J3W入門 for Linux講座

水谷純
http://www.nk.rim.or.jp/~jun/index.html

第3回　物体の形状定義と表示

3Dアニメーションに関する連載にもかかわらず言語の解説ばかりでしたが、今回は物体の形状定義と表示という3D CGらしい部分を説明します。

■物体の形状

J3Wには3Dモデラー（GUIで物体を作成するプログラム）がありません。従って、頂点座標と多角形を構成する頂点番号を指定して物体の形状を作成する必要があります。非常に面倒なようですが、J3Wでは物体をプログラムで作成できるため、基本となるいろいろな形状の物体（プリミティブ）を作成するためのライブラリ関数を用意しています。

例えば、`j3w-643/j3c_script/lib/ssolid.j3c`では立方体と4角錐（ピラミッド）を定義する関数があります。同じディレクトリの`xsolid.j3c`や`zsolid.j3c`には3、4、6、8角柱や角錐、それらを組み合わせた任意の大きさの形状を作成できるようにライブラリ関数が用意されています。`revolutn.j3c`は任意の回転体（球や楕円体、N角柱など）を作成できるようになっています。`revolutn.j3c`で使っている継承の機能は次回以降で解説する予定ですが、それまでにJ3Cのプログラム例を読んで予習することをお勧めします。

J3C言語による形状の定義の前に、J3Wにおける、頂点と面（多角形）の関係を見ておきましょう。J3Wではすべての面には向きがあり、表からは見えますが裏からは見えない（透明）という性質があります。これは陰に隠れて見えない部分の処理（陰面処理）で、裏から面を見えなくする（背面消去）と面に関する処理が半分になり表示が速くなるためです。また、J3Wが使っている「塗りつぶし法」という3Dレンダリングで最も単純な（だが速い）方法でも面の前後関係が破綻しにくくなります。

図1のような立方体があるとします。8個の頂点に1から8の番号を付けます。すべての頂点座標を決めてから6個の面（図中のABCと裏側のDEF）を定義する必要があります。面を定義するには面を表から見て左回りに頂点を指定します。「表から見て左回り」では裏側の面や上下の面が分かりにくくなりますが、これにはコツがあります。

図2は右手を親指を立てて握り、面の外側に親指を向けた状態を示しています。このとき親指をどちらに向けても小指（4本の指とも同じですが）の指先に向かって左回りになります。図2では1-4-8-5（4-8-5-1でも同じ）と頂点の順番を決めることができます。実際に図1でも各面の頂点の順番を指定してみてください。

どうですか？A、B、Cの面の頂点順序は簡単に決めることができると思います。しかし裏側には見えない頂点番号6の頂点があります。頭で思い描いて裏側の面の頂点を指定するのはつらいものです。

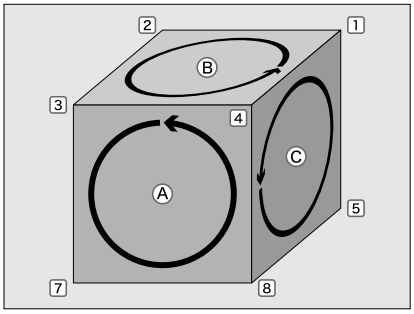
そこで図1の立方体を図3のようにデフォルメします。通常の遠近法とは逆に遠くを大きく描いています。このように立方体を描くとすべての頂点が見えます。上下左右の面C、E、D、Bに右手をあてて小指の根元から爪に向かう方向に頂点をたどることで正しく面を定義できます。例えば面Dは8-7-6-5（最初の頂点は8、7、6、5のどれでも良い）と決めることは簡単です。さて、見えない面Fも親指を紙面奥に向ければ2-1-5-6のように、2次元の図を使って頂点を指定することができます。

この方法は、立方体に限らず三角柱でも6角錐でも同じようにして面を定義できます。この方法をマスターするとJ3Wに付属するソースの頂点定義のコメントの意味が理解できると思います。私自身すべてのサンプルをこの方法で作成しています。3D CGを扱うアプリケーションのほとんどがマウスで簡単に物体を作成できるようになっていますが、複数の物体の相対的な大きさをキチンと指定するには、物体の設計図を紙に書いた方が近道であると思います。紙に設計図を描く場合に以上の手法を意識していれば、「3Dモデラーがなければ3Dは不可能」とは思いません。でも、あれば楽になるような気もチョットします(^^;。

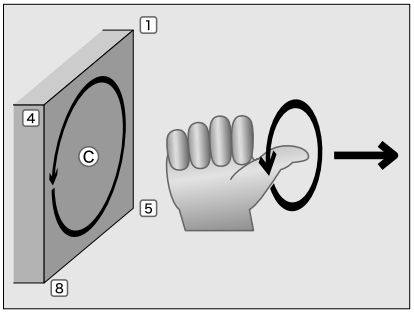
■頂点の指定

J3Wでは、物体は多角形（面）の集まりで構

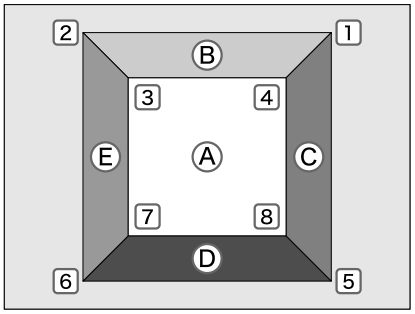
【図1】



【図2】



【図3】



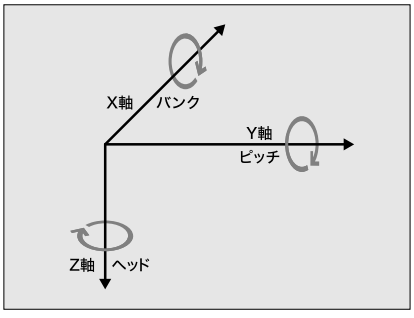
【リスト1】

```
DefPoint(100, 100, 200);
```

【リスト2】

```
RX=100; RY=100; RZ=200; Point();
```

【図4】



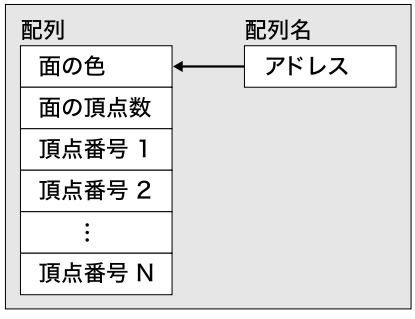
成されますが、多角形を定義するには頂点を指定しなければなりません。頂点は空間内の位置を使って1つ1つ指定します。頂点の位置は、地面に立って前がX、右がY、下がZという座標系を使って指定します。図1と図3では最も小さい頂点番号は1となっていますが、インスタンスで最初に定義される頂点番号は0となります。

頂点を定義するにはDefPoint()かPoint()組み込み関数を使います。リスト1の形で座標値を定数式で指定するか、リスト2のようにレジスタ変数経由で値を指定します。プログラム作成時に座標が決まっている場合にはDefPoint、実行時まで分からない(計算で頂点座標を求める)場合はPointのように使い分けます。

頂点の位置を指定するための座標には、図4の座標系を使用します。すべての物体はそれぞれ自分の向いている方向を基準として、前がX、右がY、下がZの増加する方向です。回転はZ軸回りの回転(右を向く)をヘッド回転、Y軸回りの回転(上を向く)をピッチ回転、X軸回りの回転(首を右に傾ける)をバンク回転と呼びます。

頂点座標はインスタンスごと(物体ごと)に1つ存在する局所座標系で指定します。その局所座標系をSetPosition関数を使って、ワールド座標系(北を向いて立ち、北がX、東がY、下がZ)に配置します。クラスごとに重力を設定できますが、重力は物体の姿勢に関係なくワールド座標系の下方向(Zが正の方向)に働きます。地上の建物のような運動しない物体の場合は、その局所座標系を、次の命令でワールド座標系の原点に配置することで、局所座標系とワールド座標系を一致させることも可能です(リスト3)。

【図5】



【リスト3】インスタンスの動作例

```
ClearRegisters();  // RX,RY,RZ,RH,RP,RB に0を代入
SetPosition();     // ワールド座標系の原点に配置
```

回転や移動は局所座標系の座標軸に沿って実行されるため、ワールド座標系とは別に、物体の回転の中心を局所座標系の原点として形状を定義すると良いでしょう。J3Wで使用している座標系は航空力学で使用されるNED系(X:North「前」、Y:East「右」、Z:Down「下」)で3次元グラフィックでよく使われる座標系とは異なりますが、地上の建造物のように広がりのある空間では高さ情報を最後に持つ座標系のほうが読みやすいと思います。座標値は符号付き32ビット整数の範囲(±20億)が可能です。

■面の定義

面(多角形)は、「面の色番号、頂点数、頂点の数の頂点番号」を指定することで定義します。すでに見てきたように面には表と裏があり、区別するために、頂点番号の指定には順序を考慮する必要があります。

面を定義する組み込み関数にはDefPlaneとPlaneがあります。DefPlaneの引数は、式に「物体」の面の色を指定し、続いて頂点数と頂点番号を定数で設定します。面を構成する多角形は必ず凸多角形でなければなりません。へこんだ形状の多角形は分割して、凸多角形で指定してください。頂点数個の頂点番号を渡すため、パラメータ数は可変となります(リスト4)。パラメータの頂点番号の数が、指定した「物体」の頂点数と異なる場合の動作は不定です。

DefPlane関数では、頂点番号を定数で指定しなければなりませんが、引数を配列で与えるPlane組み込み関数では、図5のように配列に面に関する情報を設定して、Plane(配列名)を実行します。色、頂点数、頂点番号を変数で実行時に指定することができるようになります(リスト5)。

【リスト4】

```
DefPlane( 式 , 頂点数 , 1, 2, 3, 4, 5 )
```

【リスト5】

```
Vertex[0] = color;       // 色
Vertex[1] = 3;           // 頂点数
Vertex[2] = j + i;       // 頂点0
Vertex[3] = j + i + 1;   // 頂点1
Vertex[4] = j + i + 2;   // 頂点2
Plane(Vertex);
```

頂点番号に負の値を使用すると親の頂点を指定することができます。階層構造の物体間に面を貼って接続できます。親の頂点番号の指定にはVertexRelative関数の影響はなく、オブジェクトごとに0から始まる頂点番号に単にマイナス符号を付けて指定します。親の頂点番号0は指定できません(負にならないため)。

■面の色

面の色は、0-31の番号で指定します。色番号が0-15の場合は、面と光源と視点の角度によって、面の明るさが変化(シェーディング)します。色の指定が16-31の場合は光源や視点の位置に関係なく、常に同じ明るさで表示され、発光する物体の表現や、地面のように視点の位置で明るさが変化すると不自然な場合に使います。表1に色番号と色の対応の初期値を示します。色番号と実際の色との対応はSetColor(色番号)という組み込み関数で変更できます。色番号に対応する色は、RX-RBのレジスタ変数で指定します(表2)。SetColorはグラフィックモードの時(GraphMode実行後)に実行します。

RX-RZを0に設定すると光源の影響を受けない色にできます。

【表1】色番号と色の対応の初期値

光源で変化する色番号		変化しない色番号	
0	濃灰	16	黒
1	青	17	青
2	緑	18	緑
3	水色	19	水色
4	赤	20	赤
5	うす紫	21	うす紫
6	黄	22	黄
7	白	23	白
8	濃灰	24	濃灰
9	濃青	25	濃青
10	濃緑	26	濃緑
11	うす緑	27	うす緑
12	茶	28	茶
13	ピンク	29	ピンク
14	オレンジ	30	オレンジ
15	クリーム	31	クリーム

■頂点番号の相対指定

少し複雑な物体では、頂点が多くなり、面を登録する時の頂点番号を間違えないようにすることが難しくなってきます。例えばフライトシミュレータのサンプルに登場するファルコンは180の頂点からなる1つの物体となっています。J3W付属のj3c_script/flight/falcon02.j3cで形状を定義していますが、垂直尾翼の定義部分を見てみましょう（リスト6）。

ファルコンの垂直尾翼の右側の面と左側の面を定義していますが、どちらの面も頂点番号は5までしか使っていません。これらの頂点が180の頂点の何番目か筆者も数えたことはありません。VertexRelative()関数を実行した直後に定義した頂点は、その後の面定義では0番目の頂点として指定できます。物体を構成する1つの部品の最初でVertexRelative()関数を実行すれば、その部品の最初の頂点番号を0として参照することができます。物体全体の頂点数を意識しないで部品ごとに頂点の追加や削除が可能となります。

■物体の大きさ

3Dアニメーションの主役は物体ですが、物体の大きさを決めないと「頂点座標を定義」することができません。J3Wでは空間の大きさは±20億の整数で表現できます。OpenGLやDirectXのように浮動小数点の数値を扱うことはできませんが、有効桁数はむしろ大きくなります。数値の単位は何でも構いません。例えば1cmを最小単位とすると±2万kmの範囲を表現できます。これなら、地球表面（約5億平方km）をカバーすることができてしまいます。

フライトシミュレータのように広い範囲を扱いたい場合には1cmを最小単位として考えれば良いと思います。人体モデルや家具、建物を考える場合には1mmが扱いやすい単位になると思います。単位はオングストロームでも光年でも自由に決めることができますが、それでは画面に表示される大きさはどうなるのでしょうか？

図6は視点から100の距離にある長さ200のパイプを見た場合を示しています。もし最小単位を1mと考えるならば、100mの距離にある長さ200mのパイプとなり、最小単位をナノメー

【表2】

光源から照明された場合の色	RX	拡散光 赤成分 (0 - 255)
	RY	拡散光 緑成分 (0 - 255)
	RZ	拡散光 青成分 (0 - 255)
光源から照明されない場合の色	RH	周辺光 赤成分 (0 - 255)
	RP	周辺光 緑成分 (0 - 255)
	RB	周辺光 青成分 (0 - 255)

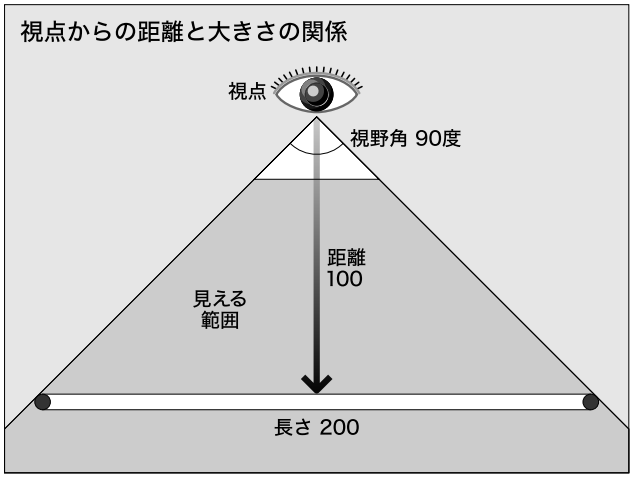
【リスト6】

```
    // 垂直尾翼
    VertexRelative();
    DefPoint( -10, -1, -50);  //0             4-- 3
    DefPoint(-300, -1, -50);  //1            /    /
    DefPoint(-300, -1, -90);  //2         5       2
    DefPoint(-390, -1,-240);  //3      0----------1
    DefPoint(-305, -1,-240);  //4
    DefPoint(-100, -1, -90);  //5
    DefPlane(color, 4, 0, 1, 2, 5);
    DefPlane(color, 4, 2, 3, 4, 5);
    VertexRelative();
    DefPoint( -10, 1,  -50);  //0              4-- 3
    DefPoint(-300, 1,  -50);  //1             /    /
    DefPoint(-300, 1,  -90);  //2          5       2
    DefPoint(-390, 1, -240);  //3       0----------1
    DefPoint(-305, 1, -240);  //4
    DefPoint(-100, 1,  -90);  //5
    DefPlane(color, 4, 5, 2, 1, 0);
    DefPlane(color, 4, 5, 4, 3, 2);
```

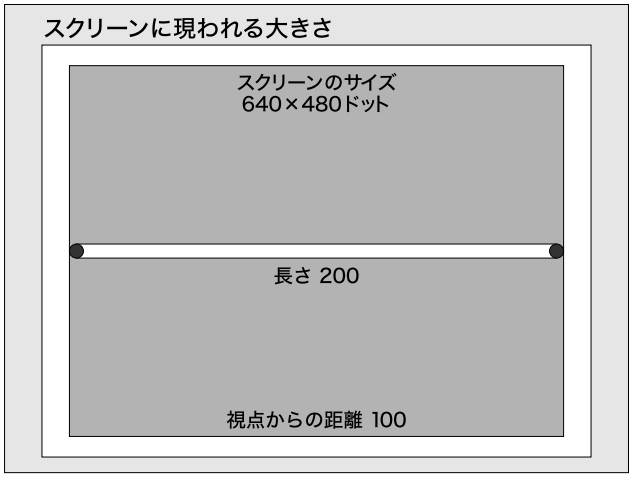
【リスト7】tetra.j3c

```
class Tetrahedron {
    int shape(int color) {
        NewObject(4, 4);        // オブジェクトの生成  4頂点 4面
        DefPoint(0, 0,  -283);  // 頂点0 三角形の頂点座標を登録
        DefPoint(200, 0,   0);  // 頂点1
        DefPoint(-100,-173,0);  // 頂点2
        DefPoint(-100, 173,0);  // 頂点3
        DefPlane(color,   3, 0, 1, 2); //面の定義 表から見て左回りに指定
        DefPlane(color+1, 3, 0, 3, 1);
        DefPlane(color+2, 3, 3, 0, 2);
        DefPlane(color+3, 3, 3, 2, 1);
    }

    int INIT() {
        shape(9);               // 形状定義
        ClearRegisters();       // RX - RBに0を代入
        SetPosition();          // 初期位置と角度の設定
    }

    int RUN() {
        RotHead(6000, 2880);    // 6秒間で360度回転
        RotPitch(6000, 2880);   // 6秒間で360度回転
        DeleteObject();         // 形状を削除
        Stop();                 // 全インスタンス終了
    }

    int EVENT() {}
}

class main {
    int INIT() {
        new(Tetrahedron, 50);   // 正4面体のインスタンスの生成
        NewObject(0, 0);        // 視点用の物体の生成
        ClearRegisters();       // RX - RBに0を代入
        RX = -1000;             // 位置の指定 10m後方
        SetPosition();          // 位置と角度を視点に設定
        See();                  // このオブジェクトが見る
        BackgroundColor(0x422); // 背景色を設定
        GraphMode();            // グラフィックウインドウを開く
        Zoom(1);                 // 視野角53度(2倍ズーム)
    }

    int RUN() {
        Wait();                 // 何もしない
    }

    int EVENT() {}
}
```

【図6】



【図7】



トルと考えれば（本物の）CPU内部の配線の幅として扱うことができます。また、組み込み関数のZoomをZoom(1)と指定することで、デフォルトの視野角90度を約53度に変更できます。Zoom(1)の状態では2倍に拡大（ズーム）され、例えば距離1000の大きさ1000の物体が画面いっぱいに表示されます。

図7は図6の状態が表示された画面を示します。つまり物体の大きさは視点からの距離だけで決まるため、単位の決め方には依存しません。カメラで人物を写す場合と山を撮る場合を考えれば簡単です。太陽も離れているから小さく見えます。ただし、視点に近すぎる物体（距離50以内）と、遠すぎる場合（距離4000000）はクリッピングによって切り取られて表示されません。

■物体の表示

J3Wの特徴の1つは、3次元空間に存在するすべての物体が「視点になることができる」ことです。視点になっている物体から見た3次元空間がスクリーンに表示されます。従って、少なくとも3次元空間に「物体」と「物体を見ている物体」が存在する必要があります。

1つのインスタンスは1つの物体になることができます。これにより最低でも、2つのインスタンスが必要になります。また視点と物体で異なる動作をする場合には、クラスも2つ必要になります。

リスト7では、自動的に生成されるmainクラスのインスタンスが空の（形のない）物体を生成し、位置をSetPosition組み込み関数で設定してSee組み込み関数で視点を取得します。このインスタンスが目となり、2番目のインスタンス（Tetrahedron）は正4面体の物体となります。

クラスmainは自動的に実行されるINITメソッドの冒頭でクラスTetrahedronのインスタンスをnew()組み込み関数を使って生成しています。これでクラスTetrahedronのインスタンスは実行を始めます。2つのインスタンスが同時に動き始めますが、文章では視点となるmainクラス（のインスタンス）の動作から説明します。

視点には頂点も面も必要ないので頂点と面の数を0として見えない物体を生成しています。ただし見えなくても、SetPositionで設定された位置と角度の情報を持ちます。ClearRegisters()はRX、RY、RZ、RH、RP、RBをすべて0にする関数です。その後、RXに-1000を代入して、ワールド座標(-1000, 0, 0)に視点を配置しています。SetPositionには位置座標(RX,RY,RZ)と角度(RH，RP，RB)をレジスタ変数に設定して呼び出します。すると、See()を実行したインスタンスから見た光景がウィンドウに表示されます。実際に表示するためにはGraphMode組み込み関数を実行してグラフィックウィンドウを開く必要があります。この組み込み関数はどのインスタンスが実行しても構いませんが、複数のウィンドウを開くことはできません。この例ではZoom(1)を実行して、視野角を約53度に変更しています。物体が2倍にズームされ、周辺部の歪みが小さくなります。INITメソッドの仕事は以上です。この例では視点は移動しないためRUNメソッドでは終了イベントを待っているだけです。

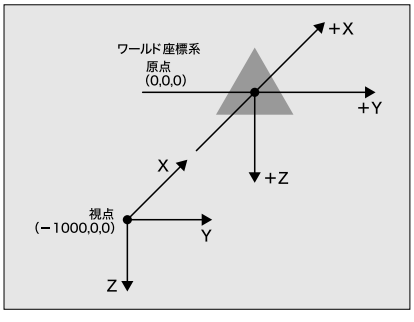
さて、クラスTetrahedronのインスタンスは、INITメソッドがshapeメソッドを呼び出して、NewObject()組み込み関数を使って形状を生成するための準備をします。NewObject()の引数には最大頂点数と最大面数を与えます。頂点数、面数は実際に使用する数より多くても問題ありませんが、少ないとエラーが発生します。形状の定義はDefPoint組み込み関数でX、Y、Z座標を指定して1つずつ頂点を登録します。頂点番号はオブジェクトごとに0から登録順に付けられます。DefPlane組み込み関数で面の色、頂点数、頂点番号列を指定することで面を定義します。

視点と物体の位置は図8の関係にあります。正4面体の中心はワールド座標系の原点にあります。J3Wでは視点の位置や角度が自由に変更できるため、物体を「見失ってしまう」ことがあります。慣れるまでは、図8のように、物体をワールド座標系の原点に配置し、視点は物体の大きさ程度の距離を後ろにさがり（Xが負の位置）、角度はすべて0にしておくと、「見失って何も表示されない」状況を防ぐことができます。

4面体の運動はRotHead(6000，2880)とRotPitch(6000，2880)で指定しています。この場合6秒間で360度ヘッド回転（Z軸回り）した後、6秒間で360度ピッチ回転（Y軸回り）して終了します。移動や回転に使う組み込み関数で指定する時間は、ミリ秒を単位とします。角度は1/8度を単位として、例えば30度は240となります。移動距離は頂点座標で使う数値と同じ単位を使います。リスト7の回転速度を変更して実際に試してください。さらにUp、Forwardなどの組み込み関数を順次実行することでいろいろな動きを与えることができます。

また、この記事で説明した方法を使って紙に正4面体を描いて見てください。DefPointとDefPlaneで頂点と面の登録の部分のコードを

【図8】



実際に書いて、リスト7と比較してみることをお勧めします。一見すると面倒で読み飛ばしたくなる部分ですが、形を変えたり頂点を追加したりすれば、すぐに慣れてくると思います。

■終わりに

物体の形状定義という3D CGを作成する上で最も重要な部分を解説しました。次回は、少しずつ異なる多くの物体をコードの再利用によって楽にプログラミングできるように、J3C言語の継承の機能の解説を予定しています。

Column J3W for Linux最新版ソースコードについて

J3W for Linuxの最新版ソースコード「j3w643.tar.gz」を、本誌付録CD-ROMに収録しています。インストールを行うには、圧縮ファイルを展開後、j3w-643ディレクトリに移動します。その後「make && make strip」と打ち込んでコンパイルを行います。次にrootユーザーになって「make install && make clean」とすると、作業は終了します。

コマンドの検索パスに/usr/local/binが含まれているかどうかの確認も忘れず行ってください。もし含まれていない場合は、.bashrcファイルに「export PATH=/usr/local/bin:$PATH」のようにパスを追加してください。

インストールについてさらに詳しい情報を知りたい場合には、作業を行う前にJ3W添付のREADME.eucファイルを確認してください。J3Wに関する詳細な情報も同梱のドキュメントに含まれています。

（編集部）

Column J3Wの内部構造の解説

前回は、物体の形状に関する情報はTPolygonsクラス、移動や回転はTAxisクラスが担当することを解説しました。TPolygons、TAxisクラスをまとめて、頂点や面を1つの物体として扱うためにTObject3Dクラスがあります。

■物体の階層構造

J3Wでは階層構造（多関節の物体）を扱うために、TObject3Dを継承して拡張したTHObj3Dクラス（図A）が3Dエンジンのまとめ役となっています。THObj3Dクラスが物体の形、位置、運動を管理し、座標変換の計算も担当する「ジオメトリックエンジン」となっています。

例えば人体は多くの関節でできており、ある関節を曲げるとその先の関節の位置が変わります（腕の関節を曲げると手首の関節の位置が変わりますね）。物体を関節ごとに階層に分けて管理する必要があります。図Bでは非常に簡略化していますが、階層構造の関係の例を示しています。

この図Bでは、人体を構成する個々のパーツが親子関係となっており、上のレベルほど人体の姿勢に大きな影響を与えます。例えば2段目の胸のパーツを右にひねることを考えると、頭も腕も胸の動きに伴って右を向きます。親が動くと子孫すべての位置や角度が影響を受けるため、頂点の座標は親から順に決定する必要があります。THObj3Dクラスのすべてのインスタンスは親物体へのポインタをもっていますが、親がない（NULL）場合は、その物体はワールド座標系に独立して存在します。親を持つインスタンスは親の局所座標系に配置されることで、親の位置や姿勢が子供に反映されるようになります。

■空間

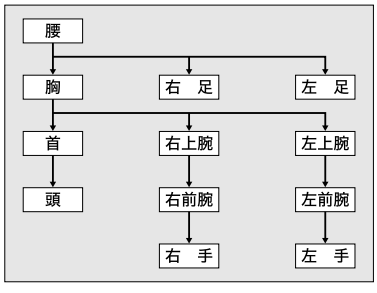
物体を実際にスクリーンに表示する部分は、TSpaceH3Dクラスが「レンダリングエンジン」として担当します。TSpaceH3Dクラスは、3次元空間全体を管理するために次のような機能を持っています（図C）。

1. 3次元空間に対して物体の登録や削除をする。
2. 視点を持つ物体を指定する。
3. 視点座標系で表されているポリゴン（多角形）をスクリーンの表示範囲に合わせて切り取る（クリッピング）。
4. 3次元のポリゴンをスクリーンに合わせて2次元に変換する。
5. 光源の位置と種類に合わせてポリゴンの色を決定する。
6. 視点から遠いポリゴンから描画する。

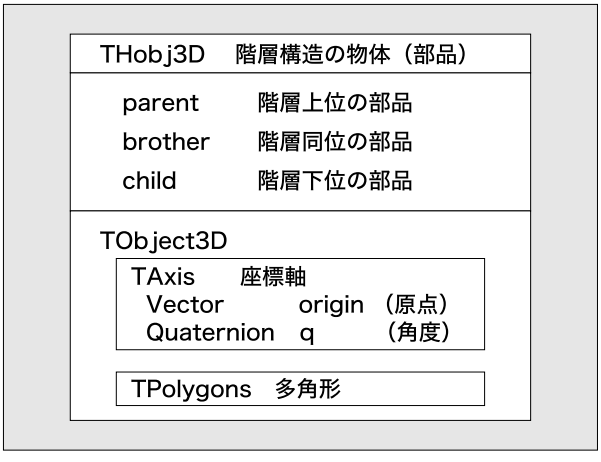
物体はすべて3次元空間内に存在しています。その3次元空間を扱うのがTSpaceh3Dクラスです。実際にスクリーンに描画するクラスはTScreenの派生クラスであるScrnXです。ScrnXはXlibを使って直線や多角形をX Window Systemのウィンドウに描画します。

次回は、これまで解説したクラスを実際に使った、小さな3dプログラムを紹介する予定です。（水谷純）

【図B】階層構造の関係の例



【図A】THObj3Dクラス



【図C】TSpaceH3Dクラス