オブジェクト指向とゲームプログラミング

▶ DirectX Graphics 3D編 － 第８回　テクスチャ ◀

テクスチャ

テクスチャとは、3DCGで物体表面の質感を表現するために貼り付ける画像のことをいいます。同じ立方体のメッシュでも、金属のテクスチャを貼り付ければ金属片に見え、木目のテクスチャを貼り付ければ木片に見えます。
テクスチャを用いると、ポリゴンの組み合わせやマテリアルでは表現できない複雑な模様を表現することができます。また、テクスチャを貼ることにより、少ないポリゴンでもリアリティあるオブジェクトを作ることができます。

テクスチャの生成

テクスチャには、さまざまな制限があります。もっとも特徴的なのは、テクスチャの大きさです。テクスチャの大きさは、128ピクセルや256ピクセルといったように、2^nに制限されます。最大サイズは、環境により異なりますが、DirectX Graphicsの仕様の上限は4,096×4,096ピクセルです。また、サポートするフォーマットも環境により異なります。テクスチャを生成する場合は、これらを考慮しなければなりません。

テクスチャの生成はIDirect3DDevice9::CreateTextureメソッド、D3DXCreateTexture関数、D3DXCreateTextureFromFile関数などで行います。Direct3DXの関数を用いれば、環境の差異を吸収してテクスチャを生成することができるので、コードを大幅に削減することができます。

```
// テクスチャ生成(pD3DDeviceは初期化済みのDirect3DDevice9オブジェクト)
IDirect3DTexture9*   pTexture;
::D3DXCreateTextureFromFile(pD3DDevice, "Texture.png", &pTexture)
```

テクスチャを生成すると、テクスチャを制御するためのIDirect3DTexture9インタフェースが返ります。

テクスチャの設定

オブジェクトにテクスチャを適用するには、テクスチャを生成したときに得られたインタフェースをIDirect3DDevice9::SetTexutreメソッドに渡すことで行います。以後、すべてのオブジェクトのレンダリング時に、設定したテクスチャが適用されます。

```
// テクスチャ設定
pD3DDevice->SetTexture(0, pTexture);
```

オブジェクトの表面は、デフォルトではマテリアルとテクスチャの色が乗算されて適用されます。マテリアルが黒[RGB(0.0, 0.0, 0.0)]の場合は、乗算の結果が０になるため、オブジェクトにテクスチャが適用されていないように見えます。

テクスチャ座標(ＵＶ座標)

テクスチャ座標とは、ＵＶ座標とも呼ばれ、テクスチャに付けられた座標のことです。テクスチャの大きさにかかわらず、左上が(0.0, 0.0)、右下が(1.0, 1.0)となります。

ポリゴン(頂点)に単純にテクスチャを貼ってしまうと、ポリゴンの移動、変形に対応できないという問題が生じてしまいます。そこで、DirectX Graphicsでは「この頂点には、テクスチャのこの部分を貼る」という方法をとっています。このようにすれば、ポリゴンが動いても変形しても、テクスチャが正しく追従できるというわけです。これが、テクスチャ座標の基本的な考え方です。

課　　題

頂点バッファクラスとメッシュクラスにテクスチャを設定する機能を追加しましょう。

(１)DXGraphics.hppに定義された定数や構造体をまとめたヘッダファイルを作成します。

DXGraphics.hppには、いくつか定数や構造体が定義されていますが、数が多くなってくると、ヘッダファイルの見通しが悪くなり、保守性が低下します。そこで、定数と構造体を別のヘッダファイルにまとめます。こうすると、DXGraphics.hppはクラス定義のみになり、管理しやすくなります。

型宣言をまとめた以下のヘッダファイル"DXGTypes.h"を作成し、プロジェクトに追加しましょう。

－DXGTypes.h－

```
/*
===========================================================================================
                                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】
    DXGTypes.h
                タイプヘッダ

  【履歴】
    * Version    1.00       2005/04/dd  hh:mm:ss

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

#pragma once

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

/*********************************************************************************************/
/*                                          定　数                                           */
/*********************************************************************************************/
// FVFフラグ定義
enum { DXGFVF_TLVERTEX = D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1,
       DXGFVF_LVERTEX  = D3DFVF_XYZ    | D3DFVF_DIFFUSE | D3DFVF_TEX1,
       DXGFVF_VERTEX   = D3DFVF_XYZ    | D3DFVF_NORMAL  | D3DFVF_TEX1 };

// シンプルシェイプタイプ
enum DXGSHAPETYPE {
    DXGSHAPE_BOX = 1,
    DXGSHAPE_CYLINDER,
    DXGSHAPE_POLYGON,
    DXGSHAPE_SPHERE,
    DXGSHAPE_TORUS,
    DXGSHAPE_TEAPOT
};

/*********************************************************************************************/
/*                                      構造体定義                                           */
/*********************************************************************************************/
// トランスフォーム済みライティング済みの頂点
```

```
struct DXGTLVERTEX {
    float       x, y, z, rhw;    // 頂点座標
    D3DCOLOR    color;           // 頂点色
    float       tu, tv;          // テクスチャ座標

    // コンストラクタ
    DXGTLVERTEX() : x(0.0f), y(0.0f), z(0.0f), rhw(0.0f), color(0), tu(0.0f), tv(0.0f) {}
};

// 未トランスフォームライティング済みの頂点
struct DXGLVERTEX {
    float       x, y, z;         // 頂点座標
    D3DCOLOR    color;           // 頂点色
    float       tu, tv;          // テクスチャ座標

    // コンストラクタ
    DXGLVERTEX() : x(0.0f), y(0.0f), z(0.0f), color(0), tu(0.0f), tv(0.0f) {}
};

// 未トランスフォーム未ライティングの頂点
struct DXGVERTEX {
    float   x, y, z;             // 頂点座標
    float   nx, ny, nz;          // 法線ベクトル
    float   tu, tv;              // テクスチャ座標

    // コンストラクタ
    DXGVERTEX() : x(0.0f), y(0.0f), z(0.0f), nx(0.0f), ny(0.0f), nz(0.0f), tu(0.0f), tv(0.0f) {}
};

// シンプルシェイプ構造体
struct DXGSIMPLESHAPE {
    DXGSHAPETYPE    Type;        // シェイプのタイプ
    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
    };
};
```

(２)マテリアルとテクスチャをまとめた構造体を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
// マテリアル構造体
struct DXGMATERIAL {
    D3DMATERIAL9            Material;
    IDirect3DTexture9*      pTexture;
};
```

3Dオブジェクトの表面は、マテリアルとテクスチャによって決定されるので、これらをまとめた構造体を定義します。

(３)CVertexBufferクラスのマテリアルを保持するメンバ変数を以下のように変更しましょう。

```
DXGMATERIAL   m_Material;       // マテリアル
```

DXGMATERIAL構造体を使用するために必要なヘッダファイルをインクルードすることも忘れずに行いましょう。

(４)CVertexBufferクラスのコンストラクタで行っているマテリアルの初期化処理を以下のように変更しましょう。

```
// マテリアル初期化
::ZeroMemory(&m_Material.Material, sizeof(m_Material.Material));
m_Material.pTexture = NULL;
```

(５)CVertexBufferクラスに、IDirect3DTexture9インタフェースの解放処理を追加します。以下のプログラムをデストラクタに追加しましょう。

```
SafeRelease(m_Material.pTexture);
```

(６)CVertexBuffer::SetMaterial関数を以下のように変更しましょう。

```
virtual void SetMaterial(const D3DMATERIAL9& inMaterial) { m_Material.Material = inMaterial; }
```

(７)CVertexBufferクラスに、テクスチャを設定する関数を追加します。以下のプログラムを完成させ、適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                          テクスチャ設定                                           */
/*****************************************************************************************************/
void CVertexBuffer::SetTexture(LPCTSTR inFileName)
{
    SafeRelease(m_Material.pTexture);   // 古いテクスチャの解放

    // テクスチャ生成
    if(::?????????????????????????(m_pD3DDevice, inFileName, &m_Material.pTexture) != D3D_OK)
        ::OutputDebugString("*** Error - テクスチャ生成失敗(CVertexBuffer_SetTexture)\n");
}
```

(８)頂点バッファクラスに、マテリアルとテクスチャを設定する関数を追加します。以下のプログラムをIVertexBufferクラスの適切な場所に追加しましょう(※第６回で作成したものとは引数が異なるので注意しましょう)。

```
virtual void SetMaterial(const D3DXMATERIAL& inMaterial) = 0;
```

(９)CVertexBufferクラスの適切な場所に、以下のプログラムを追加しましょう。

```
virtual void SetMaterial(const D3DXMATERIAL& inMaterial)
{ SetMaterial(inMaterial.MatD3D); SetTexture(inMaterial.pTextureFilename); }
```

(10)CNullVertexBufferクラスに「何もしない」SetMaterial関数を作成しましょう。

(11)頂点バッファのレンダリング時に、テクスチャが適用されるようにします。CVertexBuffer::Renderer関数の適切な場所に、以下のプログラムを完成させて追加しましょう。

```
// テクスチャ設定
if((m_FVF & D3DFVF_TEX1) != 0)
    m_pD3DDevice->???????????(0, m_Material.pTexture);
```

(12)(３)～(11)を参考に、メッシュクラスにテクスチャを設定する機能を追加しましょう(メッシュには、インデックスがあることに注意しましょう)。

(13)ポリゴンにテクスチャを貼り付けてみます。まず、ＵＶ座標を定義した三角形ポリゴンを２つ合わせて四角形を作成します。CTestScene::CTestScene関数を次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
    // 頂点バッファの生成(トランスフォーム済みライティング済みの頂点を１００個格納するバッファ)
    // 頂点バッファ生成
    m_pVertex = DXGraphics().CreateVertexBuffer(sizeof(DXGTLVERTEX) * 100, DXGFVF_TLVERTEX,
                                                sizeof(DXGTLVERTEX));

    // 頂点バッファをロックし、ポインタを得る
    DXGTLVERTEX*   vt = (DXGTLVERTEX*)m_pVertex->Lock();

    // 頂点０
    vt[0].x     = 260.0f;                                   // ｘ座標
    vt[0].y     = 180.0f;                                   // ｙ座標
    vt[0].color = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);  // 色
    vt[0].tu    = 0.0f;                                     // テクスチャＵ座標
    vt[0].tv    = 0.0f;                                     // テクスチャＶ座標

    // 頂点１
    vt[1].x     = 380.0f;
    vt[1].y     = 300.0f;
    vt[1].color = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
    vt[1].tu    = 0.2f;
    vt[1].tv    = 1.0f;

    // 頂点２
    vt[2].x     = 260.0f;
    vt[2].y     = 300.0f;
    vt[2].color = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
    vt[2].tu    = 0.0f;
    vt[2].tv    = 1.0f;

    // 頂点３
    vt[3] = vt[0];

    // 頂点４
    vt[4].x     = 380.0f;
    vt[4].y     = 180.0f;
    vt[4].color = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
    vt[4].tu    = 0.2f;
    vt[4].tv    = 0.0f;

    // 頂点５
    vt[5] = vt[1];
}
```

(14)CTestScene::ActiveProc関数を以下のように変更しましょう。

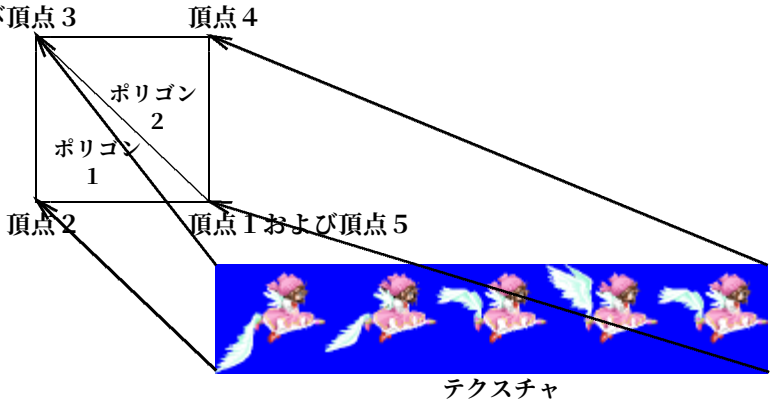
```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します

    // 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_pVertex->Render(D3DPT_TRIANGLELIST, 0, 2);

    DXGraphics().EndScene();

    DXGraphics().UpdateFrame();

    return 0;
}
```

(15)頂点バッファにテクスチャを設定します。以下のプログラムをCTestScene::CTestScene関数の適切な場所に追加しましょう(テクスチャの画像は、あらかじめ作成しておいてください)。

```
    // 頂点バッファにテクスチャを設定
    m_pVertex->SetTexture("Texture\\Block.bmp");
```

(16)次のような2Dキャラクターのアニメーションパターンが定義された画像を作成し、テクスチャにしてみましょう。

現在の状態は、正方形のポリゴンに上記のテクスチャの全領域を貼り付ける設定になっています。



テクスチャは、ポリゴンの頂点を追従するので、実行結果のように変形されて貼り付けられます。テクスチャ座標を正しく設定することにより、任意の部分を表示することができます。

(17)ポリゴンにキャラクターの１コマ目を貼りつけるようにしましょう。
ヒント１：頂点０，頂点２，頂点３のテクスチャ座標は、変更する必要はありません。
ヒント２：Ｕ座標がテクスチャ座標の横方向、Ｖ座標が縦方向です。
ヒント３：パターンが５つあり、それぞれ等間隔で配置されています。１コマ目のテクスチャ座標は、左上が(0.0，0.0)、右上が(0.2，0.0)、右下が(0.2，1.0)、左下が(0.0，1.0)となります。
ヒント４：頂点４が１コマ目の右上(0.2，0.0)、頂点１，頂点５が１コマ目の左下(0.2，1.0)です。
ヒント５：テクスチャのＶ座標は変更しません。Ｕ座標のみ変更します。
ヒント６：
```
vt[1].tu = 0.2f;
vt[1].tv = 1.0f;
```

(18)ポリゴン描画の度にパターンを変更し、アニメーションさせてみます。以下のメンバ変数をCTestScene::ActiveProc関数の適切な場所に追加しましょう。

```
// 頂点バッファへアクセスしてＵＶ座標を変更する
DXGTLVERTEX*    vt = (DXGTLVERTEX*)m_pVertex->Lock();

// 次のパターンに更新
vt[0].tu += 0.2f;     // テクスチャ座標を0.2増やし、次のパターンへ
if(vt[0].tu >= 1.0f)  // もし、パターンが1.0以上になったら、
    vt[0].tu = 0.0f;  // 0.0に戻す

vt[3].tu = vt[2].tu = vt[0].tu;
vt[5].tu = vt[4].tu = vt[1].tu = vt[0].tu + 0.2f;

m_pVertex->Unlock();
```

(19)立方体のメッシュを作成し、テクスチャを貼ってみましょう。

発展問題　DXGMATERIAL構造体にコンストラクタとデストラクタを追加しましょう。

DXGMATERIAL構造体はクラスではありませんが、C++では、構造体にもコンストラクタとデストラクタを定義することができます。これらを定義しておくことで、各メンバの初期化処理と解放処理を省くことができるようになります。

(１)DXGMATERIAL構造体にコンストラクタとデストラクタを追加します。以下のように変更しましょう。

```
// マテリアル構造体
struct DXGMATERIAL {
    D3DMATERIAL9        Material;
    IDirect3DTexture9*  pTexture;

    DXGMATERIAL() : pTexture(NULL) { ::ZeroMemory(&Material, sizeof(Material)); }
    ~DXGMATERIAL() { if(pTexture != NULL) pTexture->Release(); }
};
```

(２)コンストラクタとデストラクタにより、DXGMATERIAL構造体の初期化処理および解放処理は不要になります。これらを行っている部分を削除しましょう。

－削除箇所－

・CVertexBuffer::CVertexBuffer()

```
// マテリアル初期化
::ZeroMemory(&m_Material.Material, sizeof(m_Material.Material));
m_Material.pTexture = NULL;
```

・CVertexBuffer::~CVertexBuffer()

```
SafeRelease(m_Material.pTexture);
```

・CMesh::CMesh()

```
// マテリアル初期化
for(DWORD i = 0; i < m_SubsetCount; i++) {
    ::ZeroMemory(&m_Material[i], sizeof(m_Material[i]));
    m_Material[i].pTexture = NULL;
}
```

・CMesh::~CMesh()

```
// テクスチャ解放
for(DWORD i = 0; i < m_SubsetCount; i++)
    SafeRelease(m_Material[i].pTexture);
```