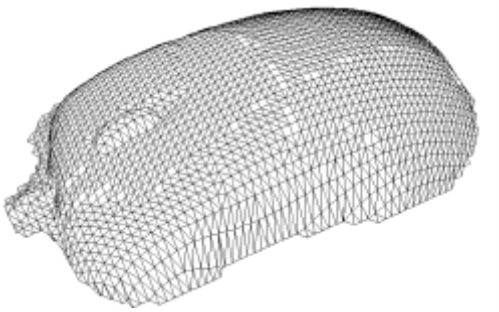
ESライブラリ＆＆
ゲームプログラミング

○　３Ｄ編 - 第１回　シンプルシェイプ　○

シンプルシェイプ

・Direct3DXには、簡単な形状のモデルを生成する機能がある
・ダミーモデルとして使用したり、モデルに重ねて衝突判定のテストに使用

メッシュ



3Dグラフィクスでは、複数の三角形をつなげていくと、網のようになります。Direct3Dでは、三角形ポリゴン(面)の集合体のことをメッシュ(Mesh)と呼びます。モデルは面の集合体なので、メッシュで形成されているといえます。
メッシュには親子や兄弟といった関係を持たせ、他のメッシュに関連づけられているものもあるので、そのままでは複雑なプログラムになってしまいます。
そこで、Direct3D拡張ライブラリのDirect3DXでは、複雑な関係を持つメッシュも簡潔に扱えるように、ID3DXMeshインタフェースを提供しています。

ESライブラリでは、さらに使いやすくするために、CModelクラスを定義しています。このクラスにより、メッシュの移動、拡大縮小、回転、マテリアルの設定が簡単に行えるようになっています。(ソースコードは"Model.hpp", "Model.cpp")

シンプルシェイプ

Direct3DXが提供する関数には、以下のように簡単な形状(シンプルシェイプ)のメッシュモデルを生成するものがあります。

D3DXCreateBox関数	立方体を生成
D3DXCreateCylinder関数	円柱を生成
D3DXCreatePolygon関数	多角形を生成
D3DXCreateSphere関数	球を生成
D3DXCreateTorus関数	トーラスを生成
D3DXCreateTeapot関数	ティーポットを生成
D3DXCreateText関数	テキストを生成

これらの関数を使うと、指定した形状のメッシュオブジェクトが生成され、それを制御するためのインタフェース(ID3DXMesh)が返されます。

ESライブラリでも、シンプルシェイプを扱えるように、CDXGraphics9クラスにCreateModelFrom SimpleShape関数を実装しています。この関数は、シンプルシェイプを生成し、管理用のCModel型変数を返すようになっています。

シーンの開始と終了を宣言

Direct3Dでは、3Dの機能を用いた描画(レンダリング)を行う際、描画の前にIDirect3DDevice9::BeginSceneメソッドを呼び出してシーンの開始を宣言し、システムにレンダリングの開始を通知する必要があります。また、3Dの描画がすべて終了した場合は、IDirect3DDevice9::EndSceneメソッドを呼び出してシーンの終了を宣言し、システムにレンダリングの終了を通知しなければならないというルールになっています。

ESライブラリでも、CDXGraphics9クラスにBeginScene関数とEndScene関数を用意しています。3D描画の前後に、これらの関数を呼び出してください。

ESライブラリでのコーディング例

----- ヘッダー[GameMain.h] -----

```
        ⋮
private:
    // 変数宣言
    IModel*    m_pShape;

    // 関数プロトタイプ
};
```

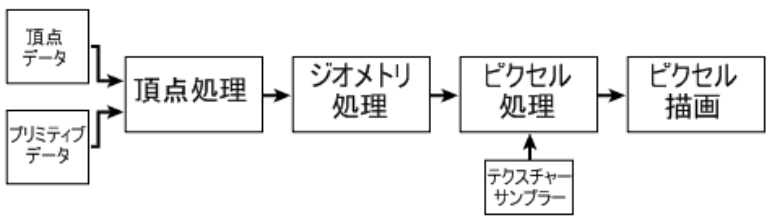
----- ソース[GameMain.cpp] -----

```
        ⋮
void CGameMain::LoadContent()
{
    // TODO: use this.Content to load your game content here
    // ティーポット生成
    SimpleShape   shape;
    shape.Type = Shape_Teapot;
    m_pShape   = GraphicsDevice.CreateModelFromSimpleShape(shape);
}

        ⋮

void CGameMain::UnloadContent()
{
    // TODO: Unload any non ContentManager content here
    GraphicsDevice.ReleaseModel(m_pShape);  // モデル解放
}

        ⋮

void CGameMain::Draw()
{
    GraphicsDevice.Clear(Color_CornflowerBlue);

    // TODO: Add your drawing code here
    GraphicsDevice.BeginScene();     // シーン開始

    m_pShape->Draw();                // モデル描画

    GraphicsDevice.EndScene();       // シーン終了
}
```

頂点データのレンダリング

頂点データは、物体や図形に関する情報を数値にして配列に並べているだけです。これを計算によって画像にする必要があります。3Dグラフィクスでは、視点、光源の種類・位置・数、モデルの形状(ポリゴンの向き)を考慮して面の消去や陰影付けなどを行って画像を計算します。

この一連の流れをレンダリングと呼びます。DirectX9(DirectX Graphics9)では、頂点データを以下の流れで画像にしています。

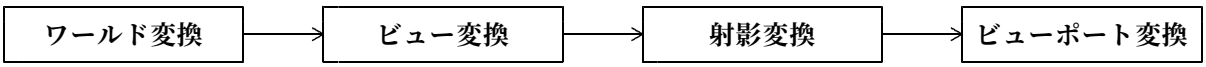


頂点データは、まず頂点処理が行われます。頂点処理では、ワールド行列、ビュー行列、射影行列といった座標変換行列により、視点から見た座標に変換されます。次にジオメトリ処理により、クリッピング(画面外ポリゴンの削除)、背面カリング(背面ポリゴンの削除)、ラスター化(ポリゴンにピクセルを割り当てる処理)が行われます。
ラスター化されたピクセルは、色がついていません。ピクセル処理では、頂点カラー、テクスチャ、ライティングといったカラー処理が施され、ポリゴンに割り当てられるピクセルの色が計算されます。最後にピクセル描画が行われ、透明度やｚバッファを適用し、最終的なピクセルカラー値を決定します。

頂点処理とジオメトリ処理の流れ

頂点処理とジオメトリ処理とは、モデルの頂点データに対して、座標変換および照明演算を行い、実際に描画できるデータに変換を行う処理のことです。DirectX Graphics9では、T&L固定機能とバーテックスシェーダがあり、どちらかを選んで使います(頂点データごとに切り換えることもできます)。

T&L固定機能では、頂点データをジオメトリパイプラインで処理します。ジオメトリパイプラインとは、一連の座標変換処理のことです。3Dオブジェクトは、ワールド、ビュー、射影、ビューポートの４つの座標変換を経てピクセルに変換されます。各座標変換には変換行列と呼ばれる行列を用います。



▲T&L固定機能におけるジオメトリパイプライン

固定機能の動作は、ステート(状態)の設定によって制御します。DirectX Graphics9は、ステートマシンとして設計されています。変換行列、マテリアル、テクスチャ、レンダリングステートといった状態は、一度設定すると変更されない限り維持され続けます(ただし、IDirect3DDevice9::Resetメソッドを呼び出した場合は消去されます)。

これに対してバーテックスシェーダとは、必要な変換処理をプログラムで記述し、ビデオカードに直接命令を送る方法です。頂点をどのように座標変換するのかをプログラムで自由に設計することができます。GPUに直接命令を出すので、高速・高度にハードウェアの性能を引き出すことができます。PCの構成に近いゲーム機以外でも、たとえばNINTENDO64やPS3も同じ考え方が導入されています。

課　　題

シンプルシェイプの「ティーポット」を画面に表示しましょう。

(１)モデルを管理するための変数を宣言します。

GameMain全体で使うような変数は、ヘッダーファイル"GameMain.h"で宣言します。C++なのでクラス内で宣言します。以下のプログラムを"private:"の下に追加しましょう。

```
MODEL    m_pShape;
```

MODEL型は、ESライブラリでモデルを管理するための基本インターフェースです。"Model.hpp"で基本型を、さらに"XNA.h"でtypedef定義しています。

(２)ティーポットを生成するプログラムを追加します。

グラフィックやサウンドの読み込み・生成は、ESライブラリでは、"LoadContent関数"に記述します。以下のプログラムを適切な場所に追加しましょう。

```
// ティーポット生成
SimpleShape    shape;
shape.Type = Shape_Teapot;
m_pShape   = GraphicsDevice.CreateModelFromSimpleShape(shape);
```

GraphicsDevice.CreateModelFromSimpleShape関数は、渡されたSimpleShape構造体の情報をもとに、シンプルシェイプを生成する関数です。関数は"DXGraphics9.cpp"、構造体は"XNA.h"→"DXGType.h"で定義しています。

(３)ティーポットを描画するプログラムを追加します。以下のプログラムの'?'を埋め、適切な場所に追加しましょう。

```
GraphicsDevice.??????????();     // シーン開始

m_pShape->Draw();                // シンプルシェイプ描画

GraphicsDevice.????????();       // シーン終了
```

ESライブラリでは、CModelクラスのDraw関数を呼び出すだけで描画できるようになっています。もとになっているDirect3DXのID3DXMeshインターフェースでも、DrawSubsetメソッドというメッシュ単体を描画する機能があります。CModelクラスのDraw関数では、DrawSubsetをモデルに含まれているすべてのメッシュ分、繰り返し呼び出すという動作をしています。

※画面に黒い帯状のものが描画されれば成功です。