２．景観シミュレーションの幾何学的基礎とデータ構造

２－１．三次元モデル

景観を任意の視点から検討するためには、検討対象である地物（地形や構造物など）が三次元空間の図形によって表現されている必要がある。景観として見るためには、地物の各要素の内部に関する情報は必ずしも必要ではなく、表面の形状と光学的性質（色彩、反射率、輝度、テクスチャ（色柄）など）が記述されていればよい。モニタ画面やプリンターに出力する画像（透視図など）は二次元画像である。この画像を得るためには、光源と物体と視点の位置関係から射影変換を行い、スクリーン上の画像を計算する。この計算処理はライブラリ化されており、景観シミュレータにおいては、OpenGL のライブラリを使用している。このライブラリは、Windows 系の OS には、標準で提供されている(OpenGL32.dll, GLU32.dll)。

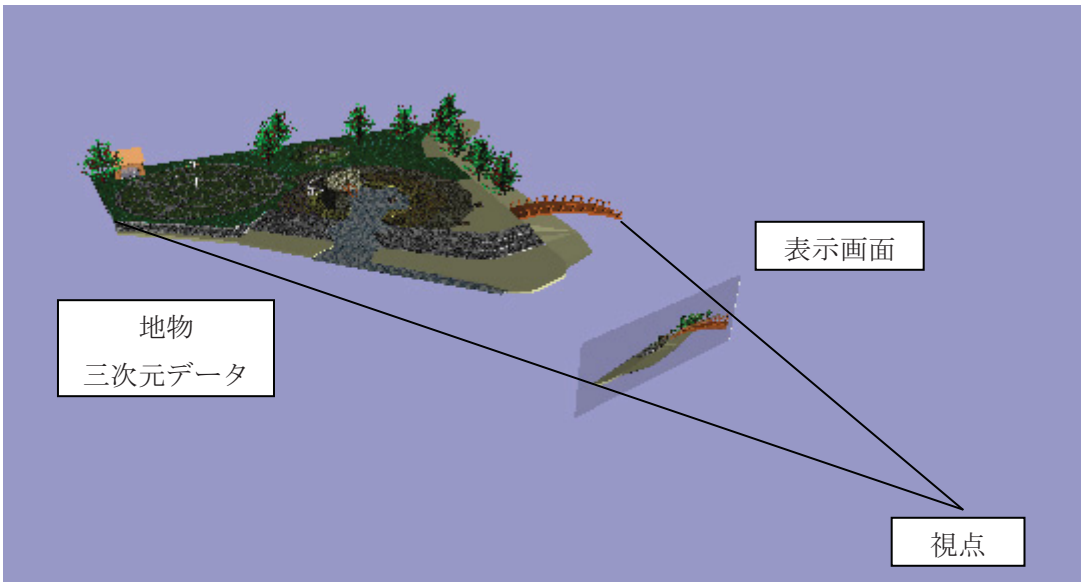


図２－１：射影変換（地物三次元データ・視点・表示画面）

景観シミュレーションを実用的に実施するためには、景観を構成する地物を表面の集合体として表現するデータ構造、それを外部ファイルとして入出力するファイル形式、同じ地物に関して表示の条件となる視点設定や光源設定、データの構築と削除などに関する各種処理機能が OpenGL による表示の一つ上の階層を構成する基盤的なシステムの要件として必要となる。これらの基本的機能の多くは、OS に依存しないライブラリとして用意した。国土交通省版・景観シミュレーション・システムの特徴の一つは、地物に固有の属性（形状、表面の光学的特性など）と、表示段階で必要となる地物以外の条件（視点設定、光源、時間）を、データ構造、および外部ファイル形式において明確に分離している点にある。

三次元の立体である地物の単位（地面や構造物など）を、表面の形状として表現する方法は、サーフェス・モデルと呼ばれる。

曲面を表現する方法としては、適切な多角形に分割して表現する方法を用いている。数

学的には、曲面を有する幾何学図形（球や円柱など）を関数とパラメータで表現する方法が可能であり、景観シミュレータにおいても、主にデータを軽くするために用いているが、表示のために OpenGL ライブラリを使用するため、出力段階では、物体の表面を多角形（ポリゴン）に分割している。そのような場合における分割数は、個別のデータには記述せず、環境設定で設定し、形状生成段階で自動的に処理している。

立体を構成する多角形を、多数の小さな三角形に分割するような方法と、ひとつの平面図形については、可能な限りひとつの多角形で表現しようとする方法がありうる。構造物等を扱うことの多い本システムにおいては、基本的には後者の方法を採用しているが、その中で前者の方法に基づいて作成されたデータを扱うことも可能である。

三次元空間内の多角形の幾何学的形状と光学的性質を記述するデータ構造（三次元ポリゴン）について、以下に説明する。

２－２．三次元ポリゴン

三次元ポリゴンとは、実空間、より正確には実空間をモデリングするためのユークリッド幾何学における三次元空間（以下、簡単に「実空間」と記す）に配置された、少なくともｎ個（≧３）の頂点を有する多角形を表現するために、コンピュータ上で座標点の配列として表現されたデータである。幾何学的には、頂点の数に等しい数の有限の長さの辺を有している。

ｎ頂点の三次元ポリゴンは、コンピュータ上のデータとしては、頂点を XYZ の座標値として表現したものを要素とする、長さｎの配列として表現される。景観シミュレータにおいては、以下の d3Face 構造体によって表現している（d3dml.h）。

リスト２－１：d3Face 構造体による面の定義

```
struct _d3Face {
/*private:*/
        d3Vertex *vl;           頂点リスト（配列）
        d3Face *next;           次の面へのポインタ
/*public:*/
        unsigned long va_f;     面の属性フラグ（法線、カラー、テクスチャ）
        unsigned long va_v;     頂点の属性フラグ（法線、カラー、テクスチャ）
        int vnum;               頂点数
        int material;           マテリアルID
        int texture;             テクスチャID
        float n[3];             法線ベクトル
        float c[4];             カラー
        int display;
        int shape;              形状種別（面、凹ポリゴン、折れ線、異常データ）
};
```

一つの d3Face 構造体（メモリ・ブロック、長さ：64 バイト）毎に、長さが vnum の頂点列(d3Vertex*)が定義される。個々の頂点は、三次元座標値を基本的な情報として持っているが、この他に、必要であれば、頂点毎の色彩や、テクスチャ座標（図柄の位置合わせ

に関する情報)、法線の情報をもつ。座標以外にどのような情報が登録されているかに関して、ポリゴンの va_v というフラグ、および頂点毎の va というフラグに記述している。法線や色彩に関する値が設定されていても、フラグが OFF となっていれば、表示には使用しない。

リスト２－２：d3Vertex 構造体による頂点の定義

```
struct _d3Vertex {
/*private:*/
/*public:*/
        unsigned long va;           頂点座標以外の属性の有無を示すフラグ
        double v[3];                頂点座標
        float n[3];                 頂点の法線ベクトル
        float t[2];                 頂点のテクスチャ座標
        float c[4];                 頂点のカラー値
};

注：va に設定する頂点毎のフラグ
#define D3_VA_NORMAL      0x0001      法線
#define D3_VA_COLOR       0x0002      色彩
#define D3_VA_TEXTURE     0x0004      テクスチャ
#define D3_VA_VIRTUAL     0x0008      仮想線の起点
```

内部のデータ、及び外部ファイルとして保存されるデータにおける座標値を記述するための座標軸の設定は、景観シミュレーションが対象とする空間尺度においては、慣習的に東を X 軸、北を Y 軸、上を Z 軸としている（CAD システムの慣習。コンピュータグラフィックス系のシステムでは、カメラが水平向きの場合に、レンズの軸を Z 軸とし、上を Y 軸とする場合もある)。

実空間に存在する多角形であれば、表側と裏側を定義することができる。景観シミュレータにおいては、上記の頂点列が反時計回りに見える側が表側である。

しかし、コンピュータ上では、現実的な図形としては許されないようなデータ（三次元ポリゴン）も作成が可能である。例えば、４の頂点を有するが、そのすべてが同じ座標値となっているようなデータである。従って、三次元ポリゴン・データとして多角形の図形処理を行うためにはいくつかの制約条件や前提条件が必要であり、形状生成や形状演算の処理結果については、適切な異常チェックを行わなければならない。以下、いくつかの例について示す。

（１） 頂点の数が２以下の面

面として描画が不可能である。点または線分に変換する必要がある（景観シミュレータでは、線のデータは表示可能であるが、点のデータは扱っていない）

（２） 多重頂点または、長さゼロの辺

同じ座標値を有する点が２以上連続する場合、見かけ上頂点の数が少ない多角形として表示される。演算処理によっては、不具合を起こす場合がありうることから、冗長な頂点は削除する必要がある。

（３） 連続する一直線上の辺

連続する辺が一直線上に連なる場合、中間の頂点は図形的に意味がない。但し、一直線上を折り返すような幅のない「ヒゲ」状の図形であって、面の内側に向かうものは、図形演算過程で生じる切込や切れ目を表現する準正常な面として許容している（２－６）。

（４） 辺が自己交差する図形

ひとつの図形を構成する辺どうしが交わり、８の字のようなねじれた図形となっている場合、現実にはありえない図形となる。

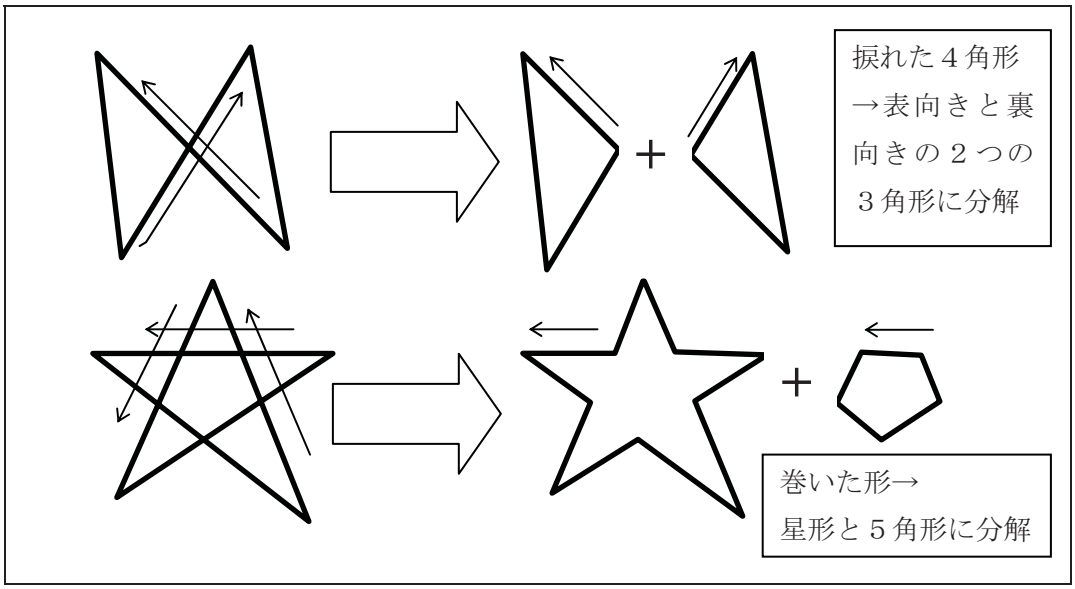


図２－２　辺が自己交差する図形と、正常な図形への分割

図形演算の結果このような図形が発生した場合には、交差する点に新たな頂点を追加した上で、表向きの部分、裏向きの部分、多重の部分を別の図形に分解処理している。

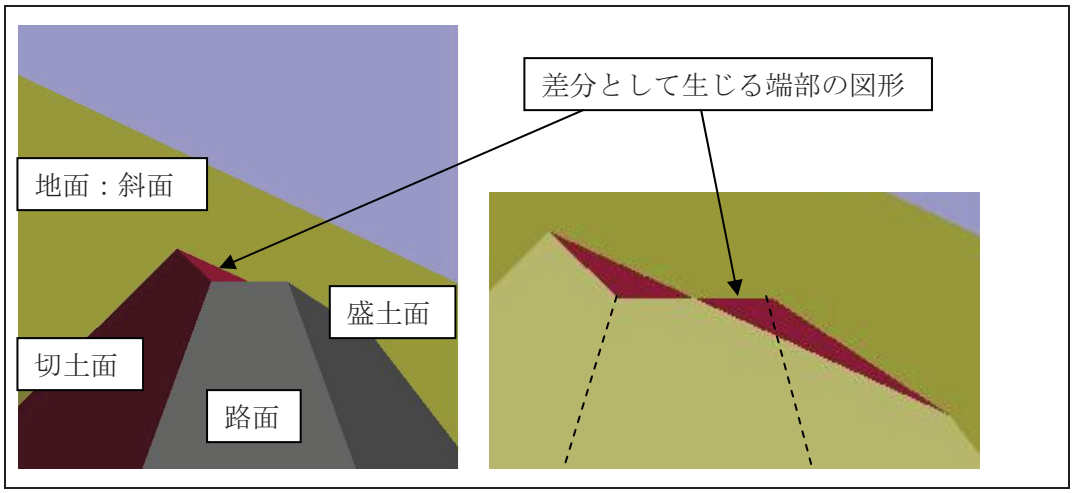


図２－３　地形と構造物の図形演算において端部に自己交差する図形が生じる例

例えば、地形の上に、切土・盛土による法面を有する道路を造成する場合、端部に、差分として捩じれた図形が生じる場合がある（図２－３）。これを表・裏で複数の面に分割し、表側を地形の上に露出した路面の端面として、また裏側を地形の下に埋没した路面の端面

として付加することにより、完結した正常な立体とすることができる。

（５） ひとつの平面上にない多角形

四以上の頂点を有する図形の場合、全頂点が同一平面上にはない図形がデータとしては生成しうる。極言すれば、浮動小数点で各頂点の三次元座標値を表す限り、多かれ少なかれ平面からのずれは生じる。ずれが少なければ、OpenGL による表示は、見た目には正常である。しかし大きいずれがある場合には、視点移動に伴い奇妙に変形する異常な表示となる。小さな図形（たとえば三角形）に分割することにより、異常な表示を解消することはできるが、分割方法は一意的ではなく、その仕方により、異なる図形が生じる。

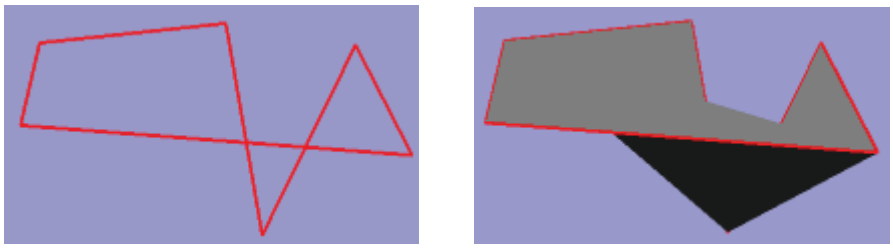


図２－４：頂点が同一平面上にないポリゴンと、OpenGL による表示状態（右）

数値地図のように、自由曲面としての地形をメッシュ型のデータで表現した立体を、サーフェス・モデルに変換する場合に、三角形ではなく四角形のポリゴンに変換すると、同一平面上ではないポリゴンが発生する。遠景の表示などにおいては支障がないが、図形演算処理においては支障を生じる場合がある。

（６） 凹ポリゴン

OpenGL に多角形データが出力された場合、ライブラリの内部処理において、最初の頂点を基準として、三角形に分割されて表示側に出力処理される。このため、多角形の頂角が 180 度よりも大きい頂点を有する凹多角形が正常に表示されない場合がある（図２－５左）。

景観シミュレータにおいては、多角形に凹ポリゴンであることを示すフラグを明示的に与えることにより、正常に表示を行うことができる（処理時間は長くなる）。このことを実現するために、フラグを有するポリゴンの表示に際しては、ポリゴンを OpenGL に出力する前に、プログラムの側で三角形に分割し、出力している。

凹多角形は、表裏に関しては明確であり、実空間に存在しうる図形である。

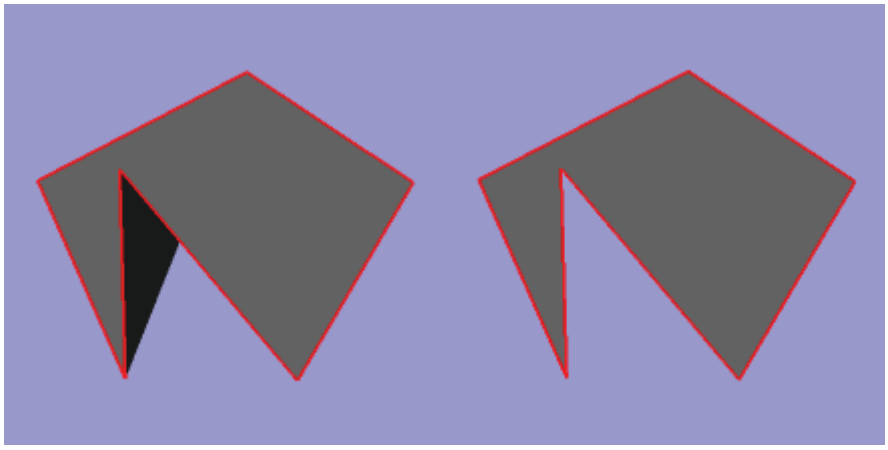


図２－５：凹ポリゴンの表示状態（左）と、分割出力による表示結果（右）

（７） 穴あき図形

図形の中に穴がある場合に、これを記述するデータ構造には、処理システムにより様々なものがある。窓がある建物の壁面など、穴あき図形は実空間においても頻繁に出現する。景観シミュレータにおいては、穴あき図形を表現するために仮想線法を用いた。この方法では、図形の外周を成すポリゴンと、穴のポリゴンを結ぶ仮想の２辺（橋）で結び、恰も穴が無いポリゴン（但し、その内の２辺が、逆向きに重複している）のように表現する。この橋の部分は、図形が生成された時点で頂点座標の比較を行い、仮想線であるフラグを辺の開始点側の頂点(d3Vertex の va メンバ)に付している。ワイヤーフレーム表示において、このフラグを有する辺を表示しないように処理している。但し、仮想線が存在することを確認するために、ポリゴンを含むグループが選択された際の強調表示（編を赤線で表示）に際しては、仮想線も表示している。

これにより、穴あき図形も、幾何学的演算において通常の凹ポリゴンと同様の処理を行うことができる。穴あき図形における仮想線は、他の辺や仮想線と交差してはならない。図形演算により、新たな辺や仮想線が生じ、これが既存の仮想線と交差した場合には、仮想線を適当な位置に付け替える処理を行う。

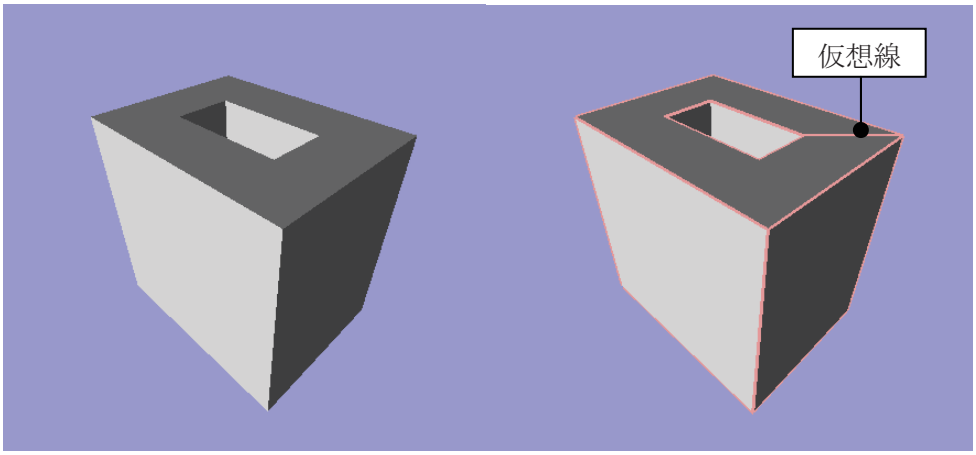


図２－６：穴あき図形と、仮想線

（８）その他

幾何学的には問題なくとも、座標値が不適切な図形が、実務上の支障となる場合がしばしば発生する。国家座標系を用いたモデリングでは、景観検討対象地域を構成する地物の座標地は通常大きな値（原点から数十～数百 km）を有する。作業中のデータに、原点付近の座標値を有する小さな物体が追加されると、それを発見し除去するためには手間を要する場合がある。また幾何学的に正常で、有限の値ではあるが、非常に小さなサイズの（不要な）ポリゴンが発生すると、ノイズのように障害となる場合がありうる。

２－３．立体の完結性（閉多面体）

複数の面により構成された立体が、「閉多面体である」とは、立体の表面が面により隙間なく埋め尽くされていて、かつ隣接する面が同じ向きであることが必要かつ十分な条件である。このことは、全ての辺が、二つの面により共有されていて、その辺を共有する二つの面の向きが等しい事、換言すると、その辺を共有する二つの面の頂点リストにおいて、その辺が逆向きに辿られていることをもって検査できる。いわゆる「メビウスの帯」のような図形、あるいは「クラインの壺」のような図形においては、辺を共有する面の向きがどこかで逆になるために、完結した立体とはならない。しかし、トーラス（ドーナツ形）のような図形は、閉じた図形となる。

閉多面体であれば、任意の点を頂点とし、各面を底面とする立体の体積（但し、高さが負ならば負値とする）の合計として、立体の体積を計算することができる。完結した立体の場合には、この頂点がどこにあっても、計算される体積は一定である。立体が閉じていなければ、頂点位置により不定となる。

また、立体であって、内部に「泡」のような空洞ないし中空部があり、その泡の面が、内側が表となるような閉じた面で構成されている場合、これもまた閉じた図形となる。

内部に空洞（泡）のある立体は、構成するすべての面を、ひとつのグループに帰属させることにより、表現することが可能である。

面＋頂点－辺＝２×（１＋ｎ）　ｎは空洞の数

という関係が成立する。

内部の空洞ではなく、複数の互いに離れて独立した閉多面体を構成する面群が、データ上ひとつのまとまりになることがありうる。このような場合においても、面＋頂点－辺が２とは一致しない。

一つの泡だけを取り出したような図形は、閉多面体ではあるが、体積が負値となり、一般的には不自然である。しかし、例えばトンネルや地下道、あるいは建物のインテリアなど、中空の空間部分だけを取り出して景観検討するような場合には、実用的な意味がある。

閉多面体を構成する面の面積ベクトルの合計はゼロベクトルとなる。しかし、逆は必ずしも真ではない。

ある任意の視点から閉多面体を見たときに、その立体を構成する各面の立体角の合計は、

その視点位置が立体の外部にある場合ゼロとなり、内部にある場合は４πとなる（視点位置の移動に伴い離散的に変化)。完結しない多面体の場合には、視点移動によりこの値が連続的に変化する。

２－４．立体の自己干渉

閉多面体であっても、それを構成する面が相互に干渉する場合が、データとして生成しうる。その場合、交差した範囲において、多重の「内側」となる領域が成立する。現実の空間と物質の世界においては、このような状態は存在しえないと考えられるので、論理チェックすることが望ましい（以後の図形どうしの演算が保証されない)。

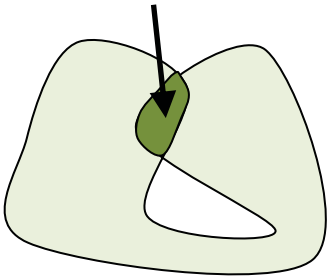


図２－７：立体の自己干渉

２－５．立体図形間の演算

立体 A と立体 B が干渉する場合に、例えば立体 B の内部に含まれることになる立体 A の部分を切除するような演算（CAD 等においては慣習的に図形の BOOL 演算と呼ばれる）の必要性が高い。例えば、山岳地に道路を通す場合、盛土・切土あるいはトンネルなどにより地形を加工する必要がある。その場合、技術基準や設計指針に基づいた法面の形状は、道路の形状から自動的に計算することができるが、計算された法面の内、地形と干渉する領域を計算し、その部分だけを法面として付加する必要がある。同様に、地形の側でも、法面として削られる部分、及び盛土により覆い隠される部分については、削除する必要がある（後者は不可視)。

これを実現するためには、いくつかの方法がある。初期の景観シミュレータにおいては、OpenGL が有するデプス・バッファを用いて実現していた。この方法は、地形と構造物・法面を真上から平行投影した画像を生成し（必ずしも表示する必要はない)、最終的に表示として残されたものが各地点において地形か、構造物・法面のいずれであるかによって、各地点における地形と構造物・法面の上下関係を判定するものである。この方法では、使用する OpenGL 画面の縦横寸法と対象エリアの面積で決まる格子間隔のメッシュ・データで上下判定を行うため、厳密な面と面の交線ではなく、概略の形状が得られるのみである。このため、複雑な地形・道路断面により発生する概略の法面形状を短い時間で検討するためには適しているが、近くから眺める価値のある正確な形状演算ではない。

より厳密な形状演算を行うためには、干渉しあう立体と立体の間で、関係するすべての面について演算処理を行う必要がある。現在の版の景観シミュレータにおいては、仮想線を含む面と面の演算に還元する方法を採用している。これにより、立体どうし図形演算を、切断される立体の全ての面と、切断する立体の全ての面の、全ての組み合わせに対して、面どうしの図形演算を適用する方法に還元することができる（立体１がｍ枚、立体２がｎ

枚の面を有する場合、ｍ×ｎ回の面と面の演算となる。但し多くの場合は空間的に離れていて相互に干渉しないため、存在範囲等の簡単な事前チェックのみで処理は終了する)。

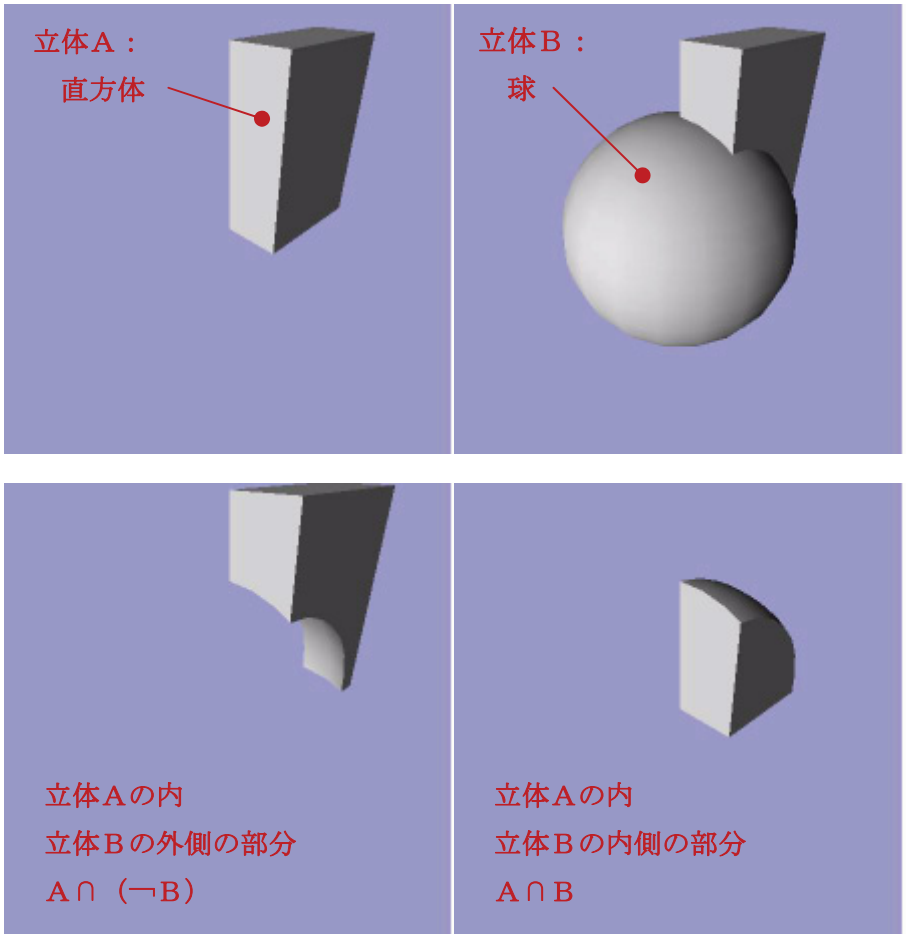


図２－８：箱と球の図形演算

２－６．面の準正常性

立体どうしの形状演算を、面と面の演算に還元するために、面の「準正常性」の概念を導入した。通常の正常な多角形であれば、それを構成する二つの辺が重なり合うことはない。これに対して、「準正常」な面においては、「幅のない切込み」を許容し、これを「反対向きに重なる二つの辺」により表現する。これには、頂点から内部に伸びるもの、辺の中間点から内部に伸びるもの、及び図形の内部に独立した切り込みの３種類がある。切断多角形が被切断多角形の全幅に亘り完全に交差し、被切断多角形が二つに分断される場合には、切り込みの形は生成せず、二つ（以上）の多角形が生成する。しかし、切断多角形が被切断多角形と部分的に交差する場合には、切込みの形状が生成する。適用する立体を構成する面を次々と被切断多角形に対して適用することにより、最終的には、切込みが次第に成長し、最終的には、切断される。このプロセスの中間段階を記述するための、拡張した面の概念である。現実世界においても、紙の多角形にハサミやカッターで切込みを入

れたような状態は存在している。

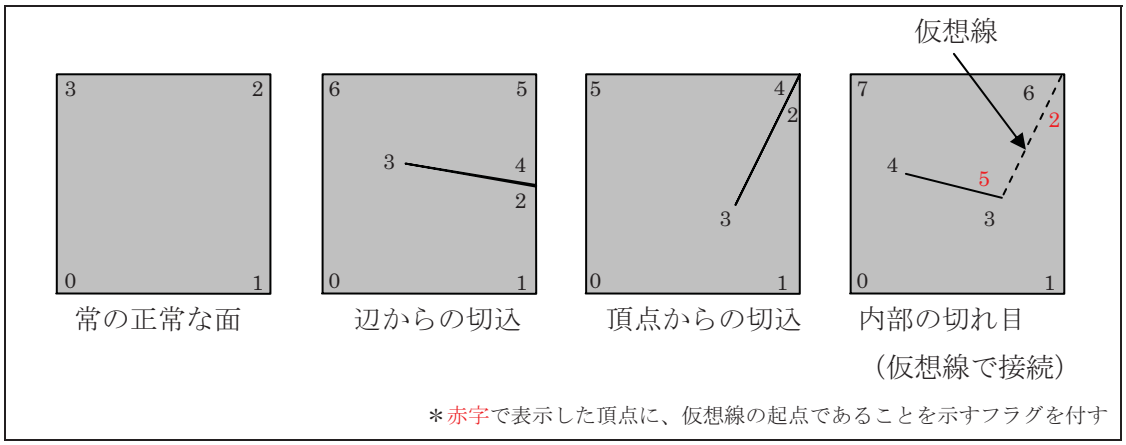


図２－９：準正常な面の例

その際、面と面の演算の結果は、媒介的なデータ無しに、処理結果としての面の形状として表現しており、どの中間段階においても、表示することができる。従って、演算処理がエラー終了した場合に、直前の状態を表示確認することが容易となった。

２－７．面の内外判定

適用する立体の各面を、被切断多角形に次々と適用することにより、切込や切目は次第に（折線状に）延長してゆき、最後に切断多角形により被切断多角形が切断され、切れ込みが解消した時点で、被切断多角形から分離独立した各部分が、切断立体の内側にあるか外側にあるかが確定する。従って、個別の切断に際して、分類を行っておき、最終的に、被切断立体の全ての面の処理が終了した次点で、分類を行うことにより、切断された立体の形状を得ることができる。

切断面との接触がない面に関しては、切断立体との内外関係による判定を行う。切断に関係しない（相互接触しない）面の内外判定のためには、切断立体が閉じた図形になっている必要がある。切断立体の側の面群が閉じていない不完全な場合であっても、交差部分が閉じた折れ線として完結していれば、切断される側の立体を構成する面群における、切断が生じた面との接続関係から辿って内外を判定することは可能であるが、まだ実装していない。

２－８．立体間の相互演算処理

被切断立体の各面から、切断立体の内部に包含される部分を取り除く処理の結果生成する図形は、閉じていない。そこで、この開口部を、切断図形と同じ表面形状で型押ししたように塞ぐことにより、閉じた立体図形を生成することができる。この塞ぐ部分を得るために、切断図形と被切断図形を入れ替えて演算処理を行い、切断の結果内部に取り込まれる部分だけを取り出し、これを先ほどの開口部に取り付ける。「切断図形を押し付けて凹ませた」ように見立てることができる（図２－８）。

２－９．テクスチャ座標

画像データをテクスチャとして面に適用することにより、恰も壁紙を貼ったような効果を得ることができる。テクスチャのピクセルは、RGB とアルファ値を持つことができ、アルファ値が零の点は透明となる。これにより、樹木などの複雑な外形を有するオブジェクトの画像を単純な長方形に適用するだけで、擬似的に樹木を表現することができる。但し、このような透過属性をもつ画像を保存できるデータ形式は限られている。

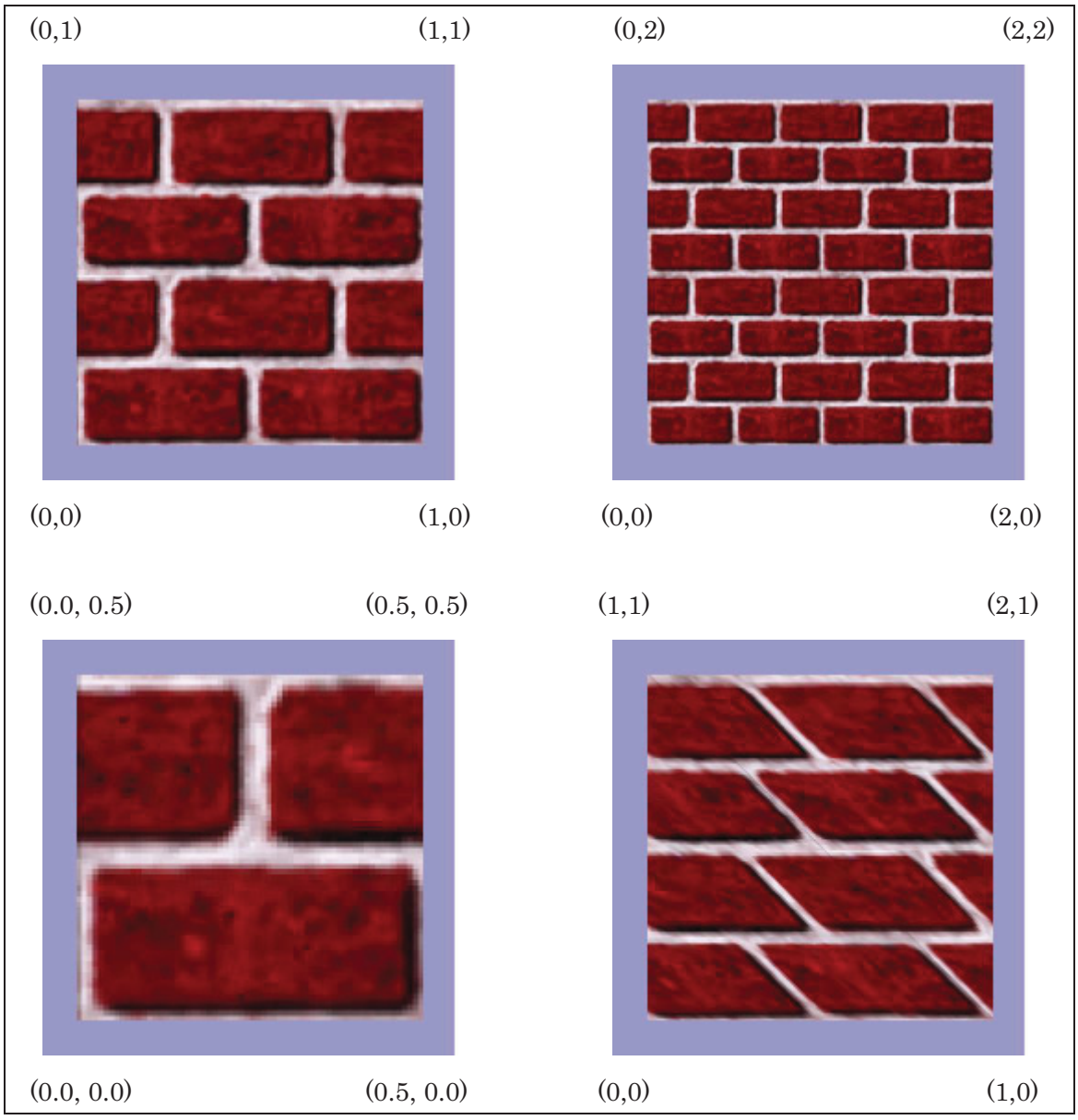


図２－１０：テクスチャ座標設定

図形の側の各頂点にテクスチャ座標（二次元）を定義することにより、貼る位置や縮尺を変えることができる。長方形の画像の中での縦・横の位置は、画像サイズにかかわりなく、それぞれ、0.0～1.0 の範囲の数値で示される。従って、テクスチャ座標として、例え

ば横が 0.3 と 0.6 の値を二つの頂点に適用すると、画像の中間部分だけが使用される。

0 より小さい値、1 よりも大きな値を適用した場合には、格子状にテクスチャを敷き並べた平面から、その座標の範囲を取り出したように、テクスチャが繰り返し適用される。たとえば、0.0 と 10.0 を適用すると、その間にパターンが 10 回繰り返される。従って、タイルやレンガのような、パターンが繰り返される素材をテクスチャとして適用する場合には、最小繰り返し単位となっている部分を画像データとして用意し、繰り返し回数を表現するテクスチャ座標を頂点に適用することで、大きな面を小さなテクスチャ画像（即ち、小さなデータ）で表現することができる。

曲面が複数の隣接する多角形で表現されている場合に、共有される頂点に同じテクスチャ座標を設定することにより、一つながりの曲面としてテクスチャを適用することができる。テクスチャ編集画面においては、一つの立体を構成する多くの面の各頂点に対して、このような座標計算を一括で自動的に行い、一斉にテクスチャを適用するような機能を用意している（４－４(22)(26)(27)）。

２－１０．法線ベクトル

法線ベクトルは、面および頂点に対して定義することができる。通常の多面体の場合（直方体や角錐）、各面の法線は、面に垂直のベクトルである。法線ベクトルが面や頂点に定義されていない場合には、自動的に面に垂直の法線ベクトルがあるものとして表示を行う。

面に明示的に法線が適用された場合、表示に際しての面の明るさは、その法線に垂直な面として計算されるため、例えばタイルを平面的に敷き並べたような物体の各面に対して、揺らぎのある法線を定義することにより擬似的に立体感（凹凸感）を表示することができる。

球や円柱など、曲面をもつ図形を、多数のポリゴンで近似的に表現する場合、共有される頂点に同一の法線ベクトル（例えば球の場合、中心からその頂点に向かうベクトル）を適用することにより、面の明るさが辺や頂点で連続することになり、滑らかな曲面として表示することができる。その場合、多面体を構成する各面は、平面ではあるが、頂点毎に異なる法線をもつことになり、明るさは面の内部で連続的に変化する。

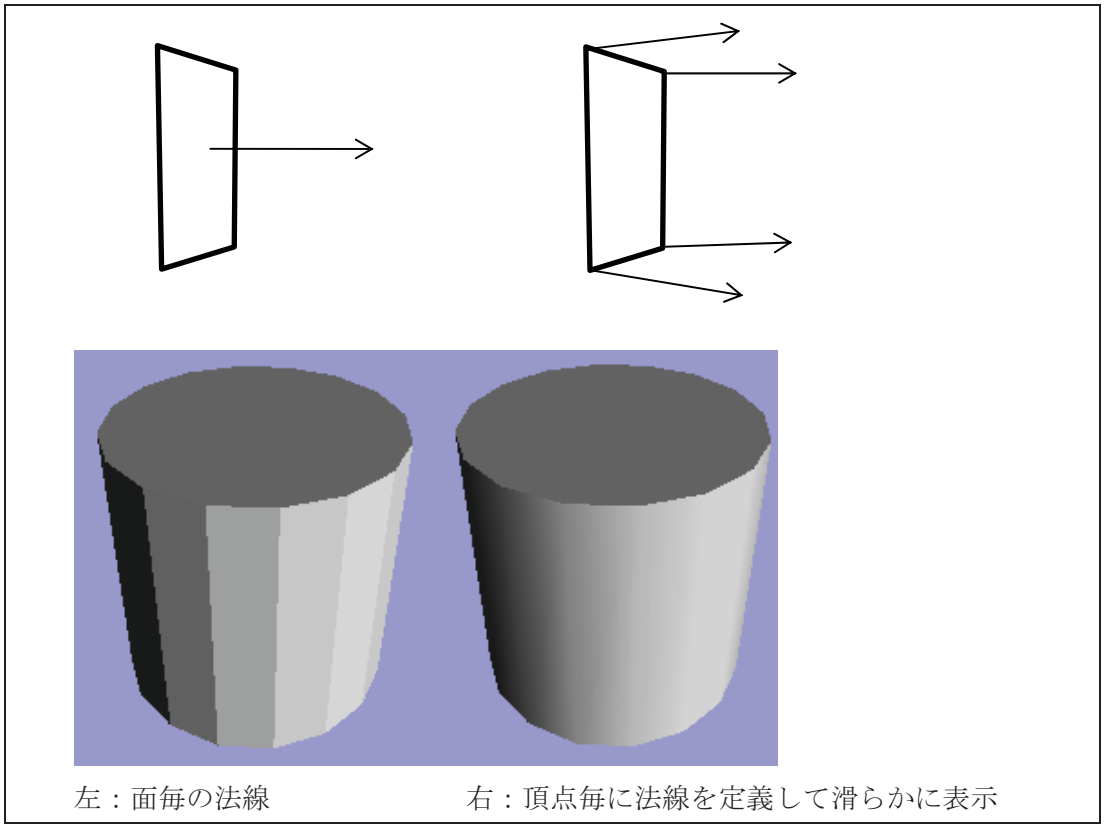


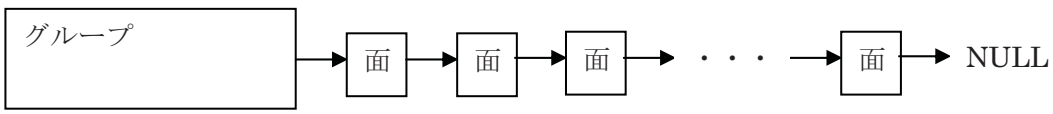
図２－１１：面の法線と頂点の法線

図形演算により発生する新たな頂点に関しては、既存の頂点のテクスチャ座標や法線ベクトルから補間計算し補うことにより、面の質感に係る情報を継承することができる。

２－１１．グループとリンク

景観シミュレータの内部において、「グループ」が重要な単位となる。面（d3Face）は、あるグループに帰属することにより、初めて存在することができ、表示可能となる。さらに、グループは、別のグループとリンク（親子関係）を結ぶことができる。これにより、CAD や GIS におけるレイヤーよりも複雑なデータ構造を表現することができる。

一つのグループは、複数の面をもつことができ、その数に制限はない。典型的には、一つの直方体等、閉じた多面体を一つのグループとして定義し、これに多面体を構成する複数の面を関連づける。しかし、閉じていない図形や、面を持たない（表示されない）グループも、データとしては存在することができる。特に、自らは面をもたないが、複数の部品を束ねる親グループとして機能するようなグループは頻繁に用いられる。



グループのポインタ f が最初の面を指す。面のポインタ next が次の面を指す。

図２－１２：面とグループの関係

グループは、リンクによって階層的に関連づけることができる。例えば、板と４本の柱から構成されるテーブルをモデリングするためにはいくつかの方法がある（図２－１３）。

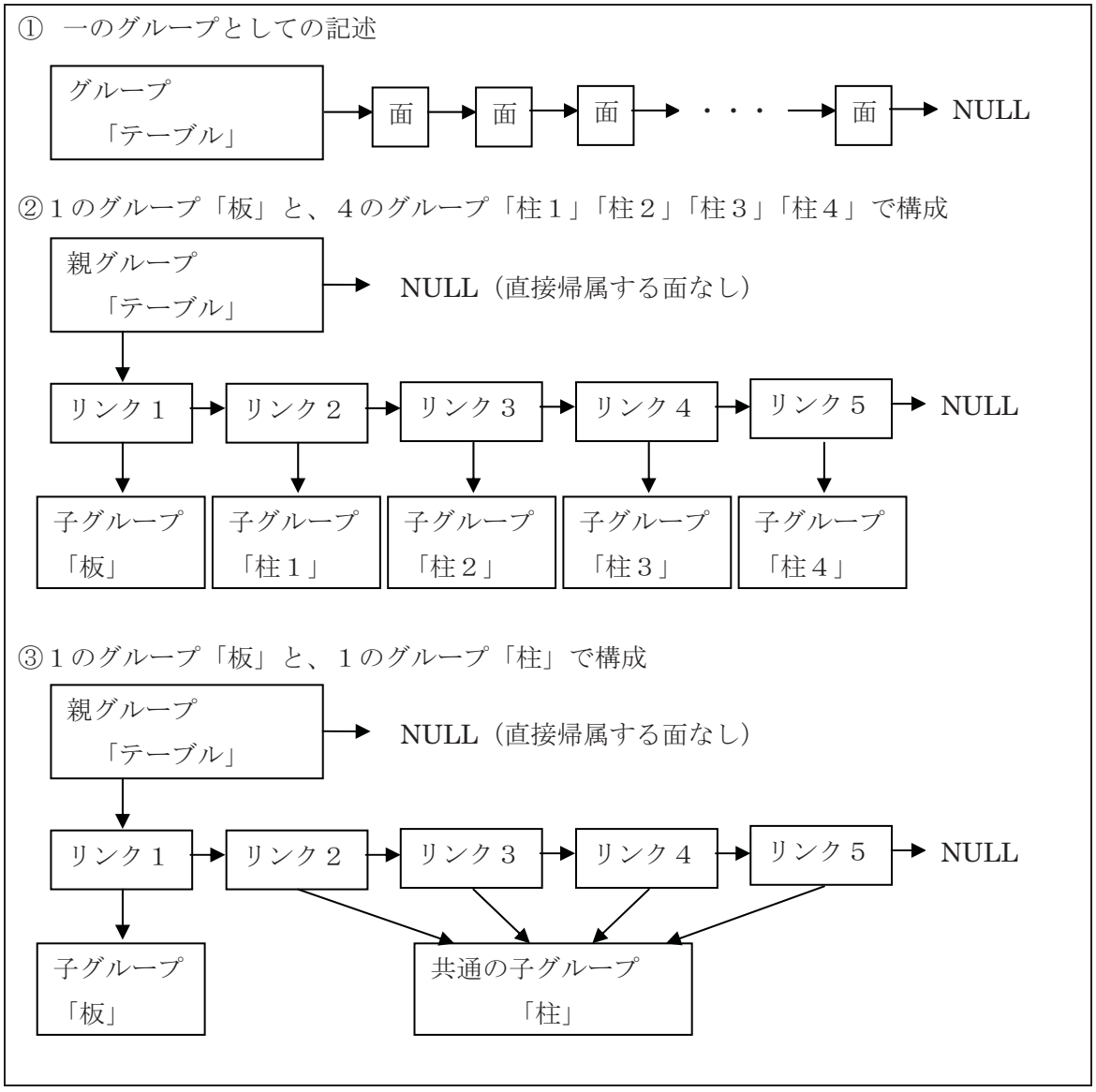


図２－１３：グループとリンクによる構成方法

単一のグループとして「テーブル」を作成し、全ての面をこれに直接帰属させる方法①、５の部品（板と４本の柱）を子グループとして定義し、これを親グループ「テーブル」にリンクする方法②、二種類の部品（「板」と共通の「柱」）を定義しておき、親グループ「テーブル」から板への１のリンクと、柱への４のリンクを定義する方法③が可能である。方法③の場合、それぞれの柱の位置を、リンク・マトリクスの違いによって指定する。共通の柱の側から見ると親グループは４存在するが、全て同じものである。

親グループに「板」を構成する面を直接定義してしまい、子グループの「板」を省略する方法も可能である。最後の方法が、最もメモリ資源を節約できる。

グループは、d3Group 構造体（メモリ・ブロック、サイズ＝144 バイト）により定義している。

リスト２－３：d3Group 構造体によるグループの定義

```
struct _d3Group {
/*private:*/
        int grpid;              ID番号
        int dbno;
        int num_u;              /* 親グループの数 */
        int num_d;              /* 子グループの数 */
        int material;           マテリアル
        int texture;            テクスチャ
        double xmin, xmax, ymin, ymax, zmin, zmax;/* 存在範囲 */
        d3Link *link;           子グループへのリンク
        d3LinkUp *lup;          親グループへのリンク
        d3Face *f;              面
        void *display;          ディスプレイ・リスト
        void *user[D3_USERDATA_MAX];  属性データ
/*public:*/
        char *name;              名称
        char *kind;              ファイル情報
        int type;                種別
};
```

グループに直接的に所属する面は、f メンバに、面を定義する d3Face 構造体（メモリ・ブロック、サイズ＝64 バイト） へのポインタを格納することにより定義される。複数の面は、d3Face 構造体の next メンバを用いてポインタでつなぐことにより定義される。グループに直接的に所属する面がない場合には、f メンバには NULL が格納される。

グループに帰属する子グループが存在する場合には、link メンバに、d3Link 構造体（メモリ・ブロック、サイズ＝144 バイト）へのポインタを格納する。複数の子グループが存在する場合には、d3Link 構造体の next メンバで複数のリンクを定義する。また、子グループの lup メンバに、d3LinkUp 構造体（メモリ・ブロック、サイズ＝8 バイト）へのポインタを格納することにより、親グループの所在を示す。

表示を行う際には、親グループから子グループに再帰的に辿るだけですべての面にアクセスすることができる。しかし、モデリング操作により、グループやリンクを削除する場合には、親グループにアクセスするために、d3LinkUp を用いる。一つのグループが、複数の親グループからリンクされている（例えば、共通のありふれた部品として参照されている）といったような場合が存在するため、d3LinkUp 構造体も、next ポインタでつなぐ構成としている。これにより、一つのグループから、子グループと親グループにアクセスすることができる。上記のテーブルの例に関しては、テーブルの子グループは複数（１の板と４の柱）存在する。一方、柱から見ると、親グループが４存在することとなる。また、このテーブルの柱が、子グループとして別の親グループにもリンクする（部品として参照される）ことができる。

一つのグループに関する、子グループの数、及び親グループの数は、それぞれ、num_u、num_d メンバに整数値として記述される。

リスト２－４：d3Link 構造体によるリンクの定義

```
struct _d3Link {
/*private:*/
        d3Matrix mat;               親グループとの位置関係を表すマトリックス(4x4)
        d3Group *parent, *child;    親グループ、子グループへのポインタ
        d3Link *next;               次のリンクへのポインタ（末尾の場合NULL）
/*public:*/
};

struct _d3LinkUp {
/*private:*/
        d3Link *up;                 親グループからのリンクへのポインタ
        d3LinkUp *next;             次のd3LinkUp構造体へのポインタ(末尾の場合NULL)
/*public:*/
};
```

リンクには、親グループと子グループの位置関係を示すためのマトリックスが定義される。これにより、上記のテーブルの例では、同じ形状の足を異なる４か所の位置に配置することができる。例えば、山（地形）に 10,000 本の桜の木（ファイル部品）を配置した場合、グループは山と桜の木の２個しか生成されず、その間に 10,000 個のリンクが生成され、それぞれが異なるマトリックスをもつこととなる。

モデリング作業においては、例えば配置作業においてグループとリンクの処理が行われる（４－４(11))参照)。ユーザーが部品等を定義したファイルを選択し、位置を指定して、現在の地物の中に新たな要素として付け加える場合に、単体配置を繰り返す方法、リニア配置（指定したラインに沿って指定した間隔で街灯などを配置)、エリア配置（指定した領域に指定した密度で樹木などを配置）の方法がある。単体配置の場合、配置する要素を記述するグループを具体的に構築するのは最初の１個に関してのみであり、２個目からは新たなリンクが追加される。またリニア配置・エリア配置においては、１個のオブジェクトに対して必要個数のリンクを作成する。例えば、山に 10,000 本の桜の木を配置する場合に、桜の木を記述するグループは一つでよい。逆に、このようにして配置された桜の木の１本を削除する操作は、リンクを一つ削除する内部処理により実現される。最後の１本の削除の際に、同時に実体としての桜の木のグループを削除する処理を行う。

言うまでもなく 10,000 本の桜の木を個別的にデータとして作成することもできる。しかしその場合、メモリ上で、あるいは外部ファイルに保存した場合のデータサイズははるかに大きくなる。

グループには、マテリアル、及びテクスチャを定義することができる。グループに所属する面にマテリアルまたはテクスチャが定義されていない場合には、グループに定義されているものを適用する。

また、子グループにマテリアルまたはテクスチャが定義されていない場合には、親グル

ープに定義されているものを適用する。カラーは、グループに直接定義することはできず、面または頂点に定義する。

グループには、name、kind、type の３の属性（必須）が定義される。

name（名称）は、グループのユニークな名称を示す文字列である。モデリングにおいて、ユーザーが指定しなかった場合には、システムの側で、”G314” 等のユニークな名称を自動的に生成して付す。

type （整数型）は、グループの種類を示し、０（通常）、１（ファイル型）、３（編集されたファイル型）がある。１（ファイル型）は、パラメトリックな部品、または固定的な部品を示し、データを外部ファイルに保存する際に、面を構成する頂点座標等の保存を行わず、ファイル名称または部品名とパラメータ値のみを保存する（詳細は３－３、及び５－１以下を参照)。３（編集されたファイル型）は、１（ファイル型）のタイプのグループを構成する面に対してユーザーが切欠きなどの形状変更や、面のカラー編集を行った場合に設定される型で、もはや部品名とパラメータや、ファイル名称などによりコンパクトに定義できない情報を含んでいるため、ファイル保存に際しては、０（通常）のタイプのグループと同様に、配下の面の記述も出力する必要があることを示す。

kind （文字列型）には、通常”unknown”（文字列）が定義される。上記の type がファイル型であった場合には、ファイル名称（文字列）が登録される。ファイル型のグループは、部品を別ファイルとして参照する場合に用いられる。前記のテーブルの例③において、板と、共通の柱（４回使用される）を一つの外部ファイルの中で記述するのではなく、板と柱をそれぞれ別のファイルで部品として定義しておき、テーブルの定義ファイルにおける板と柱の記述は、外部ファイルを参照する形式をとることができる。このことは、繰り返し利用する価値の高い部品をデータベース化するために必要な機能である。

更に、グループには「地面」など、様々の属性を定義することができ、これは user メンバに登録される（詳細は、３－３参照)。

２－１２．リンク・マトリクス

リンク・マトリクスは４×４の行列として定義されており、d3Link 構造体の中に組み込まれている。データとしては長さ 16 の倍精度実数として定義される。

リスト２－５：基本的なマトリクス計算処理

```
定義
typedef double d3Matrix[16];/* OpenGL specification */
基本的な座標変換計算
void d3TransformPointd(double *p1, double *p2, d3Matrix m)
{
        double p[3], w;

        p[0] = p2[0];
        p[1] = p2[1];
        p[2] = p2[2];
```

```
        p1[0] = m[0]*p[0] + m[4]*p[1] + m[8]*p[2] + m[12];
        p1[1] = m[1]*p[0] + m[5]*p[1] + m[9]*p[2] + m[13];
        p1[2] = m[2]*p[0] + m[6]*p[1] + m[10]*p[2] + m[14];
        w     = m[3]*p[0] + m[7]*p[1] + m[11]*p[2] + m[15];
        if (w != 1) {
                p1[0] /= w;
                p1[1] /= w;
                p1[2] /= w;
        }
}
```

三次元空間内座標変換に用いられる。OpenGL の中で基本的な演算処理機能が提供されている。実空間における移動、回転を表現できるほか、座標軸毎の拡大縮小、鏡像変換やアフィン変換など、実空間における地物ではありえない変形も記述できる点は、表示の調整等において便利であるが、部品製作などのモデリングにおいて、このような機能を多用すると、データの具体的な意味（寸法や体積など）が照会された場合に処理が複雑になる点は注意を要する。

座標変換を伴わないリンクの新規設定処理においてはデフォルトで単位マトリクス（対角成分のみ１で残りが０、これを掛けてもベクトルは変化しない）が設定される。リンクにおいては、子グループの面の座標値を列ベクトルとして、前からマトリクスを掛ける演算を行う。データとしては、リンク・マトリクスは二次元ではなく長さ１６の一次元の倍精度浮動小数点の配列として表現する。これを M とし、子グループのローカル座標における座標値をベクトル P0（計算機上は長さ４の配列）、親グループのローカル座標における座標値を P1 とすると、座標変換は以下のように計算される：

$$\begin{vmatrix} P1[0] \\ P1[1] \\ P1[2] \\ 1 \end{vmatrix} = \begin{vmatrix} M[0] & M[4] & M[8] & M[12] \\ M[1] & M[5] & M[9] & M[13] \\ M[2] & M[6] & M[10] & M[14] \\ M[3] & M[7] & M[11] & M[15] \end{vmatrix} \times \begin{vmatrix} P0[0] \\ P0[1] \\ P0[2] \\ 1 \end{vmatrix}$$

このように、ベクトルに XYZ の３値に定数項を加えた４元とすることにより、１回のマトリクス演算で、回転だけではなく、平行移動も含めた計算処理を行っている。

親グループが、更に別グループの子グループとしてリンクされた場合には、上位のリンク・マトリクスが前から掛けられる。

実際のプログラムでは、結果の列ベクトルの４番目の数値が１でなかった場合、１～３番目をその値で除して調整している。

２－１３．マテリアル

景観シミュレータにおいては、面の光学的特性（色彩、質感など）を記述する方法として、直接カラー、テクスチャを数値やファイル名として指定する方法と、マテリアルの名称のみ指定し、その具体的な内容（カラーやテクスチャ等）別途マテリアル・ファイルの

中で定義することで間接的に指定する方法を二通り用意している。

実用的には、色彩がほぼ決まっている定番の材料に関して、マテリアルを用意しておくことにより、個々のオブジェクトに対して手間のかかる色編集を行わずに、マテリアルを適用することで作業が効率化する。頻出する素材（アスファルト、鉄、煉瓦等）や、日本塗料工業会（日塗工）の色見本帳を収録したマテリアル・ファイルを標準で用意している。

これに加えて、あるプロジェクトに関して、同じ材質を共通して適用する部位に関して共通のマテリアルを設定しておき、マテリアル自体を編集することにより、リアルタイムでの景観検討の手間を省くことができる。例えば、マンションのバルコニーの塗装を検討する場合、数多くのバルコニーの色を同時に編集するためには、例えば「バルコニー塗装」という共通のマテリアルを適応してモデリングを行った後に、このマテリアルの色彩を編集する方法が可能である。

マテリアルにはまた、経年変化を検討するために、経過時間（日数）で区切って異なるカラー値や、異なるテクスチャを与えることができる。これにより、木材の熟成や、塗装の経年劣化などを表現することができる。

更に、現在の景観シミュレータが出力部分でグラフィック表示に使用している OpenGL ライブラリでサポートされている表面光学特性の内、直接編集機能や外部ファイルでの記述を用意していない特殊な属性（輝度や鏡面反射率など）に関しても、マテリアルの中で定義し、表示に反映させることができる。

マテリアルは、グループや面に対して適用することができる。しかし、カラーのように頂点単位で細かく設定することはできない。また、マテリアルには法線ベクトルは含まれない。

表２－１：部位と設定可能な属性の対照表

	グループ	面	頂点
カラー	×	○	○
法線ベクトル	×	○	○
テクスチャ	○	○	テクスチャ座標
マテリアル	○	○	テクスチャ座標

あるグループに対して指定されたマテリアルは、そのグループに属する面、およびそのグループにリンクした子グループに対してデフォルト値として機能する。即ち、個々の面や子グループに関してマテリアルが指定されていれば、そのマテリアルが親グループに適用されたマテリアルに優先して表示に反映される。面にマテリアルが指定されていなければ、グループに適用されたマテリアルが表示に使用される。

景観シミュレータにおいては、最上位のルート・グループに関しては、デフォルトのマテリアルを固定的に設定しており、これはユーザーが編集できないようにしてある。これにより、全てのグループおよびこれに帰属する面のデフォルト値は、ルート・グループのデフォルトのマテリアルとなる。このデフォルト・マテリアルは、明るい灰色に対応する

カラー値{0.8, 0.8, 0.8}を持ち、その他の項目（テクスチャや輝度、反射率）を持たない。

面に関して、カラー定義を含んだマテリアルと、カラー値が共に設定されている場合には、後者の方が優先される（面の標示を行っている drawMuka(d3Face *f) の中で、setMaterialTexture(f->mat, f->tex) を実行した後に、setColor(float *c) を実行するため、前者が設定されても後者で上書きされる）。

マテリアルと個別のテクスチャ・ファイル名の両方が設定されていて、マテリアルの中にもテクスチャ・ファイル名の定義が含まれている場合には、表示において前者を使用する。

面および頂点に定義されたカラー値は、表示段階で setColor 関数の中で、diffuse[4]（光源に対応するカラー）と ambient[4]（環境光に対応するカラー） の数値に変換され、OpenGL ライブラリに送出される。この時、diffuse にはカラー値×1.0 が、また ambient にはカラー値×0.2 が渡される。

カラー値は長さ 4 の単精度実数配列として表現されている。その値は赤、緑、青の三原色とアルファ値となっている。数値の範囲は、0.0～1.0 である。アルファ値は不透明度を表し、低くなる程透明となる。透明な物体においては、描画段階で、既に描かれている背後の物体に物体色を上書きするのではなく、アルファ値に応じた比例按分により混合を行うことで実現されている。

OpenGL への最終的なデータ送出は、

```
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, amb);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, dif);
```

という関数により実行している。OpenGL の基本的な機構が状態マシンであることから、基本的には、この関数によりカラー値が設定された後、異なる値のカラー値が設定されるまでの間、送出される頂点には、設定されたカラーが適用される。しかし、Windows VISTA の OpenGL では若干仕様が変更されたように見える。即ち、glBegin-glEnd の内側で設定されたカラーは、その直後に送出された頂点にのみ適用され、以後の頂点には適用されない。このため、高速表示モード（７－６参照）において、GL_TRIANGLES、GL_QUADS 等の機能を用いて glBegin、glEnd を省略する場合、例えば三角形が連続する途中にカラーの変更が行われるような場合に、表示が従来とは異なる結果となる。このため、プログラムの側で面のカラー値が変化したかどうかについて検査を行い、必要であれば glEnd →カラー設定→glBegin を送出するように修正を行うことで対応した。

２－１４．線のデータ

景観シミュレータにおいては、線のデータ形式を用意している。景観を構成する地物の直接的な表現においては、線が具体的な構成要素となる場合は余りないが、作業の途上での補助線的な使用や、CAD 図面の変換結果を作業下図に用いる場合の表示などに便利である。また、道路や堤防などの長尺物の断面や中心線軌跡を線のデータとして予め用意して

おき、これを用いて掃引体等の形状生成機能（外部関数）により立体を生成するような用法がある。更に、平面として表示困難な図形（８の字形など）が生成した際にエラー処理する中で、線のデータに変換して表示し、原因の理解を助けるような用法がある。

線は、頂点の配列として表現されるため、データ構造としては面とほぼ等しい。従ってd3Face 構造体により定義し、属性として D3_SHP_LINE を付与している。表示にあたっては、最初の点から始まり最後の点で終わる折れ線として処理する。従って、面の場合とは異なり、頂点の順序を巡回的に移動させると、表示される実体は変化する。ループとして閉じた折れ線を表現するためには、最初の点と同じ座標値をもつ最後の点を定義しなければならない。

線に対しても、カラー、法線、テクスチャ、およびマテリアルを定義することができる。この内、線の表示に当たって直接的な意味をもつのは、カラー、およびマテリアルの中で表現されたカラーのみである。線分の標示におけるカラーは、光源とは無関係に、単純に定義された色で描画を行う。従って、暗闇の環境条件であっても同じように表示される。

法線、テクスチャおよびそれらの定義を含むマテリアルが線に対して定義された場合、これを断面形として用いて道路や堤防などの長尺物を生成した際に、生成される立体を構成する面にそれらのデータが反映される。

なお、面の表示との一貫性を保つために、カラーが定義された線であっても、ワイヤーフレーム表示（外形線のみの表示）においては、立体の稜の表示に使用するのと同じデフォルト色の線として表示する。