研施設概成・移転	
５４年４月		建研研究業務を開始	

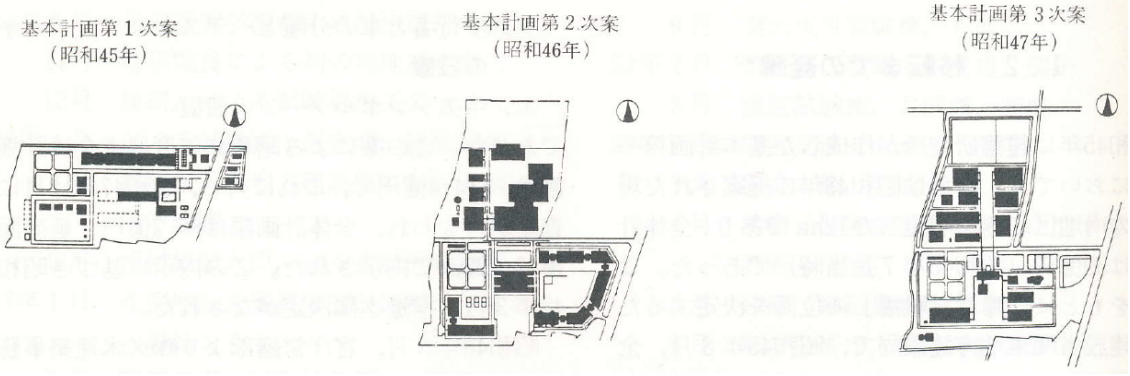


図１－７　基本計画第１次～第３次案[*10]

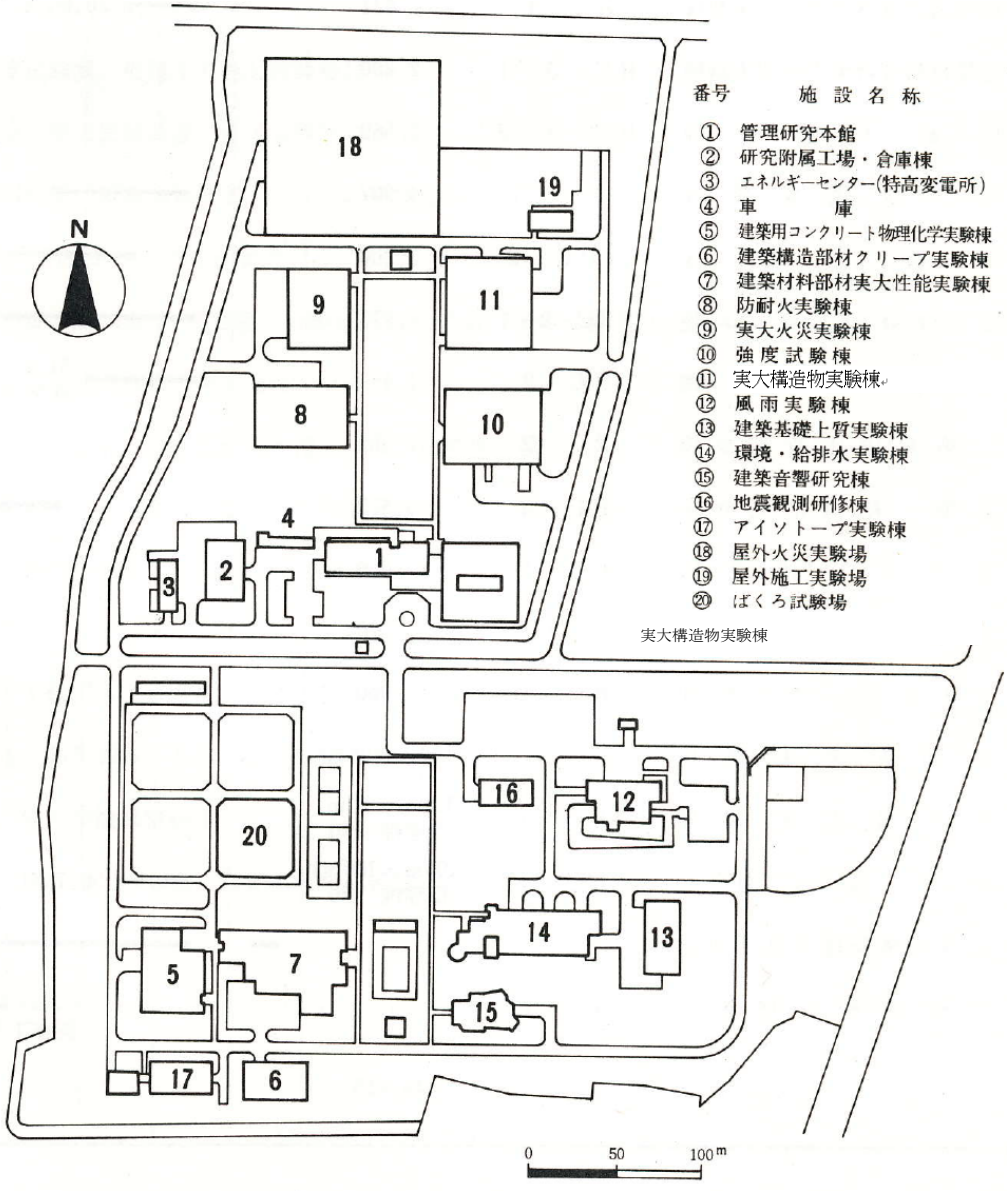


図１－８　筑波研究学園都市における建築研究所配置図[*11]

	施設名称	構造	延面積(m²)	工程計画(年次) 48	49	50	51	52	53
1	管理研究本館	SRC−7−1	13,313	エネルギーセンター				52.3	53.3
2	研究附属工場・倉庫棟	RC−1	1,250						
3	エネルギーセンター（特高変電所）	S−1	562				51.3完成		
4	車庫	RC−1	200						
5	建築用コンクリート物理化学実験棟	RC−2	2,900						
6	建築構造部材クリープ実験棟	RC−1	834			50.6完成			
7	建築材料部材実大性能実験棟	RC−S−1	2,200						
8	防耐火実験棟	RC−S−2	2,560						
9	実大火災実験棟	SRC−7	4,907				51.9完成		
10	強度試験棟	SRC−2	3,200					52.3	
11	実大構造物実験棟	SRC−8−1	7,215						53.3
12	風雨実験棟	RC−2	1,462				51.3 風洞完成		
13	建築基礎土質実験棟	RC−2	1,050						
14	環境・給排水実験棟	RC−6	3,872						
15	建築音響研究棟	RC−1	600						
16	地震観測研修棟	RC−1	570						
17	アイソトープ実験棟	RC−1	950						
18	屋外火災実験場		(100m×90m)						
19	屋外施工実験場	RC−1	(100m×50m) 管理棟 450						
20	ばくろ試験場	RC−1	(100m×100m) 管理棟 320			50.7完成			
21	構内環境整備等共用施設	1式							
22	実験排水処理施設	1式					調査工事		
	合計		48,415						

図１－９　筑波研究学園都市における建築研究所の建設計画[*12]

年、改めて今後に向けた「建研の将来構想」を練り直し、建築研究部門７研究部（定員３００名）および、都市計画研究部門（定員１００名）の規模を有し、両部門を一体の研究所とする将来の拡大策を取りまとめた。

かくして、現体制を維持しつつ、筑波に移転する事となった建研は、改めて各般にわたる最終的な移転準備を始めた。

所長
研究調整官
総務部（調査官、総務課、会計課）
企画部（企画調査課、技術管理課、海外研究協力室）
基準・認定部（管理室、建築試験室、建築基準室、建物診断室、資料室）
総合研究官
材料・施行研究部（5 研究室）
構造研究部（5 研究室）
生産・計画研究部（4 研究室）
環境研究部（4 研究室）
防火研究部（4 研究室）
設備研究部（4 研究室）
国際地震工学研修センター（6 室）
小計・約 300 名
都市・住宅研究所（所長、総務部、企画部、総合研究官
経済・社会研究部、住宅・宅地研究部、都市防災研究部、
都市計画研究部等）　約 100 名
－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－
総計・約 400 名

図１－１０　建築研究所の組織体制の構想（昭和 51 年の構想）[*13]

３）筑波の新施設による研究活動の展開への課題

ア）研究支援要員の確保

上記の様に、筑波新施設の建設が進められたが、旧施設に比して、新施設の用地は１０倍、床面積は４倍と、飛躍的に増大し、新たに建設される諸施設は、８階建ての試験体のテストが可能な、世界最大の実大構造物実験棟を始めとして、新鋭かつ大規模な実験施設を擁し、これらを現在の定員により運営することは、かなりの負担であり、特に大規模の実験に際しては、都内の大学の卒論生などの支援を必要とし、交通条件の不利な筑波において、そ

の支援をどう補うかが課題とされ、また、最新鋭の装置を用いて、それを有効に活用する為には、外部からの技術者を活用することも必要であることが問題となった。

これは、筑波施設を有効に活用して研究成果を挙げる為の必須条件とされることから、部外からの研究員を受け入れ、研修を行いつつ研究支援要員として活用する制度の創設が検討された。

イ）研究施設の有効利用と維持・管理費の確保

つくばにおける最新鋭かつ大型の実験施設を用いた研究活動を展開するには、それらの施設を有効に活用し、その運転や維持管理に要する費用を調達する事が必要とされる事が大きな課題と考えられた。

これに対応するには、従来からの公的な機関から受託して行なう試験・研究制度のみでは不十分であり、民間との共同研究を行い機材の有効利用を図る新たな制度の創設についての検討が進められた。

（３）移転準備業務の経緯

筆者は、第六研究部(都市計画)・都市開発研究室長から、この動きの中で、昭和５１年４月から２か年間、所の企画室長としての重責を担い、筑波移転への最終準備の重要な役割を務めることとなった。以下にその業務の概略を記すことにしよう。

１）研究用の機器・装置および研究資料の整理・選別

ア）研究用機器および資料の移転・廃棄の選別

イ）保存すべき試験・実験用の機器・装置および研究資料の確定

これらに関する検討の結果、各研究部門において、移転後の利用が可能であり、かつ移転可能な機器・装置などが分別されリストアップされた。また、日本最初のアムスラー材料強度試験機および、建研独自の開発・製作による加力装置、液状化試験装置などを移転し保存する事となった。

また、各部門における試験・実験・調査などの記録および研究成果と収集資料、および保存すべき文献や調査・統計資料などの整理とリストの作成が行なわれた。

２）筑波における研究環境の変化と対応方策の検討

ア）研修・宿泊施設の確保

建設省付属三機関は、国内各地の関係機関への技術指導や研修などを行い、

また、大学・研究機関の専門家の来訪や、建築研究所の国際地震工学研修の外来講師用の宿舎の確保は不可欠であることから、三機関共用の研修・宿泊施設を確保する事となり、三機関の近辺に「研修施設」を整備する事とした。但し、その規模は限定的であり、大規模実験を支援する大量の要員は、研究所の近辺の旅館などを充当する事とした。

イ）交通条件の整備

東京方面からの通勤者および外来者の交通問題に関しては、移転当時に東京方面とつくばを結ぶ既存の交通機関は、ＪＲ常磐線のみであり、上野から最寄駅の荒川沖駅または土浦駅に降り、タクシーかバスに乗り継いで、建設省関連の三機関に至る経路のみで極めて交通の便は悪く、唯一の交通手段である常磐線経由の通勤は成り立たず、また外来者もタクシーによる他なかった。

これは三機関共通の大きな課題として取り組み、荒川駅にバスベイを確保し、学園都市の中心部を経て最北部にある三機関にサービスするバス路線の開設を関係各方面に働きかけた。やがてこの唯一のバス路線が確保され、朝夕と昼の一日数本の運行が行なわれる事となった。

ウ）住宅・都市計画資料室の整備

また、ソフト部門の研究のハンディキャップを少しでも解消すべく、関係部門において検討の結果、手元に関連する図書や基本的な統計資料、行政部局による全国的な、都市計画基礎調査や住宅統計調査および各種の都市交通量調査などの情報源となる資料を収集・蓄積するための「都市・住宅資料室」の整備を行う事とし、この為のスペースの確保と、大量のデータを呼び出し、これを用いてリアルタイムで分析し、計画シミュレーションを行う「都市計画シミュレータ装置」や、住宅計画などを検討する「カラー画像シミュレート装置」を整備する事になり予算措置が講じられた。

３）移転困難者対策、移転後の生活環境および研究環境対策。

ア）移転困難者対策：夫婦共稼ぎなどの家庭的な状況、および大半の、研究助手は、都内の大学の夜間コースに就学していたが、筑波には大学の夜間コースが無く、これらの所員への対応を図る事が必要であった。

これについては、まず家庭的な事情で移転困難な職員に関しては、個別にヒアリングを行い、転職先の選択・斡旋を行って解決した。

また、助手の転職先に関しては、建設省本省官庁営繕部の極めて好意的かつ寛大な配慮により、多くの助手諸君が、専門分野の身近な職場への配置替となり、極めてスムースに対応する事が出来た。

イ）職員とその家族の移転後の生活は、貸与される住宅の居住性に大き

く左右され、保育園、幼稚園、義務教育施設や病院、購買施設、公園、運動場などの基礎的な生活関連施設の整備状況が問題となり、これは職員労組を通じて、移転の基本的な前提条件とされた。

職員の住環境への対応に関しては、まず引っ越し先の公務員住宅への入居が課題とされた。当初の研究公務員用の住宅は、規定よりゆとりのある規模の中・低層の住宅を準備する事とされていたが、予定量の用地取得が困難となり、一部に高層住宅が配置されることになり、「郊外での高層住宅への居住は困る」との職員組合を通じての強い要望があり、これを国土庁に置かれた「推進本部」と折衝したが、「極力、高層住宅の戸数を減らすと共に、高層住宅の配置に際しては環境上に十分な配慮を行なう」旨の回答を得た。

かくして建研に与えられた住戸は、階高や、居住環境、社会施設、交通条件など様々であり、これを公開して職員の希望する住戸を選ばせ、複数の希望者がある場合は、希望の変更や抽選により割り当てる方法が採られた。幸い高層住宅を希望する家庭もあり、住宅の割り当てはさした問題もなく解決された。

ウ）職員の就労環境と健康管理への配慮は、移転による環境の激変と共に極めて重要であり、筑波新施設の緑化や休息空間、また福利厚生施設やスポーツ・レクリエーション施設の配備への配慮が不可欠とされた。

研究所の防災、環境、景観などへの配慮から、各研究所は、敷地の緑披率を３０％以上とすることが目標とされたが、建研はその基準を上回り 40.5％の緑被率を有する計画とし、敷地の外周部を緑化し、アプローチ道路の並木を整え、構内も植栽や芝地を配し、さらに、野球場、テニスコート（３面）、プール（２５メートル）、などのスポーツ施設を配するなど、研究環境と健康維持のための空間を十分に配する形とした。また研究者の健康管理の為に診療室が設けられ、健康相談に応じる体制が採られた。

４）建研の国際活動の進展と旧庁舎の見納め会

ア）国際建築情報会議（ＣＩＢ）及び、国際材料構造試験研究機関連合（ＲＩＬＥＭ）への参画

ＣＩＢは、建築の研究、調査、応用およびこれらの情報に関する国際協力を奨励促進することを目的として、１９５７年に設立され、建研は日本の代表委員として毎年開かれる総会に出席して来たが、昭和５２年に日本で初めて理事会を開催した。

また、建研はかねてよりＲＩＬＥＭにおける種々の技術委員会に参加し、その活動に貢献している。これらは、建研の筑波移転後の、国際活動の基盤

になった。

イ）天然資源の開発利用に関する日米会議（ＵＪＮＲ）

ＵＪＮＲ・耐風耐震専門部会は、すでに１９６９年に第１回合同会議が開催され、以後、毎年日米交互に開催されて来た。

また、ＵＪＮＲ防火専門部会は１９７６年に米国で第１回の合同部会が開かれ、一年半毎に日米交互に開催される。その日本での合同部会の開催に当たっては、建研は日本側の代表機関として、企画室がその連絡の衝に当たり、日本側の関係機関と連携して成果を挙げた。

なお米国側は、日本に比し建築火災における死者数が多い事から、日本の建築火災対策技術の詳細について大きな関心を示した。なおこうした対応は、筑波移転後の国際活動の基盤になった。

ウ）旧庁舎の見納め会

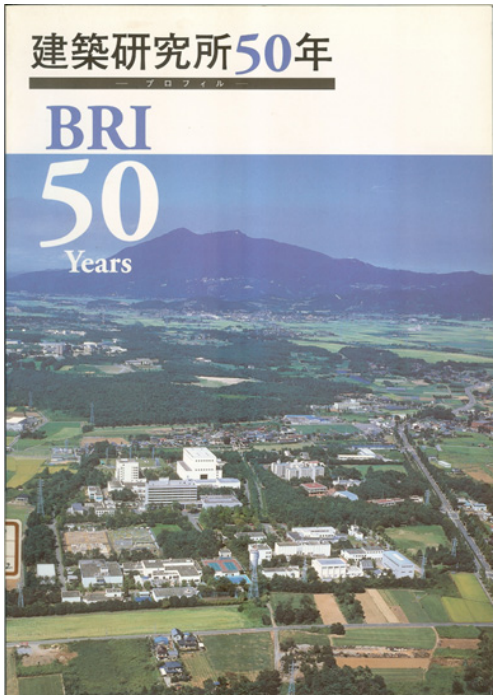
移転準備が進むうちに、先輩や関係者を招き、懐かしの新宿百人町の庁舎と試験・実験用の機器や装置に「お別れ」する会を行うことが企画され、企画室がその衝に当たり、各部と相談してその具体案を練り上げた。

当日は、創設以来の大先輩の諸氏や、学・協会および関連機関からの来賓を迎え、各部において酒肴を用意し、和やかに見納めの話が弾んだ。なお所としての閉所式は、他日、近傍の海洋会館で行われた。

５）筑波における新施設の建設管理

一方、企画室に於いて、小山補佐の率いる技術管理チームは、前述の、如く、筑波に移動する実験装置・機器、また歴史的な保存装置や機器などの選別を行い、既に建設の進んでいる筑波研究学園都市における新施設の建設計画の調整などに当たっていた。

私が企画室長になった昭和５１年４月には、既に、ばくろ試験場はじめ風雨実験棟、クリープ試験棟などが完成しており、建設大臣等の政府要人の視察が相次ぎ、これに対応して、所長はじめ総務部長、企画室長などが、現地で一行を迎えて説明を行った。そうした中で、厳冬の朝、足の踝までが埋まる程の霜の置く中で、震えながら視察の一行を玄関先に迎え、全体計画を説明し、ばくろ試験場などを案内した事が忘れられない。



北東の四半分は
国立教育会館筑波分館

写真２－１　建研筑波新施設全景（昭和 53 年度）[*14]

左手奥：実大火災実験棟
右手奥：実大構造実験棟
手前：管理研究本館
（右手１階：国際地震工学部）

左：実大火災実験棟
右：実大構造実験棟

南側研究施設群の全景
手前左：風雨実験棟
手前中：地震観測研修棟
手前右：展示館
左手奥：建築基礎土質実験棟
中央：環境・給排水実験棟
中央奥：建築音響実験棟

写真２－２　筑波新施設の主要施設の事例[*14]

２．　筑波新施設の整備と研究活動の展開

上記の任務を終えた私は、昭和５２年４月に、第６研究部長（都市計画）となって移転の最終準備に当たり、移転後は５年間にわたり部長として、筑波施設の新たな環境の中で研究業務に当たった。

それは、建研の筑波新時代の幕開けであり、次々に開始される大型プロジェクト研究に取り組み、筑波の新施設をフルに活用し、成果を納めんとして活動を始めた。

（１）筑波新施設における研究の展開 {写真２－１，２－２}

１）建研所掌の全分野における研究の展開

建研は、筑波移転後の新時代（敷地１０倍、建物延べ面積４倍）における、研究活動の飛躍的な展開に向けて、昭和５４年に「長期的な研究の方向と研究の具体的な方策」を策定し研究開発の５大目標として、
①災害の防止、②居住環境の改善、③建築生産の合理化と新技術の開発、④資源・エネルギーの有効利用、⑤国際協力の推進を設定した。

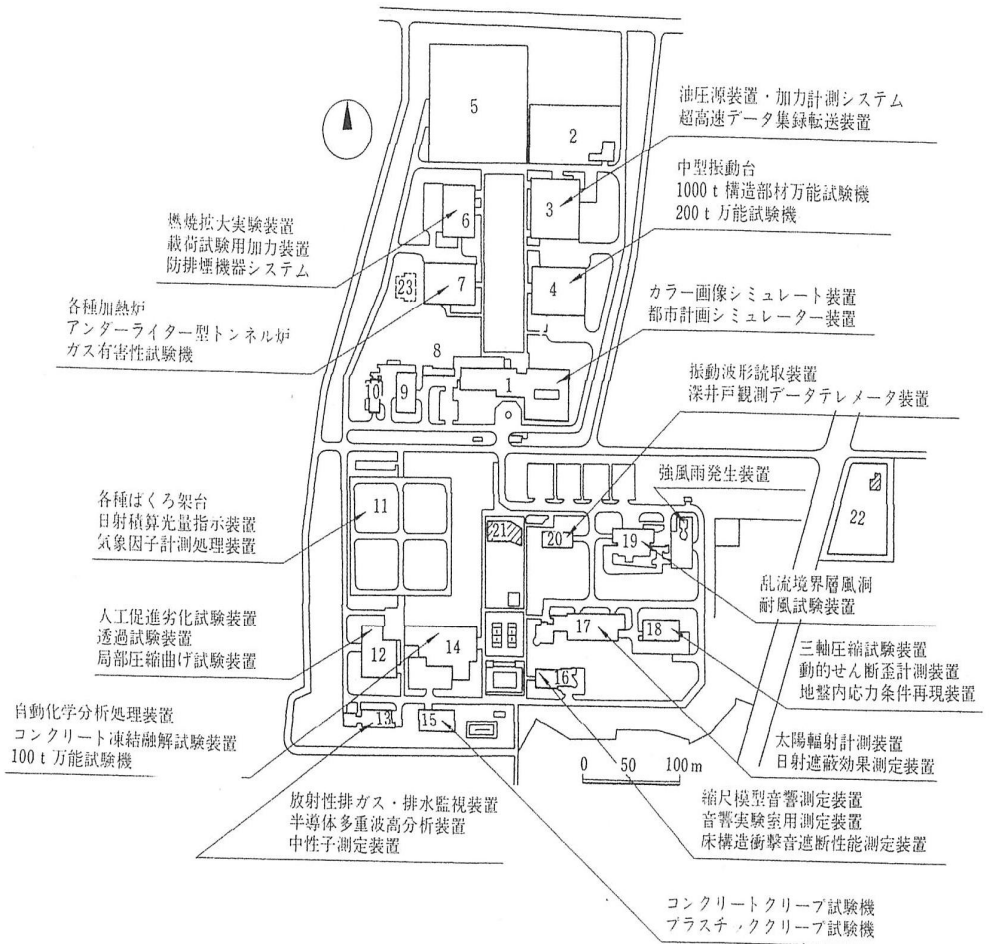
この基本的な方針に従い、全面的に装備された最新鋭の研究・実験施設を活用して各分野における研究を展開した。(図２－１、表２－１)

なお、建研の筑波移転前後における定員と予算を比較すると、移転前の昭和３７年～５１年の間においては、定員は１７５～１８０名、予算は９億円程度であったが、移転後の昭和５５年度には予算は２０億円となり、以後２１億円台（自己予算は１７～１８億円）となり、ほぼ倍増した。しかしながら定員は１８２名であり、増員はないままに研究活動が進められた。(年表２－１)

２）大型プロジェクト研究の展開

前述のように、我が国の経済成長が進展する一方、国民生活に関わる建築や都市における防災、環境および省エネ等の諸問題に対処する為の新技術の開発が要請され、これらに応えるため国の機関が中心となり、大学や民間機関の協力の下に行う、大型の研究・技術開発プロジェクトの時代が到来した。

その先駆けとして、昭和 47 年から 5 か年間に亘り、建設省の総合技術開発プロジェクト（総プロ）「新耐震設計法の開発」が開始され、一方、昭和 50 年より５か年間に亘り、環境庁研究調整費による「都市環境保全計画モデルの開発」が開始され、ここに大型プロジェクト研究の時代の幕が開かれた。



番号	施設名	番号	施設名
1	管理研究本館	13	アイソトープ実験棟
2	屋外施工実験場	14	建築材料実験棟
3	実大構造物実験棟	15	クリープ実験棟
4	強度試験棟	16	建築音響実験棟
5	屋外火災実験場	17	建築環境実験棟
6	実大火災実験棟	18	基礎・土質実験棟
7	防耐火実験棟	19	風雨実験棟
8	車庫	20	地震観測研修棟
9	研究機器試作工場	21	展示館（55年度完成）
10	エネルギーセンター	22	多目的実験棟（57年度完成）
11	ばくろ試験場	23	画像情報棟（建設予定）
12	建築部材実験室		

図２－１　建築研究所主要施設・機器設置状況[*15]

表２－１　建築研究所施設概要説明書[*16]

61.6.1現在

番号	施設名	構造	延面積(㎡)	施工年度	施設内容
1	管理研究本館	SRC-7 B-1	13,355	49～52	管理部門と研究室，標準実験室及び国際地震工学研修所を統合したもので地下に全施設の中央監視室を設けた。
2	研究機器試作工場	RC-2	1,178	53	試験機器の試作，改良，修理，維持，試験体の製作等を行い試験機器の集中管理を行う。
3	エネルギーセンター(特高変電所)	S-2	531	53	全施設の特高変電を行う。
4	車庫	RC-1	192	52	乗用車，マイクロバス，等を収納する。
5	建築材料実験棟	RC-2	3,005	52～53	建築用コンクリートの物性を材料，調合，養生条件等から実験的に解明する他，建築材料・部材の品質性能を物理，化学的に解明する。
6	クリープ実験棟	RC-1	745	49～50	建築材料・部材のクリープ性状を，環境条件を変化させる等して実験的に解明する。
7	建築部材実験棟	RC，S-2	2,070	52～53	建築の部材，接合部，材料等の変形や強度に関する性能，および壁材の結露に関する性能を実験的に解明する。
8	防耐火実験棟	RC，S-2	2,581	52～53	建築材料部材の火災時における性状を実験的に解明する。
9	実大火災実験棟	SRC-7	4,963	49～51	建築物の燃焼拡大等の燃焼機構や煙の流動と制御手法を実大の建物を使って実験的に解明する。
10	強度試験棟	S・RC-3	2,833	49～50	建築構造部材や接合部等の強度試験を行う他，モデル試験体の振動試験を行う。
11	実大構造物実験棟	SRC-8 B-1	7,324	49～52	建築物の耐震性を実大の建物や部材を使った加力試験により，実験的に解明する。
12	風雨実験棟	RC-2 RC-1	1,437 135	53～54	建物や建築部材の耐風設計，建物周辺の風環境，あるいは防水設計に関するデータを風洞や防水試験装置を使った実験から得る。
13	基礎土質実験棟	RC-2	1,053	53	建築物の基礎地盤，土質に関して，実験的な解明を行う。
14	建築環境実験棟	RC-4	3,199	51～53	人間の生理，心理，行動面から見た室内温熱環境，日照等の環境や材料部材の断熱性能等を実験的に解明する。
15	建築音響研究棟	RC-1	647	53	建築物の騒音防止設計，室内音響設計に関する資料や音の伝播性状に関する資料を実験から得る。
16	地震観測研修棟	RC-1	517	53	国際地震工学研修所の観測地震学についての研修及び実習を行う。
17	アイソトープ実験棟	RC-2	1,024	52	放射性同位元素利用施設の遮蔽性能，室内仕上げ材料の汚染防止等に関して，実験的な解明を行う。
18	屋外火災実験場	RC-1	39	53～54	実大住宅及び模型住宅の一戸又は数戸の燃焼，炎上による外部空間への影響を，実験的に解明する。

RC：鉄筋コンクリート造　SRC：鉄骨鉄筋コンクリート造　S：鉄骨造　B：地下

年表２－１　筑波移転前後における組織・定員と予算の変遷[*17]

昭和 26 年~35 年　定員 100~110 名　予算　約 5 億円（人頭研究費）
昭和 37 年~51 年　定員 175~180 名　予算　約 9 億円　人頭研究費＋特別研究費）
昭和 42 年~筑波研究学園都市への移転関連経費を計上
昭和 47 年度以降、建設省建設技術開発経費を計上
昭和 50 年度には、所外予算が自前の予算の 1.8 倍に達する
昭和 52 年、総務部、企画部、六研究部、国地部、試験室体制となり、定員１８２名
昭和 52 年以降、所外予算が急増　(建設本省・科技庁・環境庁・国土庁など)
昭和 53 年度、予算 28 億円（内、内部固有の予算は 24.5 億円）
昭和 55 年度、予算 20 億円となり以降、21 億円台（固有の予算 17~18 億円）
　以降横這いとなり人件費の漸増により研究費は圧迫される。

年表２－２　大型研究・調査活動展開の事例[*18]

昭和 47 年~総プロ「新耐震設計法の開発」(5 か年)
昭和 50 年~「都市環境保全計画モデルの策定と応用に関する研究」(５か年)
昭和 50 年　ＵＪＮＲ防火専門部会発足、複合材料の燃焼性状などの共同研究
昭和 51 年、建基法の改正に伴い住宅性能の評価基準・測定試験法の開発
昭和 52 年~総プロ「都市防火対策手法の開発」(5 か年)
昭和 52 年~総プロ「省エネルギー住宅システムの開発」(5 か年)
昭和 52 年　ＣＩＢ理事会を東京で開催　(昭和 34 年加盟)
昭和 54 年、「長期的研究の方向と研究の具体的方策」を策定（研究開発の５大目標*）
昭和 54 年 3 月、筑波研究学園都市に移転（敷地：10 倍、建物延べ面積：4 倍に）
昭和 57 年~　建築物の合理的な総合防災設計法に関する研究開始
昭和 61 年~総プロ「新木造建設技術の開発」(５か年)
昭和 61 年　国際地震工学部研修生(セミナーを含む)：合計 50 の国・地域より 610 名
昭和 62 年~高齢化社会への対応に関する研究開始
昭和 63 年~総プロ「鉄筋コンクリート造建築物の超軽量・超高層化技術の開発」(５か年)
　＊：①災害防止　②居住環境の改善　③建築生産の合理化と新技術の開発
　　④資源・エネルギーの有効利用　⑤国際協力の推進

これらの成果は、1981 年の建築基準法施行令の改正による「新耐震設計法」や、環境保全の為の都市計画の策定方法の基本として活用された。

これに次いで、昭和 52 年より「都市防火対策手法の開発」（建設省・総プロ、5 か年間)、昭和 53 年より「省エネルギー住宅システムの開発」（建設省・総プロ、5 か年間)、が次々に開始され、いよいよ、筑波新施設をフルに活用しての研究・技術開発が進められた。

この間､これらの大型プロジェクトを行う為の所外（建設本省､科技庁、国土庁など）からの予算は昭和 52 年度以降急増した。

大型・研究技術開発プロジェクトは、昭和 57 年～「建築物の合理的な総合防災設計法の開発」、昭和 61 年～総プロ「新木造建設技術の開発」、昭和 63 年～総プロ「鉄筋コンクリート造建築物の超軽量・超高層技術の開発」など、いずれも 5 か年間のプロジェクトとして次々に開始され、筑波の新施設がフルに活用された。(年表２－２)

（２）移転後における国際研究・研修活動などの展開（年表２－３）

前述の如く、昭和 50 年に開始された「天然資源の開発利用に関する日米会議（ＵＪＮＲ）第 2 回防火専門部会は、昭和 52 年に日本で開催され、さらに同年、世界建築研究機関会議（ＣＩＢ）の総会が初めて日本で開催され、これらが本格的な国際化時代の幕開けとなった。

筑波移転後の昭和 54 年 8 月に「ＣＩＢの防火シンポジウム」が開催され、同年には、「日米・大型耐震実験研究」が開始され、世界最大の 8 階建ての実大実験が可能な小ストロング棟の、疑似動的加力システムを用いて、鉄筋コンクリート造（ＲＣ)、鉄骨造（Ｓ)、組石造、およびプレキャスト造などの各種の構造物の実大実験による研究が開始され、これには、米国から試験体の作製費などが用意され、まさに世界に誇る建研新施設を用いた国際的な共同研究の先駆けとなった。

さらに昭和 59 年には、国際材料試験連盟（ＲＩＬＥＭ）の総会を筑波で開催し、益々国際的なレベルの新施設を有する研究所としてのステータスが示された。また、最新鋭の試験・実験施設を海外研究機関の研究者の利用に供する事例も増大した。

これらに加えて、既に百人町において 10 年間の実績を有する「国際地震工学部（ＩＩＳＥＥ)」における、世界の地震帯に位置する諸国や地域からの、地震学および地震工学に関する研修活動は、国際的に極めて高い評価を得ていたが、筑波新施設における最新の研修施設における研修活動が、さらに内容を充実して展開された。

そして、世界各地で発生する大地震の被害調査と耐震指導が展開され、昭和 55 年の「アルジェリア地震」、同 60 年の「メキシコ地震」などに際し、建研の専門家が数多く派遣され、国際貢献に寄与した。(表２－３)

年表２－３　国際活動の展開[*19]

昭和５４年８月　「ＣＩＢ防火シンポジウム」つくばで開催
昭和５４年～　日米共同大型耐震実験研究開始（ＲＣ，Ｓ，組石造、プレキャストなど）
昭和５５年　アルジェリア地震調査
昭和５５年～国際地震工学部：「地震工学上級者セミナー（１か月)」を隔年に開催
昭和５７年～　同　　上　　：ジャカルタ・バンドンにおける地震防災の第三国研修
　　インドネシア建築研究所（バンドン）に耐震設計の専門家派遣
　　アジア工科大学（タイ・バンコック）に都市計画部門の教官を派遣
昭和５９年１０月　「ＲＩＬＥＭ総会」つくばで開催
昭和６０年　メキシコ地震調査
昭和６０年～共同研究「インドネシアの住宅・都市開発」実施（２か年）
昭和６１年～「日本・ペルー地震防災センター」プロジェクト(５か年)
平成　２年～「メキシコ地震防災センター」プロジェクト（７か年）
　　「インドネシア人間居住研究所」バンドン市東郊への移転拡充
平成　４年、国際地震工学研修３０周年研修 IDNDR 地震防災技術国際シンポジウム開催
平成　５年、第６回建設材料部材の耐久性に関する国際会議開催
平成　５年～「トルコ地震防災センター」プロジェクト（５か年）
　　［インドネシア集合住宅適正技術開発］プロジェクト（５か年）
平成　７年、「欧州連合共同研究センター情報システム安全研究所」との研究協力協定締結

海外渡航者数（ＪＩＣＡ派遣専門家を含む）の増大

昭和 45 年~52 年：各年 8~18 名、　昭和 53 年~57 年：各年 23~34 名、
昭和 58 年~60 年：各年 36~54 名

海外よりの受け入れ研究員の増大

昭和 51 年～56 年：年間 2~6 名　　昭和 57 年～60 年：年間 7～14 名

国際地震工学部における研修生

昭和 61 年：昭和 37 年より開始した地震学および地震工学コースに参加した研修生は、上級セミナー参加者を含み、50 の国と地域より、合計 610 名となる。

（3）筑波新施設における大型実験の実施　（年表２－４）

筑波移転を契機に各種大型実験施設が整備された。移転前後の頃は大型実験の研究需要が予測出来ずに、活発な施設の利用を疑問視する向きもあった。しかしながら、移転後は、構造、材料及び火災部門を中心に大型研究実験が間断なく実施され、このような研究需要が維持される見込みとなった。

最近各実験棟で実施された大型実験の例を年表２－４に示した。研究費では、総プロ等のように、年間予算が 1,000 万円を超す大型プロジェクトに組み込まれた実験が多い。

この種の大型実験は、国立研究機関としての当所が国費を投じて行うべき性格のものが多く、これに必要な施設や予算を拡充整備する事が今後も引き続き必要とされよう。

関連実験棟	実験研究タイトル	研究期間	担当研究部	関連予算
実大構造物実験棟	高層（6～8階建て）現場打ち壁式構造の耐震性能実験	54.4～54.11	第3部，4部	受託研究
	擬動的手法による日米共同耐震実験（RC，S，RM造）	54.4～(63.3)	3部，4部	国際共同研究
屋外火災実験場	都市施設等の構成による延焼遮断効果に関する実験	55.3	6部	総プロ
	鋼構造建築物の耐火設計法開発のための実大家屋火災実験	61.3	5部	総プロ
強度試験棟	家具の耐震実験	57.2	国地部	受託研究
	建築配管用銅管の振動実験	58.8	4部	同
	杭頭接合法に関する実験	59.8～59.12	3部	共同研究
実大火災実験棟	煙の流動実験	59.3	5部，2部	一般研究
	都市施設等による延焼遮断効果に関する模型火災実験	56.1～56.3	5部，6部	総プロ
多目的実験場	建築解体古材を用いた木造建築物の建設実験	59.12～61.3	4部	総プロ
大分県佐賀関町	実家屋による大規模火災対策に関する調査・実験	54.8	6部	国土庁予算
静岡県大井川市海洋技術総合研究施設	各種建築材料の実海域ばくろ試験	59.11～(62.3)	2部	総プロ

年表２－４　代表的な大型実験研究[*20]

（4）共同研究の前進および受け入れ研究員の活用（年表２－５）

筑波に整備された最新鋭の新施設を広く民間等にも開放し、有効に活用するため、それまでは、地方公共団体や特殊法人からの受託による研究や試験に限定されていた施設の利用の対象を、公益法人に拡大し（５５年)、さらに６０年以降は民間企業との共同研究も可能となり、６１年以降には官・民連携研究の予算化の道も開かれ、建設省総合技術開発プロジェクトに継ぐ大型研究が開始され、広く社会に拓かれた施設としての活用が行なわれるようになった。

年表２－５　共同研究の前進および受け入れ研究員の活用[*21]

１）共同研究の前進
昭和５５年以前：地方公共団体、特殊法人からの受託研究と受託試験に限定されていた。
昭和５５年　「共同研究規則（建設省告示)」で公益法人を含むに拡大。(毎年１０数件)
昭和６０年度以降：民間企業との共同研究も可能となる。（３０件以上となる）
昭和６１年度：官民連帯研究の予算化始まる。総プロに次ぐ大型研究開始。
２）受け入れ研究員の活用
昭和 55 年 4 月：「部外研究員受け入れ既定（建設省訓 3 号)」の制定により、昭和 55 年より 60 年の間に、延 60 名の部外研究者を受け入れ、先進技術の普及、国内の研究協力に役立つ。

（5）建研の筑波関連経費などの推移（表２－６）

５５年には、広く公共団体や民間機関からの部外研究員の受け入れの関する規定が」設けられ、多数の部外研究員を受け入れ、先進技術の普及と研究協力に役立てる道が拓かれた。

年表２－６　建研の筑波関連経費などの推移[*22]

（1）つくば関連経費の推移

昭和５３年度：１５億円、　昭和５４年度：１３億円

以後、各年：６～８億円

（2）機器等の整備の推移

５０年度：	０．７億円	５６年度：	１．７億円
５１年度：	１２．２億円	５７年度：	１．７億円
５２年度：	４．６億円	５８年度：	１．２億円
５３年度：	１３．５億円	５９年度：	１．８億円
５４年度：	４．９億円	６０年度：	１．７億円
５５年度：	４．２億円		

（3）所外予算の推移

1）建設技術研究開発経費

５２～５５年度：　約2億円

５６年度：　約3億円

５７年度：　約2億円

2）科学技術振興調整費

５６～６０年度の間、約4～6千万円

注）特別研究費（５２年度～５５年度限り）

５４、５５年度：５～７千万円

（4）受託研究・受託試験

1）受託研究：　５５年度～６０年度：　各年7～11件　約1000万円

移転前後あまり変わらず

2）受託試験：　５４年度～６０年度：　各年20～25件　(防火関係が主)

移転後やや増加（５２～５３年度は14～18 件）

３．追補：旧・建築研究所の移転跡地計画

建設省は、昭和 53、54 年の両年度にわたり、「筑波研究学園都市移転跡地利用による都市整備計画調査」を行った。高山英華（東大名誉教授）を会長とし、6 名の学識者および建設省、国土庁、東京都、新宿区、埼玉県、川口市および日本住宅公団などの関係諸機関からの 19 名の委員により、移転跡地の事例として　新宿地区（建設省建築研究所の跡地 2.1 ha および教育大学光学研究所跡地２ha ）、川口地区（公害資源研究所跡地 4.3ha ）を取り上げ、両地区の周辺部を含み、各 190 ~200 ヘクタールの範囲を対象として行われた。

この調査委員会では、53 年度には、跡地利用原則の検討、現況分析および計画課題の検討と、現況調査を行い、これらを通じて整備基本方針の検討を行った。さらに 54 年度には、、基本構想の代替案の検討を通じて基本計画を策定し、さらに事業手法および事業主体の検討を経て整備計画が策定された。

筆者は移転機関の関係者として、この委員会に参画する機会を得た。以下に、これらの調査結果のうち、新宿地区について調査の要点を整理しよう。

（１）「筑波研究学園都市移転跡地利用による都市整備計画調査」の概要[＊２３]

１）建研跡地の広域的位置づけ（図３－１）

建研跡地は、戸山ヶ原と呼ばれる一帯に位置しており、北は妙正寺川に、西は神田川に囲まれ、新宿の位置する淀橋台から一段低く張り出した豊島台上にある。この地域は明治以降、戦後まで陸軍用地として使われていた。戦後、空地となったこの地域は、復興の過程で、学校、公園、公共住宅、各種公的機関、民間施設および住宅地として分割利用され、現在見るような姿となった。

戸山ヶ原の一帯は、戦後の復興計画においては、市街地外部を囲む「緑地」として位置づけられていたが、その後、縮小の一途をたどり、現在戸山公園として都市計画決定されている地区が残ることとなった。

しかしながら当地域内にある、教育、研究機関および住宅団地などの大規模施設は高度の不燃化がなされていること、また公的な住宅や公共施設として安定した利用がなされていること等によって、周囲とは際立った特徴をもっている。

従って、ここにある跡地の利用を考えるに際しては、こうした戸山ヶ原地域一帯としての特徴を生かし、これらを強化していく方向で検討する必要がある。

２）跡地関連地域の基本的整備課題

以上のような広域的な位置づけに従い、跡地関連地区の基本的な整備課題は以下の様に

整理される。

① 土地利用計画からの課題

東京区部の中心市街地における安定した住宅地としての整備

② 都市防災面からの課題

戸山ヶ原地域の全域を大避難地として、情報、救急救護、応急対応を含めた広域避難拠点とすべく、全域の不燃化と、被災時の応急対応、普及活動の機能を強化する、

③ 交通計画面からの課題

この地域の骨格的な補助幹線道路（７２号および７５号）および小滝橋通りから跡地周辺へのサービス道路の整備

３）建研跡地と周辺地区の整備計画の要請 （図３－２）

上記の基本的整備課題を踏まえて、建研跡地を含むその周辺地区整備計画に要請される事項は以下の如くである。

① 公園、緑地系スペースの系統的な整備

② 住宅地整備の要請と課題

建研跡地北側に隣接する木造密集住宅地区は、戸山ヶ原広域避難拠点内における災害危険度の高い地区であり、不燃化を図ることが要請されている。しかし、この密集地区は建蔽率８０％を超える住宅が多く、そのままの不燃化建て替えは困難であり、南側にある公務員住宅を含めて、跡地を再開発の種地として活用して不燃建て替えを行い、良好な住環境を形成させることが必要である。

４）整備基本計画（図３－３）

以上の如き当地区への整備要請に対応して、地区内の道路整備および住宅整備の基本計画は下記の如くに策定された。

① 道路：

補助幹線道路（７２号（西）、７４号（北）山手線（東）および区画街路（南）に囲まれた地区を、計画地区として設定し、計画道路を配置する。

②. 住宅：

ⅰ）再開発事業用地

北側の密集住宅地の再開発事業用地は、住民、官営機関等の調整を待ち、適正なオープンスペースの確保、動線の確保、良好な住環境の形成を目的として、適正な事業手法を考える。

ⅱ）公務員住宅用地

避難拠点の一部として不燃化を図り、連続的なオープンスペースの確保を行う様、配置計画を考え、建研跡地の一部も利用する。

iii）病院

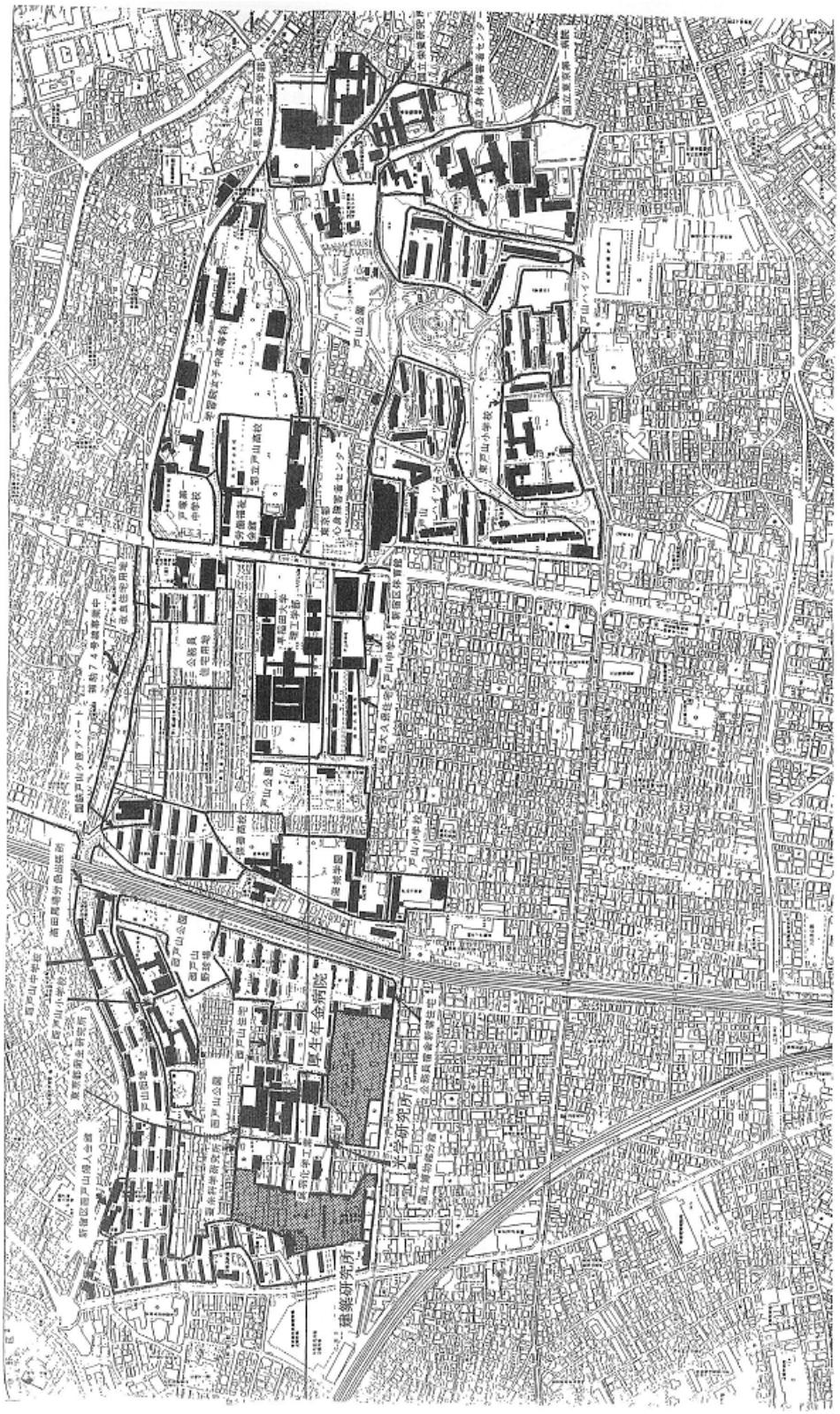
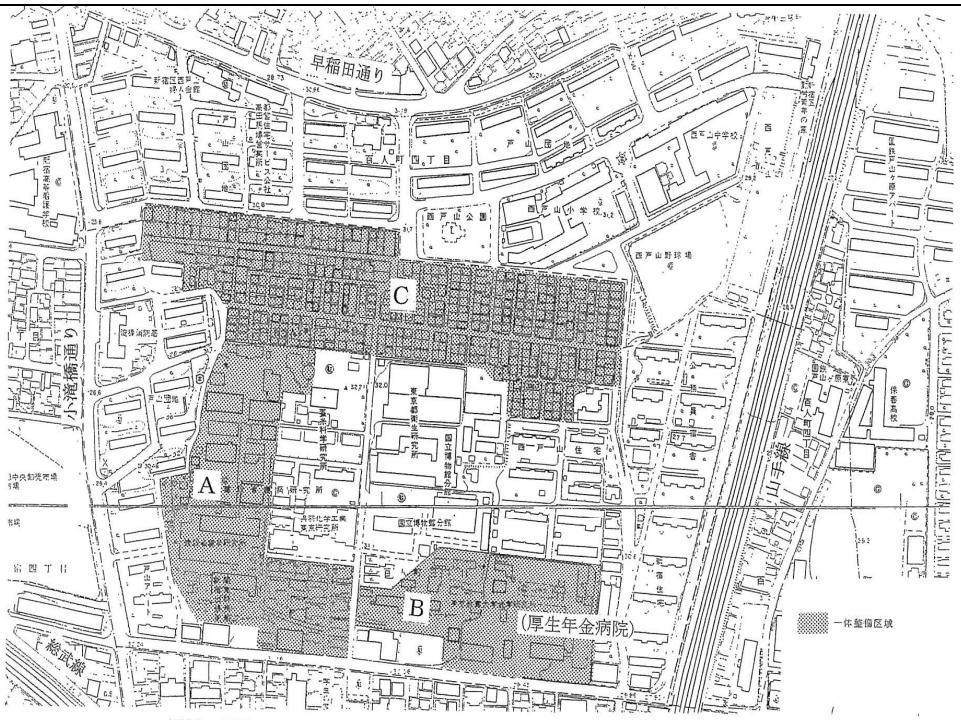


図３－１　戸山ケ原地域現況図（昭和 53 年）[*23]



A：建築研究所跡地　B：教育大光学研究所跡地　C：木造住宅密集地区

図３－２　新宿百人町地区整備区域図[*23]

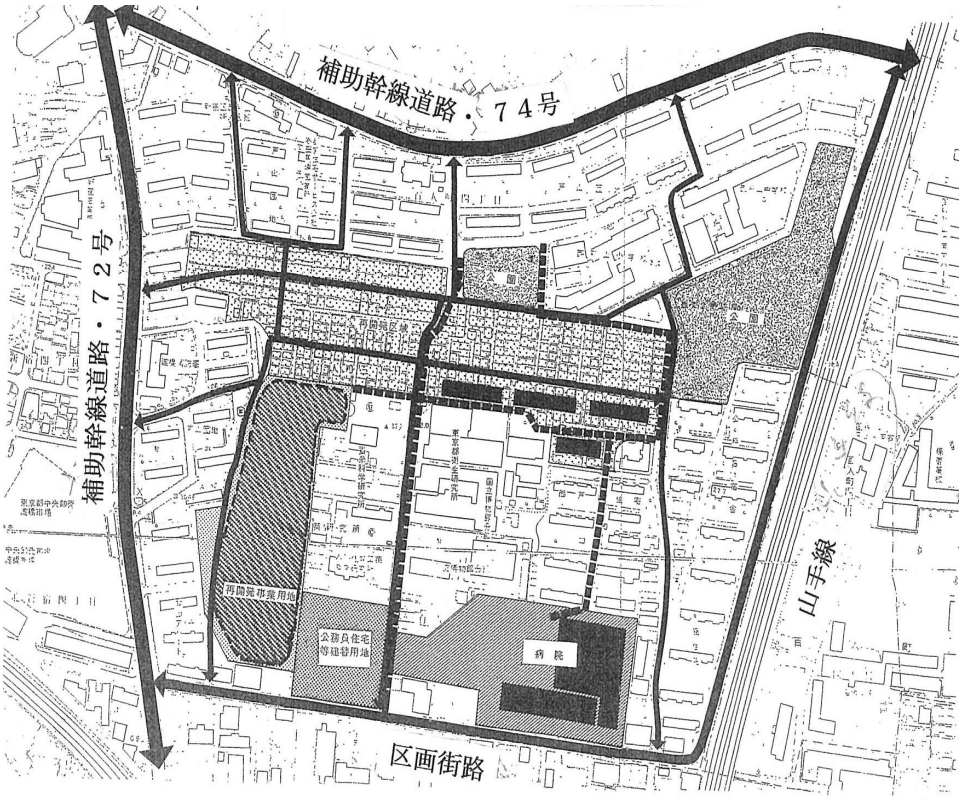


図３－３　新宿百人町地区整備基本計画[*23]

出来るだけ高層集約化を図り、可能な限り有効なオープンスペースを確保する。また周囲の日影の影響、オープンスペースの連続的な形成の観点から、南東側に高層化を図ることが望ましい。

以上の調査結果を踏まえ、新宿区及び大蔵省などにより、密集市街地の住民の意向を踏まえながら、最終的な跡地利用の計画を策定し、再開発・整備事業方法が検討され、これらが実施された結果、別図に示す如く、旧・建築研究所の跡地の中心部分は、公園として整備され、北側の部分は、近傍の木造市街地の区画道路の拡幅整備と移転代替住宅建設などの防災化に使われ、西側及び南側の部分は、公務員住宅、公共住宅等の不燃住宅用地として用いられ、全般的に、都市防災不燃化事業に活用された。

建研の跡地は、前述の様な手順を踏んで、慎重にその利用が検討され、当初の首都の既成市街地の再開発整備に役立てられたが、この結果は当初の移転の大義名分に照らして、意義のあるものとなり、旧・研究機関等の職員の納得の行くものであったと言えよう。

別図　新宿百人町建研跡地の利用[*24]

写真：
新宿百人町建研跡地の現況
百人町ふれあい公園から南の都営住宅等の高層住宅群を見る
(2014.2.26 小林撮影)

出 典 一 覧

＊1：「建築研究所２０年のあゆみ」ｐ－３より　昭和４１年１１月　建築研究所刊
＊2：同　上
＊3：「建築研究所３０年のあゆみ」ｐ－４３より　昭和５１年１１月　同　上
＊4：「建築研究所２０年のあゆみ」ｐ－４より　昭和４１年１１月　同　上
＊5：同　上　ｐ－５より　〃　〃
＊6：「建築研究所３０年のあゆみ」　口絵より　昭和５１年１１月　〃
＊7：同　上　ｐ－１３９より
＊8：都市住宅に関する国立研究所の設置について　昭和４６年　６月　同　上
＊9：創立４０周年記念「建築研究所この１０年のあゆみ」ｐ－１１９～１２０
昭和６１年１０月　同　上
＊10：同　上　ｐ－１２２より　同　上　＊11：
＊11：
＊12：「建築研究所３０年のあゆみ」ｐ－１４１より　昭和５１年１１月　建築研究所刊
＊13：同　上　ｐ－１４２より　同　上
＊14：同　上　ｐ－１３９より　同　上
＊15：創立４０周年記念「建築研究所この１０年のあゆみ」ｐ－２８
昭和６１年１０月　同　上
＊16：同　上　ｐ－１２５より　同　上
＊17：上記の建築研究所２０年、３０年、４０年のあゆみなどより　建築研究所刊
＊18：同　上　同　上
＊19：同　上　同　上
＊20：創立４０周年記念「建築研究所この１０年のあゆみ」ｐ－３０
昭和６１年１０月　同　上
＊21：上記の建築研究所２０年、３０年、４０年のあゆみなどより　同　上
＊22：同　上　同　上
＊23：筑波研究学園都市移転跡地有効利用による都市整備計画調査報告書
昭和５５年３月　建 設 省

なお、この報告書作成に当たり、当時の刊行地図等が利用されたと推測できるが出典は不明である。

＊24：住宅地図・東京都新宿区　平成８年　ゼンリン

なお、掲載が発行元より不許可となったため、原著に掲載されていたことのみ指摘し、代替として現地の道路案内地図の写真を本稿には掲載した。

－－－－－－－－－－－－－－－－－－－－－－－－

著者略歴　　棚橋一郎　工博　（社）建築研究振興協会技術顧問
昭和３７年　早稲田大学大学院博士課程修了
昭和３９年　建設省建築研究所　第１研究部都市施設研究室研究員
昭和５０年　企画室長
昭和５２年　第６(都市計画)研究部長
昭和６２年　「日本・ペルー地震防災センター」主席顧問
平成５年より　早稲田大学・福井大学客員教授
ペルー国立工科大学　名誉教授、名誉博士
（社）日本都市計画学会　名誉会員

４－７．アナログ資料の扱い

（1）ステレオ計測写真

1993 年 11 月に、旧建設省建築研究所が北海道南西沖地震による津波の被害を受けた奥尻島の状況を記録するために、ステレオ計測写真を撮影した。使用した機材は、PENTAX 社製計測カメラ PAMS645 と、ステレオ雲台および三脚である(写真 4-7-1)。この装置は 2019 年現在まで国総研の重要備品として継承されている。

写真 4-7-1 ステレオ雲台と計測カメラ

当時は、デジタルカメラはまだ普及しておらず、ロールフィルムを用いた銀写真が一般的であった。一方、計測写真には、フィルム面が平面であり変形しない、より古い形式である乾板も用いられていた。使用した計測カメラにおいては、ブローニーフィルムを使用し、一コマ６０×４５ｍｍのサイズに撮影し、撮影時点でのフィルムの平面性を確保する

ために、フィルムの背面に多数の孔をあけた平板を設け、フィルムをポンプで吸引する機構を有している。但し、ブローニーフィルムは裏紙と共に巻き取られているため、事実上はフィルム面自体ではなく裏紙を介して間接的に平面性を確保しているに過ぎない。

ステレオ雲台は、三脚の上に設置し、左右二カ所にカメラを固定する雲台を持つ。現場での撮影に際しては、１台のステレオ雲台に、１台の計測カメラを左右に付け替えながら撮影を行い、１地点につき８方位をパノラマで撮影した。よって１地点あたり合計１６コマを撮影した。

計測カメラとステレオ雲台は、2000 年以降は使用されていないが、2016 年時点で国総研の備品として保存されている。

PENTAX 社はその後 RICOH 社と合併し、引き続き写真機材を製造しているが、計測カメラは光学センサを用いたデジタルカメラとなり、フィルムの平面性の問題は解消している。

（２）ステレオ計測写真のデジタル化

一般の写真資料（印画紙およびネガフィルム）は、通常のスキャナにおいてデジタル画像ファイルに変換することができる。本研究においては、2011 年当時の市販の民生用スキャナの内で 9600dpi（35mm の長さでは 13,440 ドット）、と最も解像度が高い、キャノン社製の CanoScan9000F を使用して、1990 年代の古写真のネガフィルムを画像ファイルに変換した。

写真 4-7-2 ステレオ計測写真ネガフィルムの保存状態

撮像面（ガラス）は A4 サイズの紙資料がスキャンできる構造であり、ここにフィルム・

ホルダを装着して、３５ｍｍフィルム、ブローニーフィルム、およびマウントされたスライドをスキャンする。紙資料等を扱う場合には、撮像面の下のセンサと同じ側の光源を使用し反射光をスキャンするが、フィルムを扱う場合には、装置の蓋の内部の光源を使用して、透過光でスキャンする。フィルム・ホルダは撮像面に固定することができ、フィルムを撮像面に平行に固定すると共に、マーカーがあり、附属のドライバソフトがスキャン後の画像を認識して、フィルムの撮像面だけを抽出することができる。これにより、数コマから成るネガフィルムをスキャンした結果は、撮影時の駒数に分解して、撮影単位毎の画像ファイルを生成する。また、ネガの画像の色彩を反転して、焼き付けた場合と同等のカラー画像としている。

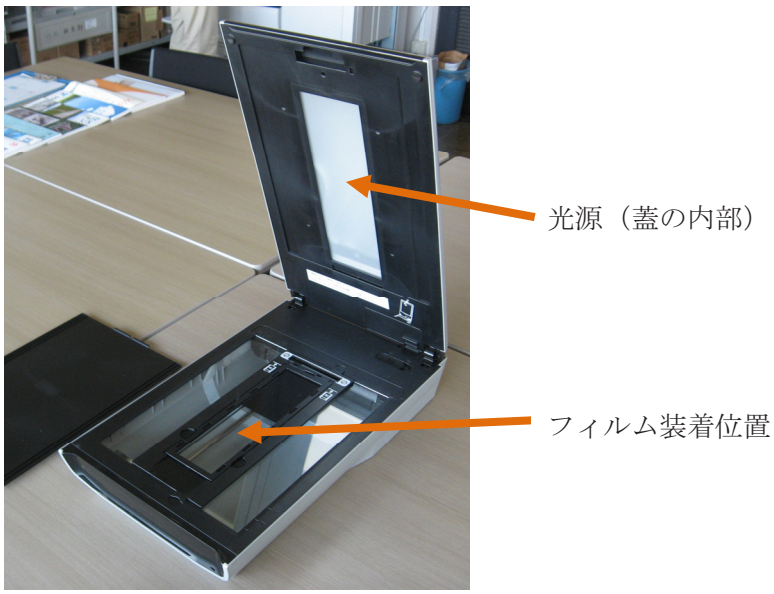


写真 4-7-3 本研究に使用したフィルム・スキャナ

但し、フィルム・ホルダを使用する方法では、スキャン時のフィルムの曲がりを防ぐことが難しい。とりわけ。ブローニーフィルムを使用したステレオ計測写真の場合には、撮影段階で吸引ポンプを用いて、フィルムを裏板に吸着する方法により、ネガフィルムの平面性を確保した撮影が行われている。よって、スキャン段階においても平面性を確保した処理が必要となる。このためには、無反射ガラスを使用して、スキャナ上部のガラス面にフィルムを押し付けて平面にする方法を用いることができる。

無反射ガラスは、ガラスの表面にコーティングを行うことにより、反射を低く抑えた素材であり、写真専門店等において販売されている。フィルム・ホルダを使用しない場合には、画像のトリミングを手作業で行う必要がある。

（３）古図面、古地図のデジタル化

古い資料では、紙質が劣化している場合があり、通常のフラットベッド型のスキャナを使用すると、資料が痛む場合がある。更に、古地図や大判の地図帳のようなサイズの大きい資料の場合には、スクロールしたり頁をめくったりしながら裏返してスキャナの上に載

せてスキャンする操作を繰り返すと、資料の劣化が著しい。

このような脆弱な資料をスキャンする場合に、以前であれば、三脚を用いて資料の上部から撮影を行った。近年では、オーバーヘッド型のスキャナが普及しつつある。ごく簡単な装置から、図書館等が使用する本格的な機種まで幅広い選択肢がある。本研究においては、2014 年度に富士通の ScanSnap(SV-600)という機種を備品として調達し、古い図面や委員会資料等のスキャンに使用した。

写真 4-7-4 オーバーヘッド型スキャナによる古い地図のスキャン

一般にこの種のスキャナにはハンドラ・ソフトが DVD-ROM の形で附属しており、系統的に画像ファイルを整理する機能などができる。使用したシステムの場合には、画像処理ソフトウェアにより、冊子型の資料の上下の縁の曲線を自動認識し、これに基づいてページの中央部の曲がりを画像処理により自動的に修正するような操作も行うことができる。但し、実際に使用した経験によると、こうした機能はまだ発展途上にあり、大量の資料を処理する実務に使用するためにはまだ改良の可能性があるように思われた。

PC 側にセットアップするために、以下のソフトウェアが付属している。

①マネージャ

個別の資料のスキャン作業のための GUI を提供する。PC の起動に際して自動的に起動するサービス(PfuSsMon.exe,3.19MB)として動作しており、スキャナを USB 接続し電源を投入した段階で、操作画面を表示する。

資料を適切な位置にセットした上で、画面上のスキャンボタンまたはスキャナの開始ボタンを操作することによりスキャン動作が開始する。スキャン作業自体は、冊子型の資料の場合反転作業なしに、紙の頁をめくるだけでよいため、１分に２コマ以上の速度で進めることができる。一連のスキャン作業が終わった段階で終了すると、画像ファイルを登録するデータベースが作成される。このデータベースは、オーガナイザによって使用される。スキャナには、黒色の下敷きマットが付属しており、これとの明らかな彩度の違いから、画像の外周を認識し、余白をトリミングする画像処理は、この段階で行われる。

設定画面は、以下のタブにより構成されている。

(1) アプリ選択：
　画面上部の「クイックメニューを使用する」にチェックが入っている場合、クイックメニューだけが選択可能である。
　チェックが入っていない場合には、以下の選択肢が提供されている。
・オーガナイザ（本製品附属の画像ファイル処理ソフトウェアである）
・起動しない（ファイル保存のみ）
・Adobe リーダ
・カードマインダ
・指定したフォルダに保存
・メールで送信
・プリンタで印刷
・モバイルに保存
・Google ドキュメント（TM）に保存
・Salesforce Chatter に投稿
・WORD 文書に変換
・EXCEL 文書に変換
・PowerPoint 文書に変換
・ABBYY スキャンによる検索可能な PDF
・ピクチャ・フォルダに保存
・追加と削除（アプリケーションを１０まで登録できる）
（これらは、2015 年時点において広く使われていた画像処理方法を示している）
(2)保存先：画像を保存するディレクトリを指定する。
(3)読み取りモード
　画質（解像度）、カラー自動判別、読み取り面（片面／両面）、画像の向きを自動的に回転する（チェック）、継続読み取りを有効にする（チェック）
(4)ファイル形式（JPEG／PDF）
　マーカー部分の文字列を PDF のキーワードにする（チェック）
　検索可能な PDF にする（チェック）
　テキスト認識オプション（日本語）、先頭ページのみ／全ページ（ラジオボタン）
(5)原稿
　後から選択／平らな原稿／見開き原稿
　原稿サイズの選択（サイズ自動検出、最大エリア）
　ファイルサイズ（圧縮率）

②オーガナイザ

　マネージャによる一連のスキャン操作が終了した時点で自動的に起動し操作画面が表示される。また、単独で起動することもできる。

　画像の歪みを補正する機能を有する。但し、まだ完成度は高くない。メーカー側もそのことを認識しているためか、歪み補正を修正する機能だけを有するアプリケーションを単独で有償発売する前の試験的な無償の機能として、ハンドラ・ソフトの機能に加えている。

　歪み補正は、対話型で確認しながら進めるものであるため、スキャン操作よりも時間を要する。従って、遠隔地等で失敗が許されない場合を除き、持ち帰ってから歪み補正の作業を行うような段取りが望ましい。

③カードマインダ―

　名刺を画像入力し OCR で認識し、データベースに登録する。本研究においては使用していない。

④カードマインダ―ビューワ

　文字列で、名刺データを検索する。本研究においては使用していない。

⑤無線設定ツール

　無線 LAN(Wi-Fi)を介して、PC をネットワークに接続し、外部からのアクセスを可能にするための機能である。本研究においては使用していない。

(1) 検索可能にするファイルの一覧

(2) パスワード操作
　WEB サーバー上に構築する画像データベースにアクセスするためのパスワードを設定する（パスワード設定に使用したセットアップを行った PC が故障した場合に、パスワードが再設定できなくなり、サーバー上のデータにシステム管理者以外誰もアクセスできなくなるようなリスクに関する解説はない）。
(3) オンライン・アップデートの設定
　上記のソフトウェアを最新の状態に更新するために、サーバーに確認のためにアクセスする時刻や頻度を設定する。サービスとして動作しており、システムの起動と同時に自動的い開始する（SsUWatcher.exe, 52.0KB）。
(4) サポートツール
　無線接続ができない場合に、修復を行うツール
　アクセス権チェック（データ保存先の書き込み権限）

これらを全て活用すると、スキャナを接続した PC がインターネットに常時接続した環境において、メーカーが提供するサーバー上にスキャン画像を蓄積した上で、画像処理や文字に関する OCR 処理を行い、検索可能な文書に変化することが可能である。

但し、必ずしも必要としない機能を使用しないような設定を注意深く行わないと、常時接続できない環境における動作障害や、重要な情報のネット上への流出の危険性を抱えることになる。

三次元アーカイブス作成において、中間結果である貴重な資料のスキャン画像等をインターネットで直接公開する必要はないため、これらをローカルに保存するような設定を行ったが、開発者が期待している設定ではないためか、フリーズする現象が生じた。更に、この障害について報告のために WEB アクセスしようとするために、常時接続していない環境においては、マシンを再起動しなければ作業続行できないような状況も生じた。

このため、大量の画像を連続してスキャンする作業方法を改めて、上記のようなトラブルの発生による手戻りを小さくするために、小割にして少しずつスキャン作業を区切るような作業手順に改めた。

本研究においては、資料を借り出して研究室内でスキャン作業するのではなく、ノート型 PC とスキャナを、資料保管庫に近い会議机に設置して作業する方法としたため、ネットワーク接続はなく、また PC 内部のディスク容量も小さいため、大量の画像をスキャンするような場合には、USB 接続の外付け HDD 等に格納するのが便利である。

画像ファイルには、タイムスタンプから自動生成したファイル名称が付されている。更に、歪み補正を行う、という選択を行った場合には、パラメータを格納したファイルが附属する。

（４）建築図面

建築図面が、文書ファイルの中に折りたたまれて保存されているような場合には、折り目がスキャン作業の大きな障害になる。このような場合には、通常のフラットベッド型のスキャナ（A3 サイズ）で分割スキャンを行い、画像処理で再合成するような方法も試みた。但し、向きを微修正（例えば角度で１° 回転）する画像処理は解像度を下げるため、図面中の意味あるまとまりは一つのファイルに収まることが望ましい。価値の高い画像で、外部の業者等に持ち出して大判のスキャナにかけることが難しいような場合には、一度オプティカルに平坦な分割ハードコピーを作成した上で、裏面に向き合わせのための補助線を入れ、角度を合わせたスキャンを行う必要がある。

本研究の中で使用した、旧建設省建築研究所の除却前実測図に関しては、オリジナルのトレーシングペーパーに鉛筆手描きの図面の損傷を避けるために、当時複製のために作成され、共に保管されていた第二原図を、分割スキャンする方法を採った。CAD を用いた手入力を行うための作業用としては十分である（図 4-7-1）。オリジナルの手描き図面は、損傷や紛失がなければ、少なくとも今後数十年の保存には耐えると判断されたため、なるべく触れないように配慮した。

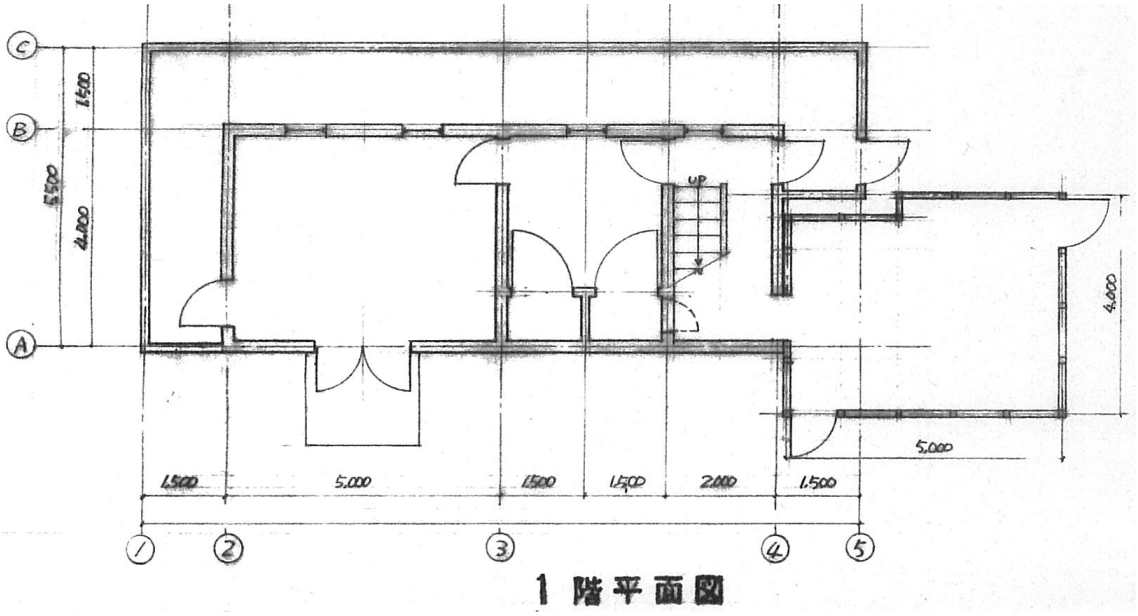


図 4-7-1　建設省建築研究所　実大排煙実験室の 1 階平面図

４－８ 参考文献一覧

（1）バックグラウンドとなる研究

1.小林英之「住環境形成シミュレーション」日本建築学会・第 16 回情報・システム・利用・技術シンポジウム論文集(1993.12)

2.小林英之・狩野勝重「依怙都市・二本松」調査報告

(1)　序論 日本建築学会大会梗概 1992.8

(2)　字別に見たフローとストック 1993.9

3.小林英之「歴史的観点からみた住宅のライフサイクル」

あらか 12、建設省建築研究所 1994.10

4.小林英之・丹羽薫「景観シミュレータ・景観データベースの研究開発について」(JACIC 情報 No.34,日本建設情報総合センター1994.4)

5.Hideyuki Kobayashi ： GIS and Landscape Simulation (One-day workshop on historical heritage, ministry of Public Works, Indonesia, 1996.5 Yogyakarta, Indonesia

6.小林英之「景観シミュレータ」(公共建築 36/3 No.141;公共建築協会 1994.7)

7.小林英之「景観シミュレータの研究開発」(測量 Vol.45,No.5, 日本測量協会 1995.5)

8.小林英之「３次元 CG による土木建築施設のための景観検討システムープロトタイプ版(Ver.1.0)－」(建設省建築研究所・建築研究資料 No.85,1995.9)

9.小林英之ほか「景観デザインにおけるシミュレーション・評価・プレゼンテーションの活用とその実際」(工業技術会,1996.5)

10.小林英之「景観シミュレータができるまで」(ランドスケープ・デザイン,マルモ出版 1997.6)

11.小林英之「建設省版景観シミュレータ・操作自習の手引き(Ver.2.03)」(建設省建築研究所・建築研究資料 No.92,1997.11 絶版、但しその内容の殆どは文献 16, 17 に含まれている)

12.小林英之「景観シミュレータで見る地域の将来像」あらか１５、建設省建築研究所 1997.11

13.小林英之「建築與都市發展電腦視覺模擬(Computer Graphic Simulation for Building and Urban Development)」中日工程技術検討會 建築研究組 論文集(1997.11.4 繁字体中文)

14.일본건설성 건축연구소 제 6 연구부 도시개발엔구실
고바야시 히데유끼 "경관시뮬레이터기술의 지역개발에
의 응용" (농어촌진흥공사 농어촌연구원 1998.8)

(日本建設省　建築研究所　第６研究部　都市開発研究室
こばやし　ひでゆき　"景観シミュレータ技術の　地域開発へ
の応用"(農漁村基盤公社　農漁村研究院 1998.8)

15. Hideyuki KOBAYASHI, "Urban Simulation Technologies for Planning – from

synchronic to diachronic-“, Proceedings of International Symposium on City Planning 1999.9, Tainan, Taiwan.
16.小林英之「成熟都市シミュレータ Ver.1.0＋景観シミュレータ Ver.2.05 実務マニュアル」建設省建築研究所・建築研究資料 No.96,2000.7
17.小林英之「まちづくりのためのコミュニケーションシステムの開発」国総研アニュアルレポート No.1,2002.3
18.Hideyuki KOBAYASHI “Development of Communication System for Town Planning”, Annual Report of NILIM 2002
19.小林英之「まちづくりのためのコミュニケーションシステムの開発」平成１４年度国土交通省国土技術研究会　自由課題（論文集,No.47, pp.185-188）
20.小林英之「まちづくりのためのコミュニケーション・システムの開発」土木研究センター、土木技術資料 Vol.45 No.3 2003.3 表紙及びグラビア
21.小林英之「まちづくり・コミュニケーション・システム　操作・運用マニュアル」国土技術政策総合研究所資料 No.134, 2003.9 (290p)
22.国土技術政策総合研究所高度情報化研究センター・環境研究部緑化生態研究室「国土交通省版・景観シミュレータの活用実績と、今後の応用」平成 15 年度国土技術研究会ポスター 2003.11
23. Hideyuki KOBAYASHI,”Development of Communication System for Town Planning”, International Conference on Construction Information Technology(Incite 2004):World IT for Design & Construction, Langkawi, Malaysia:18-21 February 2004(8p)
24. Hideyuki Kobayashi “Configuration Process of Landscape in Japanese Settlements –Regional Context, Historical Background and Future Scope- Proceedings of International Symposium on City Planning 2004, City Planning Institute of Japan, 2004.9
25. 小林英之「電子納品データ（SXF）と、５ｍメッシュ数値地図を用いた景観検討用データの生成」情報・システム・利用・技術シンポジウム論文集 No.27 2004.12, pp.245-249
26. 小林英之・小栗ひとみ「まちづくりのための景観シミュレーションの活用」日本造園学会造園技術報告集 2005, No.3 pp.138-141
27. 小林英之「まちづくり・コミュニケーション・システムーネットワークを用いた景観シミュレーションー」測量 Vol.55 No.5 pp.37-40 2005.5
28. Hideyuki Kobayashi “Analysis of Satellite Images for Measuring Urban Green Coverage Ratio in Bandung and Cirebon – As Basis for Planning Future Urban Form regarding Climate Change” Second International Symposium on Sustainable Humanosphere 2007.7, Bandung, Indonesia
29. Hideyuki Kobayashi “Monitoring CO2 Emission in Indonesian Planned Housing Complexes and Designing Alternative Future Images”, Technical Note of Nilim No.440,

2008.3 (56p)
30 小林英之「データ活用による景観シミュレーションの試み」平成 20 年度国土技術研究会ポスター 2008.10
31. 小林英之「データ活用による景観シミュレーションの試み」土木研究センター、土木技術資料 2009 No.2 pp.22-27, 2009.2
32. 小林英之「情報が生産方式を変える　地域づくりに ICT 活用－景観シミュレーションの展開－」日刊建設工業新聞 2009.3.27
33. 小林英之「国土交通省版・景観シミュレータのヒストリー(1993-2009)－ソフトウェアの完成を目指して」情報・システム・利用・技術シンポジウム論文集 、日本建築学会 2009.12.3, pp.111-114
34. 小林英之「国土交通省版・景観シミュレーション・システム Ver.2.09 のアーキテクチャ」国土技術政策総合研究所報告 No.42, 2011.3 (603p)

（２）本研究とその成果の発表

35. 小林英之、稲垣森太「奥尻島集落における古写真の位置比定と編年について－1993 年以前の青苗集落を中心として」日本建築学会北海道支部研究報告集(pp. 405–412)、2013. 6
36. Hideyuki KOBAYASHI: ”3D Archiving Houses and Settlements -Alternative Technologies to support Lasting Diachronic Memory-”, International Symposium on City Planning, 2013.8, Sendai
37. 小林英之「奥尻島における古写真を用いた 1993 年被災集落の立体的復原について」日本建築学会大会学術講演梗概集(pp.457-8)2013.8
38. 小林英之「高台整地の景観シミュレーション－プラグインの開発－」土木学会全国大会Ⅳ-048(2013.9)
39. 岡田成幸、中島唯貴、大柳佳紀、小林英之ほか「奥尻島災害復興過程にける生活環境の変容に関する研究」北海道地域自然災害資料センター紀要 2014. 3
40. 南慎一、小林英之、稲垣森太、大柳佳紀「奥尻島の記憶の町並再生プロジェクト報告」北海道地域自然災害資料センター紀要(2015.3)
41. 小林英之「新宿百人町の除却された建物群のデータ復原－三次元アーカイブスの永久保存に向けて－」日本建築学会大会学術講演梗概集(2015.9)
42. 小林英之「建築物などを記録し長期保存されたレガシーデータの利活用方法－メタファイル・コンパイラと利活用ライブラリによる処理系－」日本建築学会大会学術講演梗概集(情報システム技術 No.11032 ,pp.71-72, 2016. 8)
43 Hideyuki KOBAYASHI: “Use of 3D Archives of Houses and Settlements for Permanent Memory -2 cases of 3D data attached with metafile for preservation of records with 4 trial cases of application toward future usage- “, 11th International Symposium on Architectural Interchange in Asia (ISAIA, September 2016, Sendai, C-5-4, 5p)

（３）本研究において参照した技術資料等

①OpenGL について

44. OpenGL Architecture Review Board“OpenGL Programming Guide(日本語版)”
アジソン・ウェスレイ(1993.12)

45. アフタブ・ムンシ、ダン・ギンズバーグ 、デーブ・シュライナー 著、松田 晃一 訳、「Open GL ES 2.0 プログラミングガイド」ピアソン桐原 、2009.11

②写真の解析（立体の復原）

46. 高木幹雄・下田陽久監修「画像解析ハンドブック」
東京大学出版会(1991.1)

③計算アルゴリズム

47. P.チャドウィック著、後藤学訳：「連続体力学－簡明な理論と例題－」ブレイン図書出版株式会社、1979.2
(原著：P.Chadwick:”Continuum Mechaniocs –Concise Theory and Problems-“ London George Allen & Unwin Ltd. 1976)

48. 奥村晴彦「アルゴリズム事典」技術評論社(1991.2)

49. 金谷一朗著「３D-CG プログラマーのためのクォータニオン入門」工学社、2004.1

50. J.H.コンウェイ／D.A.スミス著、山田修司訳「四元数と八元数」培風館、2006.11

51. 林晴比古「明快入門　コンパイラ・インタプリタ開発」ソフトバンク・クリエイティブ株式会社、2010.1

④三次元データ形式

52. James D Murray et-al. “Encyclopedia of Graphics File Formats for PC, Macintosh, and Unix Platforms”, O’Reilly & Associates, Inc.1994

53. 金澤文彦ほか「道路中心線形データ交換標準（案）基本道路中心線形編 Ver.2.0」
国土技術政策総合研究所資料 371, 2007.1（**電子納品データ三次元形式**）

54. Thomas Liebich “IFC 2x Edition 3：Model Implementation Guide, Version 2.0” 2009.5

55. ３次元設計データ交換標準（案）に準じた LandXML1.2 拡張（案）平成 25 年 3 月　国土技術政策総合研究所（Web 公開）

⑤開発環境

56. 杉松秀利「ＳＱＬリファレンス・ブック」ナツメ社,2000.12

57. Ken Arnold, James Gosling, David Holmes 著、柴田芳樹訳「プログラミング言語 Java 第 4 版」ピアソン・エデュケーション,2007.4

58. フランク・アブルソン、チャーリー・コリンズ、ロビ・セン著、土肥拓生、谷沢智史訳「コードからわかる Android プログラミングの仕組み」日経 BP マーケティング,2010.1

59. 出村成和「Android NDK ネイティブプログラミング」秀和システム,2011.8

（４）本研究において参照した機関史、地方史等

①過去の機関の紀要等

61 内務省土木試験所「試験調査事項　年報」

昭和 14～17 年度の年報が、国土技術政策総合研究所の図書館に保管されている（旭庁舎）

昭和 17 年度の年報の奥付は、

東京市本郷区駒込上富士前町二十六番値　電話大塚(86)自 3101 番 至 3103 番

62 内務省防空研究所「内務省防空研究所彙報　第一号」昭和 18(1943)年 7 月 15 日」

東京都世田谷区玉川野毛町一〇〇三番地 電話田園調布 4001,4002,4003 番、玉川 370 番

（国立国会図書館蔵） http://dl.ndl.go.jp/info:ndljp/pid/1062956

63 総理府戦災復興院技術研究所「技術研究所報告」第 1 号［非売品］

昭和二十二年七月二十五日印刷、三十日発行

編集者　総理府戦災復興員技術研究所　代表者　菅原　肇

東京都新宿区百人町四丁目三九四番地

（メリーランド大学ゴードン・W・プランゲ・コレクション）

（目次）

（１）板硝子工業と復興建築の見透　　新海悟郎　内山諫

（２）住宅及び店舗の復興に関する実情調査　　碓井憲一　川越邦雄　入澤恒

＊なお、50 年史別冊年表に上記②の、建築研究所の蔵書印のある異本の表紙の写真が掲載されている(p.47)。これは 30 周年記念アルバムの５ページに掲載された写真の転載と思われる（現物未確認）。

64 戦災復興院官房技術研究所要報，建設院第二技術研究所要報，建設省建築研究所要報

ガリ版刷り手書き原稿の研究報告であり、機関名称の変更を越えた通し番号（第 1 号～第 162 号）で整理され、建築研究所図書室に保管されている。30 周年記念アルバムには、「戦災復興院　技術研究所要報 第一号」の表紙写真が掲載されている(p.7)。手書きガリ版刷りの総目録が先頭に付されている。[２０年のあゆみ]の「１０．主要研究発表文献目録」の発表誌が「建研要」となっているものがこれらに収録された報告である。

②調査報告書

65 建設省「筑波研究学園都市移転跡地有効利用による都市整備計画調査報告書」昭和 55(1980)年 3 月

③機関史

旧、建設省建築研究所が 10 年毎に記念出版した所史等として以下の文献がある。

66 建設省建築研究所（菅原肇）「建築研究所 20 年のあゆみ」(1966.11.20)

東京都新宿区百人町 4 丁目 394 番地

第 1 章建築研究所の概要　に成立期の記述がある

参考資料として年表(p.109)

巻末に藤田、竹山、平賀の回顧録(p.189)

67 建設省建築研究所「創立二十五周辺記念出版　主要研究論文集」（1971）

68 建設省建築研究所「国際地震および地震工学研修 10 年のあゆみ」(1972.8.5)

69 建設省建築研究所「建築研究所 30 年のあゆみ」(1976.11)

３０周年記念事業出版部会

〒160 東京都新宿区百人町３丁目２８番８号

筑波移転に向けた準備状況が記載されている

70 建築研究振興協会「建築研究所創立 30 周年記念アルバム」(1976.11.15)

編年体で各建物の写真が掲載されている

71 建設省建築研究所「創立４０周年記念　建築研究所この 10 年のあゆみ」(1986.10)

創立 40 周年記念行事実行委員会出版部会

〒305 茨城県筑波郡大穂町立原１番

72 建設省建築研究所「国際地震工学研修の 30 年」(1992.12.1)

73 建設省建築研究所「建築研究所 50 年」(1996.10)

巻末資料集として、戦前から戦後の形成期に関する資料が再整理されている。

74 建設省建築研究所「BRI 50 YEARS 建築研究所 50 年-プロフィル-」(1996.10)

年表及び小さな写真が掲載されている。和文。

④紹介パンフレットなど

75 建設省建築研究所「建築研究所 1963」(1963.10)

東京都新宿区百人町４丁目 361-4151

当時の配置図が掲載されている。敷地面積 21,018 ㎡(6,396 坪),延床 9,111 ㎡(2,761 坪)

76 建設省建築研究所「建設省建築研究所 1976-77」(1976.7)

東京都新宿区百人町 3-28-8 電話 361-4151

筑波に移転する２年前当時の配置図が掲載されている。

敷地面積　庁舎 20,024 ㎡, 宿舎 1,031 ㎡

建延床面積　　12,297 ㎡,　　927 ㎡

⑤定期刊行物、機関誌

77 建築研究所年報

時期によりスタイルが変化している。

194x-1969 年：施設整備と実験機材整備の詳細な記録がある

1970 年-2000 年：研究成果を中心に報告している

78 建築研究報告

79 建築研究資料

80 建築研究成果選「あらか」

81 エピストラ

82 雑誌「住宅」(1952 年 7 月創刊、日本住宅協会)

第一研究部長新海悟郎氏は雑誌「住宅」の創刊に尽力し、初期のグラビア記事や論文を多数投稿している。急逝した 1962 年には新海の追悼文なども掲載されている。

1952.8「日本の不良住宅」（グラビア）
1952.10「船宿生活」（グラビア）
1953.8「都市住宅は老朽している」（グラビア）
1954.6「投げやりな住宅維持」（論文）
1954.10「住宅の維持費はどの位かかるか」
1954.12「建物老朽化による住居水準の低下」
1956.6「人物寸評：新海悟郎」U.N. ［3］学会賞を受賞した新海の人物を紹介。筆者は西山卯三と思われる。

1961.2「どん底の町・釜ヶ崎」（グラビア）
1961.11「高橋寿男君の霊に」
1962.5 木村巳代治「新海悟郎氏の逝去を悼む」
1962.6 西山卯三「新海悟郎君を憶ふ」、城谷豊「新海さんのこと」

83 建築技術(昭和 25(1950)年 7 月～

建設省建築研究所から月刊で発行された。昭和 35(1960)年第 111 号から同研究所監修となり、建築技術社による発行となった。

84 都市計画

都市計画学会の設立まで、都市計画に関する

85 建築の研究

建築研究所の創立に関係した回想録等が収録されている

⑤営繕事業記録

86 筑波研究学園都市建築の記録

全体の経緯に関する解説がある

研究機関毎の基本的な諸元、設計者、施工者、記録写真がある

87 筑波研究学園都市官庁営繕事業記録(非売品,1981)

営繕の体制全体に関して記録が掲載されている。
研究機関毎に移転計画と施工の記録が掲載されている。
建築研究所の計画に当たっては、久米設計が執筆を担当した。

⑥建設省建築研究所が施設管理、研究資料等として所蔵している内部資料

88 UNESCO レポート

International Institute of Seismology and Earthquake Engineering Tokyo, Japan

Report prepared for the Government of Japan by the United Nations Educational, Scientific and Cultural Organization acting as Executing and Participating Agency for the United Nations Development Programme, Special Fund Component, for the period 1963-1968

国際地震工学部の建物図面、写真が掲載されている。

89 百人町時代の建物の記録図

主に解体除却の資料とするために作成されたとされ、総務部のラインで保管されていたようであるが、調査時点では建築研究所情報技術課の倉庫に保管されていた。発見場所に、３Ｄデータ入力結果を添えて引き続き保管している。

・建物実測図
原図（トレペに手描き）
第二原図
白焼２部
・工作物実測図
原図
第二原図

90 旧企画室資料：レターファイル１７冊

昭和 48 年ころに、筑波移転後の庁舎計画を内部検討した資料であり、途中段階での計画案が図面として残されている。

91 立原庁舎の設計図書

実際に施工された設計図である。営繕建設本部が、谷田部町大角豆に存在していた時期にはここで作成・保管された（1973～）。本部が閉鎖となった後も、しばらく倉庫として図面を保管していた。倉庫を撤去する際に、各研究機関に引き継がれた。建築研究所の場合には、企画調査課の倉庫に搬送された（作業を担当した小林由二氏談）。

⑦住宅地図等

国土技術政策総合研究所住宅都市資料室に、以下の地図等が保管されている

92 航空写真地図帳

昭和 38(1963)年～53(1978)年、この間発行元および地図の名称が変化しているが、同一系列と考えらえる。筑波移転機関の移転前の所在地確認に使用した。

93 住宅地図（ゼンリン）

1973~2002 年頃の歴代地図が首都圏を中心に揃っている。筑波移転機関の跡地利用の確認に使用した。初期の発行所は、日本住宅地図出版株式会社（旧㈱住宅地図出版社）東京都千代田区西神田 3-8-7 となっている。

94 日本建築学会「日本近代建築総覧」1980 年 3 月 30 日、技報堂

⑧関連機関史

95 建設省土木研究所 50 年史編集委員会「土木研究所 50 年史」昭和 47 年 11 月 28 日

東京都文京区本駒込町 2 丁目 28 番地 32 号

96 建設省三十年史編集委員会編集、社団法人建設広報協議会発行

「建設省三十年史」昭和 53 年 7 月 10 日

東京都港区西新橋 3-15-8 西新橋中央ビル 03(432)1428・1429

97 建設省土木研究所「土木研究所 60 年史」昭和 57(1982)年 9 月 16 日

茨城県筑波郡豊里町大字旭 1 番地

98 建設大学校監修、建設大学校三十年史編集委員会編集、
財団法人全国建設研修センター発行「建設大学校三十年史」昭和 62 年 9 月 5 日

東京都小平市喜平町 2-1-2 0423(21)1634

99 建設省土木研究所 70 周年記念事業実行委員会記念誌編集班「土木研究所 70 年史」

平成 4 (1992)年 10 月　　〒305 茨城県つくば市大字旭 1 番地

100 建設大臣官房官庁営繕部監修「霞ケ関 100 年－中央官衙の形成－」平成 7(1995)年 11.20
公共建築協会

101 日本都市計画学会「都市計画学会五十年史」平成 13(2001)年 11 月 13 日

〒102-0082 東京都千代田区一番町 10 一番町ウエストビル 6 階

⑨単行本

102 村松貞次郎「日本建築家山脈」昭和 40 年 10 月 20 日、鹿島出版会

103 田中孝「物語・建設省営繕史の群像<上>」昭和 60(1985)年 7 月　日刊建設通信社

建設省建築研究所開設へ　の節(p.40)に　建築研究室の終戦時の状況が記されている。

104 篠澤清見「創造する人びと－建築研究所誕生から 30 年大久保時代の第四研究部・外史」1999 年 11 月 1 日「創造する人びと」を出版する会（非売品）

千葉県船橋市習志野台 5-28-17 tel:047(465)0712

沼津の技術員養成所に関する詳しい記述がある

105 沢井実「近代日本の研究開発体制」2012 年 11 月 15 日　名古屋大学出版会

⑩本節の時代を扱った最近の論考

106 長谷川直司「大蔵省営繕組織の系譜」公共建築 42-4, 2000.10

107 阿部正隆・西村幸夫・窪田亜矢「戦前における内務省地方計画構想の一終着点－地方計画法案・関東地方計画要綱案の策定過程に着目して－」都市計画論文集 Vol.46, No.3

(2011.10)

108 中島直人「戦後復興期における都市計画研究者の組織化と研究課題の動向－都市計画研究連絡会の活動に着目して－」都市計画論文集 Vol.52, No.3 (2017.10)

⑪シンポジウム記録

109 ［特別寄稿］旧・建設省建築研究所の筑波研究学園都市への移転
－その経緯と移転後の研究の展開－ について

旧建設省建築研究所の筑波移転に関して、本研究の一環として、4回のシンポジウムを開催した。第1回は、2014年3月12日に、まだ第2回は2015年9月3日に、建築研究所展示館を会場として実施した。第3回は、2017年3月、第4回は2017年11月、建築研究所画像情報棟を会場として実施した。第1回は、旧建設省建築研究所の新宿百人町から筑波研究学園都市への移転計画の企画立案に従事された元、建設省建築研究所企画室長の棚橋一郎氏と、北海道立寒地住宅都市研究所の札幌から旭川への移転計画の企画立案に従事された大柳佳紀氏に講演して頂き、討論を行った。第2回は移転当時の担当者の確認と現在の消息、および新発見の記録実測図、計画図等を中心にパネル展示し、当時関係者を招いた討論を行った。第3回は、大型計算機の変遷を整理するとともに、百人町の復元建物をコンテンツとしてタブレットにVC-3Mをセットアップした表示を行った。第4回はタイガー計算機、画像討議室、都市シミュレータ等の変遷について討論した。

第1回のシンポジウムでご講演頂いた棚橋一郎氏には、発表資料を基に原稿をまとめて頂くと共に、講演を記録した映像を作成した。以下、この資料を「特別寄稿」としてこの報告に掲載すると共に、その際に記録した映像をこの報告のDVD-ROM に収録して記録とする。

⑫地方史

110 一宮町役場「一宮町誌」昭和42(1967)年12月10日発行

昭和19～20年に大蔵省大臣官房営繕課建築研究室が疎開していた相興村、相興小学校に関する地元側の記述がある。

p.628 以下に「第三節 太平洋戦争と一宮」には、東京からの疎開があったこと、7月6日の甲府大空襲などについて記されている。

p.1132以下の「第二節 明治後の私塾及び私立学校」以降、町内の学校に関する詳しい記録があり就中、明治6年に中尾村宝樹院に仮設された中尾学校を期限とする一宮北小学校（終戦当時の相興小学校）の昭和42年当時の写真が1145頁に掲載されている。

４－９ WEB サイト一覧

（１）国総研ホームページからの技術情報の公開

プログラムの公開

http://www.nilim.go.jp/lab/bcg/program.html

技術資料の公開

http://www.nilim.go.jp/lab/bcg/siryou/index.htm

本書に関する研究成果

http://sim.nilim.go.jp/MCS/phi

奥尻島関連のデータ等

http://sim.nilim.go.jp/Okushiri

筑波関連のデータ等

http://sim.nilim.go.jp/Tsukuba

（２）関連機関

奥尻町

http://town.okushiri.lg.jp/

国立研究開発法人建築研究所

http://www.kenken.go.jp

国会図書館：東日本大震災アーカイブ（ひなぎく）

http://kn.ndl.go.jp/

インドネシア公共事業省研究開発総局人間居住研究所

http://puskim.pu.go.id

大韓民国農漁村公社農漁村研究院

http://rri.ekr.or.kr

４－１０ 付録 DVD-ROM の構成

（１）ソースコード

①基幹部分

②VC-1C ビルド

③VC-1C_D ビルド

④VC-2V ビルド

⑤VC-3M ビルド

⑥VC-4D ビルド

（２）セットアップ

①VC-3M(奥尻：Android)

②VC-3M(百人町：Android)

③VC-4D(Web アプリケーション)

（３）サンプル・コンテンツ

①シンプルな、データファイル＋メタファイル（機能テスト用）

②ファイル形式別のメタファイル例

③出力用メタファイル

（４）研究報告のデータ(pdf 形式)

（５）シンポジウム記録(動画)

国土技術政策総合研究所研究報告
RESEARCH REPORT of ＮＩＬＩＭ
Ｎｏ．６２　　February　2019

編集・発行　©国土技術政策総合研究所

〒305-0804　茨城県つくば市旭１番地
企画部研究評価・推進課　　TEL 029-864-2675