オブジェクト指向と ゲームプログラミング

▶ DirectX Graphics 3D編 － 第3回　シンプルシェイプ ◀

メッシュ

三角形を複数定義すると、網のようになります。DirectX Graphics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。3Dオブジェクトは、面の集合体なので、このメッシュで形成されているといえます。

メッシュといっても、頂点の集まりなので、DirectX Graphicsでは頂点バッファで管理します。しかし、メッシュには親子や兄弟といった関係を持つ複数のオブジェクトからなるものもあるので、頂点バッファだけでは複雑なプログラムになってしまう場合があります。

このようなメッシュを簡潔に扱うのためのインタフェースがDirect3DXが提供するID3DXMeshインタフェースです。

Direct3D エクステンション ユーティリティ ライブラリ

「Direct3D エクステンション(Direct3DX) ユーティリティ ライブラリ」とは、DirectX Graphicsのプログラミング作業を簡単にする関数群およびインタフェースのことです。

ベクトルと行列の算術演算、テクスチャ、メッシュ、シェーダといった処理の簡易化を行うためのヘルパー関数と、メッシュ、アニメーションメッシュ、スキンメッシュ、シンプルシェイプ、スプライト、フォント、エフェクトなど多数のインタフェースを提供しています。

これらの関数とインタフェースを用いることにより、複雑になりがちな部分を簡潔にすることができます。

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)のメッシュオブジェクト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。

```
// 立方体のメッシュオブジェクトの生成(pD3DDeviceは、初期化済みのデバイスオブジェクト)
ID3DXMesh*   pMesh;
D3DXCreateBox(pD3DDevice, 1.0f, 1.0f, 1.0f, &pMesh, NULL);
```

メッシュオブジェクトは、ID3DXMesh::DrawSubsetメソッドを呼び出すだけでレンダリングすることができます。

```
// メッシュ描画
pMesh->DrawSubset(0);
```

メッシュオブジェクトが不用になった場合は、ほかのオブジェクト同じようにReleaseメソッドを呼び出せば、使用していた頂点バッファなどの資源も含めて解放されます。

```
// メッシュ解放
pMesh->Release();
```

課　　題

シンプルシェイプを描画してみましょう。

(１)メッシュをカプセル化するクラスを作成します。

メッシュは、ID3DXMeshインタフェースをとおして操作を行います。このインタフェースは、メッシュオブジェクトの生成に失敗した場合、NULLを保持します。ほとんどの場合、メッシュオブジェクトが無いということは、3Dオブジェクトの処理が行えないということになり、異常発生ということでプログラムを終了させます。しかし、デバッグ時など、場合によっては実行を続けたいこともあります。

そこで、メッシュクラスにNull Objectパターンを適用し、NULLチェックを省けるようにします。ます。Null Objectパターンを適用したメッシュクラスのヘッダファイル(Mesh.hpp)を以下のように作成しましょう。

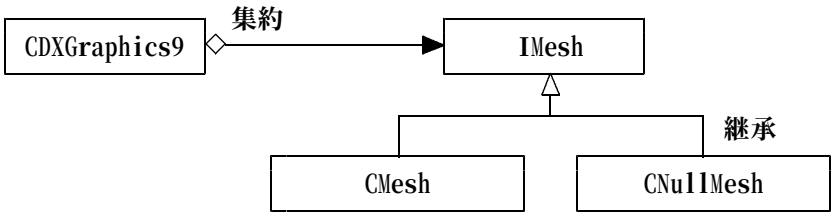
－Mesh.hpp－

```
/*
===========================================================================================
                                オブジェクト指向ゲームプログラミング
    Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
===========================================================================================

  【対象ＯＳ】
    Microsoft Windows2000/XP

  【コンパイラ】
    Microsoft Visual C++ 2005

  【プログラム】
    Mesh.hpp
                メッシュクラスヘッダ

  【履歴】
    * Version    1.00       2005/03/dd  hh:mm:ss

===========================================================================================
*/

#pragma once

/*****************************************************************************************/
/*                                    インクルードファイル                                 */
/*****************************************************************************************/
#include <d3dx9.h>

/*****************************************************************************************/
/*                                  メッシュインタフェース定義                             */
/*****************************************************************************************/
class IMesh {
public:
    virtual ~IMesh() {}
};

/*****************************************************************************************/
/*                                    メッシュクラス定義                                   */
/*****************************************************************************************/
class CMesh : public IMesh {
public:

private:
};

/*****************************************************************************************/
/*                                  NULLメッシュクラス定義                                 */
/*****************************************************************************************/
class CNullMesh : public IMesh {
public:
};
```

IMeshクラスは、CMeshクラスとCNullMeshクラスのインタフェースとして使用します。

CMeshクラスがメッシュオブジェクトをカプセル化するクラスになります。このクラスに、メッシュを制御するために必要な機能と属性を追加していきます。
CNullMeshクラスがNull Object役になります。今後、CNullMeshクラスには、IMeshクラスのpublicな関数をすべて「何もしない処理」として実装します。

最終的なクラス相関図は、以下のようになります。



(２)メッシュクラスのソースファイル(Mesh.cpp)を以下のように作成しましょう。

－Mesh.cpp－

```
/*
================================================================================================
                                オブジェクト指向ゲームプログラミング
     Programmed by Hibikino software. Copyright (c) 2005 Hibikino software. All rights reserved.
================================================================================================

  【対象ＯＳ】
     Microsoft Windows2000/XP

  【コンパイラ】
     Microsoft Visual C++ 2005

  【プログラム】
     Mesh.cpp
                メッシュクラス

  【履歴】
     * Version    1.00       2005/03/dd  hh:mm:ss

================================================================================================
*/

/*********************************************************************************************/
/*                                        インクルードファイル                                 */
/*********************************************************************************************/
#include "Mesh.hpp"
#include <cassert>
```

(３)CMeshクラスに、メッシュを管理するのに必要なメンバを追加します。以下のプログラムを適切な場所に追加しましょう。

```
    IDirect3DDevice9*   m_pD3DDevice;
    ID3DXMesh*          m_pMesh;

    const DWORD         m_SubsetCount;      // サブセット数
```

メッシュを管理するだけなら、ID3DXMesh*型の変数だけメンバ変数にすれば良さそうですが、メッシュの変換変換行列、マテリアル、テクスチャを設定する際に、IDirect3DDevice9*型の変数が必要になります。また、レンダリングの際にサブセット数が必要になるので、これらもメンバ変数に加えておきます。

サブセット数とは、オブジェクトの部品数のことです。１つのオブジェクトをいくつかの部品に分割する場合があります。たとえば、人間オブジェクトを作成した場合、人間全体を１つのオブジェクトにするのではなく、頭、胴体、腕、足などいくつかの部品に分割してその集合体として作成します。
サブセット数には、この部品数を格納しておきます。マテリアルやテクスチャの設定、レンダリング時は部品ごとに行わなければならないので、格納しておいたサブセット数だけループで繰り返して処理することになります。

(４)CMeshクラスのコンストラクタを作成します。以下のプログラムを完成させてソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          コンストラクタ                                           */
/*****************************************************************************************************/
CMesh::CMesh(IDirect3DDevice9* pD3DDevice, ID3DXMesh* pMesh, const DWORD inSubsetCount)
    : m_pD3DDevice(pD3DDevice), m_pMesh(pMesh), m_SubsetCount(inSubsetCount)
{
    assert(m_pD3DDevice != NULL && m_pMesh != NULL);

    m_pD3DDevice->??????();     // 参照カウンタを増やしておく
    m_pMesh     ->??????();
}
```

コンストラクタの仕様は、以下のとおりです。

CMesh　　コンストラクタ

CMeshオブジェクトを構築します。

試 `CMesh(IDirect3DDevice9* pD3DDevice, ID3DXMesh* pMesh, const DWORD inSubsetCount);`

pD3DDevice	Direct3DDevice9オブジェクトへのインタフェース
pMesh	D3DXMeshオブジェクトへのインタフェース
inSubsetCount	メッシュに格納されているサブセットの数

(５)CMeshクラスのデストラクタを作成します。以下のプログラムを完成させてソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CMesh::~CMesh()
{
    ※ここは各自考えましょう(m_pMeshを解放します)
    ※ここは各自考えましょう(m_pD3DDeviceを解放します)
}
```

(６)CMeshクラスのコンストラクタおよびデストラクタのプロトタイプをクラス定義の適切な場所に追加しましょう。

(７)CNullMeshクラスに「何もしない」デストラクタを作成しましょう。

(８)メッシュオブジェクトの解放漏れを防ぐため、すべてのメッシュオブジェクトをCDXGraphics9クラスが管理するようにします。
以下のメンバをCDXGraphics9クラスに追加しましょう。

```
    std::list<IMesh*>    m_Mesh;          // メッシュリスト
```

(９)CDXGraphics9クラスに、シンプルシェイプメッシュを生成する関数群を追加します。以下の仕様を読み、プログラムを完成させて適切な場所に追加しましょう。なお、プロトタイプの追加も忘れずに行いましょう。

CreateBox　　生成

立方体のメッシュを作成します。

試 `IMesh* CreateBox(const float inWidth, const float inHeight, const float inDepth);`

Return	生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inWidth	立方体の幅
inHeight	立方体の高さ
inDepth	立方体の奥行き

CreateCylinder　　生成

円柱(円錐)のメッシュを作成します。

試IMesh* CreateCylinder(const float inRadius1, const float inRadius2,
const float inLength, const UINT inSlices, const UINT inStacks);

Return 生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inRadius1 先頭の半径
inRadius2 後尾の半径
inLength 長さ
inSlices 横断面の辺の数
inStacks 横断面の数

CreatePolygon 生成

多角形のメッシュを作成します。

試IMesh* CreatePolygon(const float inLength, const UINT inSides);

Return 生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inLength 各面の長さ
inSides 面の数。三角形以上でなければならないので、３以上でなければなりません

CreateSphere 生成

球のメッシュを作成します。

試IMesh* CreateSphere(const float inRadius, const UINT inSlices, const UINT inStacks);

Return 生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inRadius 球の半径
inSlices 横断面の辺の数
inStacks 横断面の数

CreateTorus 生成

トーラス(ドーナッツ)のメッシュを作成します。

試IMesh* CreateTorus(const float inInnerRadius, const float inOuterRadius,
const UINT inSides, const UINT inRings);

Return 生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります
inInnerRadius 内側の半径
inOuterRadius 外側の半径
inSides 横断面の辺の数。３以上でなければなりません
inRings トーラスを構成する環の数。３以上でなければなりません

CreateTeapot 生成

ティーポットのメッシュを作成します。

試IMesh* CreateTeapot();

Return 生成されたメッシュオブジェクトへのインタフェース。生成に失敗した場合、NULLオブジェクトへのインタフェースが返ります

```
/*****************************************************************************************************/
/*                                          立方体生成                                               */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreateBox(const float inWidth, const float inHeight, const float inDepth)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*      pShape;
    ID3DXMesh*  pMesh = NULL;
    if(::?????????????(m_pD3DDevice, inWidth, inHeight, inDepth, &pMesh, NULL) == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - 立方体生成失敗(CDXGraphics9_CreateBox)\n");
        pShape = new ??????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加
```

```
    return pShape;
}

/*****************************************************************************************************/
/*                                              円柱生成                                             */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreateCylinder(const float inRadius1, const float inRadius2, const float inLength, co
nst UINT inSlices, const UINT inStacks)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*       pShape;
    ID3DXMesh*   pMesh = NULL;
    if(::※ここは各自考えましょう == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - 円柱生成失敗(CDXGraphics9_CreateCylinder)\n");
        pShape = new ?????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}

/*****************************************************************************************************/
/*                                              多角形生成                                           */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreatePolygon(const float inLength, const UINT inSides)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*       pShape;
    ID3DXMesh*   pMesh = NULL;
    if(::※ここは各自考えましょう == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - 多角形生成失敗(CDXGraphics9_CreatePolygon)\n");
        pShape = new ?????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}

/*****************************************************************************************************/
/*                                              球生成                                               */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreateSphere(const float inRadius, const UINT inSlices, const UINT inStacks)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*       pShape;
    ID3DXMesh*   pMesh = NULL;
    if(::※ここは各自考えましょう == D3D_OK) {
```

```
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - 球生成失敗(CDXGraphics9_CreateSphere)\n");
        pShape = new ?????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}

/*****************************************************************************************************/
/*                                          トーラス生成                                             */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreateTorus(const float inInnerRadius, const float inOuterRadius, const UINT inSides,
 const UINT inRings)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*       pShape;
    ID3DXMesh*   pMesh = NULL;
    if(::※ここは各自考えましょう == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - トーラス生成失敗(CDXGraphics9_CreateTorus)\n");
        pShape = new ?????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}

/*****************************************************************************************************/
/*                                          ティーポット生成                                         */
/*****************************************************************************************************/
IMesh* CDXGraphics9::CreateTeapot()
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    IMesh*       pShape;
    ID3DXMesh*   pMesh = NULL;
    if(::※ここは各自考えましょう == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - ティーポット生成失敗(CDXGraphics9_CreateTeapot)\n");
        pShape = new ?????????();   // 生成に失敗した場合は、NULLオブジェクトを生成
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}
```

(10)CDXGraphics9クラスに、メッシュオブジェクトを解放するReleaseMesh関数を追加します。この関数の引数は、解放するメッシュオブジェクトのインタフェースです。
　次のプログラムを完成させて適切な場所に追加しましょう。なお、プロトタイプの追加も忘れずに行いましょう。

```
/*****************************************************************************************************/
/*                                          メッシュ解放                                             */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseMesh(CMesh*& pMesh)
{
    m_Mesh.??????(pMesh);    // メッシュリストから削除
    delete pMesh;            // メッシュオブジェクト解放
    pMesh = NULL;
}
```

(11)以下のプログラムは、CDXGraphics9クラスが保持しているすべてのメッシュオブジェクトを解放するReleaseAllMeshes関数です。足りない部分を補い、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          全テクスチャ解放                                         */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseAllMeshes()
{
    for(std::list<IMesh*>::iterator it = m_Mesh.?????(); it != m_Mesh.???(); it++)
        delete *it;
    m_Mesh.clear();
}
```

(12)CDXGraphics9クラスの解放時に、すべての頂点バッファが解放されるようします。以下のプログラムを適切な場所に追加しましょう。

```
    // メッシュ解放
    ReleaseAllMeshes();
```

(13)メッシュをレンダリングするRender関数を作成します。以下のプロトタイプをIMeshクラス定義の適切な場所に追加しましょう。

```
    virtual void Render() = 0;
```

(14)CMeshクラスにRender関数を追加します。Render関数のプロトタイプを(13)をもとに作成し、適切な場所に追加しましょう。

(15)Render関数を実装します。以下のプログラムを完成させ、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          レンダリング                                             */
/*****************************************************************************************************/
void CMesh::Render()
{
    for(DWORD i = 0; i < m_SubsetCount; i++) {
        m_pMesh->??????????(i);
    }
}
```

(16)CNullMeshクラスに「何もしない」Render関数を追加しましょう。

(17)プリミティブを描画してみましょう。CTestSceneクラスに、以下のメンバを追加します。

```
    IMesh*    m_pMesh;
```

(18)頂点バッファ生成し、頂点データを設定します。CTestSceneクラスのコンストラクタを以下のように変更しましょう。

```
CTestScene::CTestScene()
{
    // ここに、機能テストの初期化処理を記述します
    // ティーポット生成
    m_pMesh = DXGraphics().CreateTeapot();
```

```
    FPSTimer().Reset();      // タイマリセット
}
```

(19)CTestSceneクラスのデストラクタを以下のように変更し、テストシーン終了時に生成したすべてのメッシュが解放されるようにしましょう。

```
CTestScene::~CTestScene()
{
    // ここに、機能テストの解放処理を記述します
    DXGraphics().ReleaseAllMeshes();
}
```

(20)CTestScene::Active関数を以下のように変更し、メッシュを描画してみましょう。

```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します

    // ここに、機能テストの描画処理を記述します
    if(FPSTimer().IsSkip())
        return 0;

    DXGraphics().Clear(D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DXCOLOR(0.0f, 0.0f, 0.5f, 0.0f));

    DXGraphics().BeginScene();

    m_pMesh->Render();              // メッシュのレンダリング

    DXGraphics().EndScene();

    DXGraphics().UpdateFrame();
}
```

※各変換行列およびマテリアルを設定していないため、正しく描画されていないように見えます。

発展問題　シンプルシェイプ生成関数をまとめて１つの関数にしましょう。

立方体や球などのシンプルシェイプを生成する関数は、メッシュオブジェクトを生成する部分(D3DXCreateBox関数やD3DXCreateShape関数を呼び出している部分)以外は、まったく同じ処理を行っています。この部分をまとめることができれば、６つの関数(CreateBox関数からCreateTeapot関数)をまとめて１つの関数にすることができ、コードが大幅に削減され、形状の追加も容易になります。

以下の手順に従ってこれらの関数を１つにまとめてみましょう。

(１)形状に定数を割り当てます。以下のプログラムを適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          定　数                                                   */
/*****************************************************************************************************/
// シンプルシェイプタイプ
enum DXGSHAPETYPE {
    DXGSHAPE_BOX = 1,   // 立方体
    DXGSHAPE_CYLINDER,  // 円柱または円錐
    DXGSHAPE_POLYGON,   // 多角形
    DXGSHAPE_SPHERE,    // 球
    DXGSHAPE_TORUS,     // トーラス
    DXGSHAPE_TEAPOT     // ティーポット
};
```

この定数は、シンプルシェイプを生成するときに、どの形状を生成すればよいのかを判別するために用います。

(２)シンプルシェイプ生成関数の引数を共通化しましょう。

前述６つの関数で異なる部分は、シンプルシェイプ生成関数とその引数です。まず、引数をまとめることを考えます。シンプルシェイプを生成するのに必要なすべての情報を構造体にまとめれば、引数を共通化することができます。

以下の構造体を適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          構造体定義                                               */
/*****************************************************************************************************/
// シンプルシェイプ構造体
struct DXGSIMPLESHAPE {
    DXGSHAPETYPE    Type;           // 形状のタイプ
    float           Width;          // 立方体の幅
    float           Height;         // 立方体の高さ
    float           Depth;          // 立方体の奥行き

    float           Length;         // 円柱または多角形の辺の長さ

    float           Radius;         // 球の半径
    float           Radius1;        // 円柱の先頭の半径
    float           Radius2;        // 円柱の後尾の半径

    float           InnerRadius;    // トーラスの内側の半径
    float           OuterRadius;    // トーラスの外側の半径

    UINT            Sides;          // 多角形またはトーラスの横断面の辺の数
    UINT            Slices;         // 円柱、球の横断面の辺の数
    UINT            Stacks;         // 横断面の数
    UINT            Rings;          // トーラスを構成する環の数
};
```

(３)(２)の構造体は、引数をそのままままとめただけなので、無駄が多く見られます。以下のように同時に使用されない情報および酷似した情報を共用体にまとめましょう。

```
// シンプルシェイプ構造体
struct DXGSIMPLESHAPE {
    DXGSHAPETYPE    Type;           // 形状のタイプ
    float           Width;          // 立方体の幅
    float           Height;         // 立方体の高さ
    union {
        float       Depth;          // 立方体の奥行き
        float       Length;         // 円柱または多角形の辺の長さ
    };
    union {
        float       Radius;         // 球の半径
        float       Radius1;        // 円柱の先頭の半径
        float       InnerRadius;    // トーラスの内側の半径
    };
    union {
        float       Radius2;        // 円柱の後尾の半径
        float       OuterRadius;    // トーラスの外側の半径
    };
    union {
        UINT        Sides;          // 多角形またはトーラスの横断面の辺の数
        UINT        Slices;         // 円柱、球の横断面の辺の数
    };
    union {
        UINT        Stacks;         // 横断面の数
        UINT        Rings;          // トーラスを構成する環の数
    };
};
```

(４)シンプルシェイプ生成関数を１つにまとめます。形状のタイプ(DXGSIMPLESHAPE構造体のTypeメンバ)から呼び出す関数と使用するメンバを決定するようにします。
以下のプログラムの足りない部分を補い、適切な場所に追加しましょう(プロトタイプの追加も忘れずに行いましょう)。

```
/*****************************************************************************************************/
/*                                        シンプルシェイプ生成                                       */
/*****************************************************************************************************/
```

```
IMesh* CDXGraphics9::CreateSimpleShape(const DXGSIMPLESHAPE& inShape)
{
#ifdef _DEBUG
    if(m_pD3DDevice == NULL) {
        ::OutputDebugString("*** Error - Direct3DDevice9未初期化(CDXGraphics9_CreateTeapot)\n");
        return NULL;
    }
#endif

    // シンプルシェイプ生成
    ID3DXMesh*   pMesh = NULL;
    HRESULT      hr    = D3DXERR_INVALIDDATA;
    switch(inShape.Type) {
      case DXGSHAPE_BOX:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, inShape.Width, inShape.Height, inShape.Depth,
                                        &pMesh, NULL);
        break;

      case DXGSHAPE_CYLINDER:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, inShape.Radius1, inShape.Radius2, inShape.Length,
                                        inShape.Slices, inShape.Stacks, &pMesh, NULL);
        break;

      case DXGSHAPE_POLYGON:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, inShape.Length, inShape.Sides, &pMesh, NULL);
        break;

      case DXGSHAPE_SPHERE:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, inShape.Radius, inShape.Slices, inShape.Stacks,
                                        &pMesh, NULL);
        break;

      case DXGSHAPE_TORUS:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, inShape.InnerRadius, inShape.OuterRadius,
                                        inShape.Sides, inShape.Rings, &pMesh, NULL);
        break;

      case DXGSHAPE_TEAPOT:
        hr = ::※ここは各自考えましょう(m_pD3DDevice, &pMesh, NULL);
        break;
    };

    // メッシュオブジェクト生成
    IMesh*   pShape;
    if(hr == D3D_OK) {
        pShape = new ?????(m_pD3DDevice, pMesh, 1);
    } else {
        ::OutputDebugString("*** Error - シンプルシェイプ生成失敗(CDXGraphics9_CreateSimpleShape)\n");
        pShape = new ??????????();
    }
    SafeRelease(pMesh);

    m_Mesh.push_back(pShape);       // メッシュリストへ追加

    return pShape;
}
```

(５)CDXGraphics9::CreateBox関数からCreateTeapot関数の６つの関数は不要なので、プロトタイプと関数本体をコメントにするか、削除しましょう。

(６)CTestSceneクラスのコンストラクタで行っている「ティーポット生成」を以下のように変更しましょう。

```
    // ティーポット生成
    DXGSIMPLESHAPE   shape;
    shape.Type = DXGSHAPE_TEAPOT;
    m_pMesh    = DXGraphics().CreateSimpleShape(shape);
```