Chapter 1

X ファイルを使用したアニメーションしないモデル表示。

1.1 X ファイル

X ファイルとは DirectX2.0 から導入されたモデルフォーマットで、DirectX9 までサポートされていました。DirectX10 以降は標準モデルフォーマットというものは用意されなくなり、自前でモデル表示処理を実装する必要があります。

現在はサポートされていないフォーマットのため、X ファイルの使い方を学ぶ意味は薄いように思えるかもしれませんが、そもそもどこの環境に行っても使えるモデルフォーマットなど存在しません。自前でエンジンを作っている会社は自分たちでモデルフォーマットを作成しています。しかし、モデルを表示するための基本的な概念はどのモデルフォーマットでも共通となっており、x ファイルを使用したモデル表示の仕方を学んでおけば、独自のモデルフォーマットを使用するライブラリに出会ったとしても、「似たような感じだな」というように思えるはずです。

Tips
モデルの表示に関して、どの環境でも通用する技術とは頂点バッファ、インデックスバッファ、シェーダーなど低レベルな知識になります。非常に重要な知識で先生は大好きな分野なのですが、２年生の前期にこれをやっていると就職作品の作成を開始するまでにアニメーションするモデル表示まで話を勧められません。ですので、この手の基礎の話は後期に行います。

1.2　X ファイルのロード

X ファイルを用いてモデルを表示するためには、D3DXLoadMeshFromX 関数を使用して X ファイルをロードして、ID3DXMesh のインスタンスを作成する必要があります。ID3DXMesh とは内部にモデルを表示するための頂点バッファやインデックスバッファを保持したモデルクラスのようなものです。この API は下記のように使用します。

```
D3DXLoadMeshFromX(
   "Tiger.x",                    //ファイルパス
   D3DXMESH_SYSTEMMEM,           //メッシュ作成のオプション。基本これでいい。
                                 //他にも大事なオプションはあるのだが、今は説明しない。
   g_pd3dDevice,                 //D3D デバイス。
   NULL,                         //ポリゴンの隣接情報の出力先。
                                 //モデルをロードするだけなら NULL でいい。
   &pD3DXMtrlBuffer,             //マテリアルバッファの出力先。後述。
   NULL,                         //NULL でいい。
   &g_dwNumMaterials,            //マテリアルの数の出力先。後述。
   &g_pMesh                      //ID3DXMesh のインスタンスの格納先。
)
```

これで ID3DXMesh のインスタンスが生成されました。

1.3 マテリアル

マテリアルとはモデルの質感を決定するためのものです。例えばテクスチャ、鏡面反射率などの設定を行うものです。D3DXLoadMeshFromX から取得できるテクスチャ以外のマテリアル情報は固定機能と呼ばれる、現在は廃れた機能の情報しか取得できないため。今回使用するサンプルでは使用しません。今回のサンプルではマテリアル情報はテクスチャを引っ張ってくるためだけに使用します。マテリアルからテクスチャを引っ張ってくるコードは下記のようになります。

```
D3DXMATERIAL* d3dxMaterials = ( D3DXMATERIAL* )pD3DXMtrlBuffer->GetBufferPointer();
//テクスチャ配列を new
g_pMeshTextures = new LPDIRECT3DTEXTURE9[g_dwNumMaterials];
//マテリアルの数だけループを回してテクスチャをロード。
for( DWORD i = 0; i < g_dwNumMaterials; i++ )
{
    g_pMeshTextures[i] = NULL;
    if( d3dxMaterials[i].pTextureFilename != NULL &&
        lstrlenA( d3dxMaterials[i].pTextureFilename ) > 0 )
    {
        // テクスチャを作成。
        if( FAILED( D3DXCreateTextureFromFileA( g_pd3dDevice,
                                                d3dxMaterials[i].pTextureFilename,
                                                &g_pMeshTextures[i] ) ) )
        {
            /テクスチャが見つからなかった。
            MessageBox( NULL, "Could not find texture map", "Meshes.exe", MB_OK );
        }
    }
}
```

1.4 エフェクトファイルのロード。

モデルを表示するためには、拡張子が.fx のエフェクトファイルと言われるものをロードする必要があります。このエフェクトファイルは HLSL という言語で記述されたシェーダープログラムになります。シェーダーは近年のゲームのグラフィックスを語る上で欠かすことのできない、非常に重要な要素になります。しかし、この話をするだけでかなりの時間がかかりますので、この話は後期に行います。今はこのように記述を行う必要があるのだなという風にだけ覚えておいてください。

エフェクトファイルのロードは下記のように行います。

```
//シェーダーをコンパイル。
HRESULT hr = D3DXCreateEffectFromFile(
            g_pd3dDevice,
            "basic.fx",
            NULL,
            NULL,
#ifdef _DEBUG
            D3DXSHADER_DEBUG,
#else
            D3DXSHADER_SKIPVALIDATION,
#endif
            NULL,
            &g_pEffect,
            &compileErrorBuffer
            );
if (FAILED(hr)) {
            MessageBox(NULL, (char*)(compileErrorBuffer->GetBufferPointer()), "error", MB_OK);
            std::abort();
}
```

1.5 モデルの描画処理

ここまでは全て初期化と言われる処理で、これでやっとモデルを表示することができます。では実際にモデルを描画するコードを見てみましょう。

```
//シェーダー適用開始。
g_pEffect->SetTechnique("SkinModel");
g_pEffect->Begin(NULL, D3DXFX_DONOTSAVESHADERSTATE);
g_pEffect->BeginPass(0);

//定数レジスタに設定するカラー。
D3DXVECTOR4 color( 1.0f, 0.0f, 0.0f, 1.0f);
//ワールド行列の転送。
g_pEffect->SetMatrix("g_worldMatrix", &g_worldMatrix);
//ビュー行列の転送。
g_pEffect->SetMatrix("g_viewMatrix", &g_viewMatrix);
//プロジェクション行列の転送。
g_pEffect->SetMatrix("g_projectionMatrix", &g_projectionMatrix);
 //回転行列を転送。
g_pEffect->SetMatrix( "g_rotationMatrix", &g_rotationMatrix );
//ライトの向きを転送。
g_pEffect->SetVectorArray("g_diffuseLightDirection", g_diffuseLightDirection, LIGHT_NUM );
//ライトのカラーを転送。
g_pEffect->SetVectorArray("g_diffuseLightColor", g_diffuseLightColor, LIGHT_NUM );
//環境光を設定。
g_pEffect->SetVector("g_ambientLight", &g_ambientLight);

//この関数を呼び出すことで、データの転送が確定する。描画を行う前に一回だけ呼び出す。
g_pEffect->CommitChanges();

// Meshes are divided into subsets, one for each material. Render them in
    // a loop
    for( DWORD i = 0; i < g_dwNumMaterials; i++ )
    {
        //テクスチャを設定。
        g_pEffect->SetTexture("g_diffuseTexture", g_pMeshTextures[i]);
        //描画。
        g_pMesh->DrawSubset( i );
    }

g_pEffect->EndPass();
g_pEffect->End();
```

よくわからないコードが多いかと思います。非常に長いコードになりましたが、これがモデルを表示するときに必要なコードになります。まだ、よく分からない部分がいくつかあるかと思いますが、今は構いません。少なくともワールド行列、ビュー行列、プロジェクション行列の設定などをしている部分を分かってもらうだけで今は十分です。

1.6 終了処理。

プログラムが終了、もしくはモデル表示が不要になった場合は、ここまでロードしたID3DXMesh やテクスチャ、エフェクトファイルなどを破棄する必要があります。下記に破棄を行うコードを記述します。モデルが不要になったら必ず終了処理を実行するように気をつけてください。

```
//テクスチャを破棄
if (g_pMeshTextures != NULL) {
   for (int i = 0; i < g_dwNumMaterials; i++) {
      g_pMeshTextures[i]->Release();
   }
   delete[] g_pMeshTextures;
}
//メッシュを破棄
if (g_pMesh != NULL) {
   g_pMesh->Release();
}
//エフェクトを破棄
if (g_pEffect != NULL) {
   g_pEffect->Release();
}
```

1.7 まとめ

X ファイルを使用してモデルを表示するためには下記の手順が必要でした。

① D3DXLoadMeshFromX 関数 を使用して X ファイルをロードし、ID3DXMesh のインスタンスを作成する。(初期化時に一度だけ実行)

② D3DXLoadMeshFromX 関数を使用して取得できたマテリアル情報を元に D3DXCreateTextureFromFileA 関数を使用してテクスチャをロードする。(初期化時に一度だけ実行)

③ D3DXCreateEffectFromFile 関数を使用してエフェクトファイルをロードする。(初期化時に一度だけ実行)

④ ロードした要素を使用してモデルの描画処理を記述する。(毎フレーム実行する)

実習課題

下記の URL から実習用のプログラムを pull して、実習を行ってください。

① トラをクラス化してみましょう。

トラのクラスの最低限の要求仕様。

・トラのクラスは下記のメンバ関数実装する。

Init 関数を実装するようにしてく

X ファイル、テクスチャ、エフェクトファイルのロードなどの処理を記述する。

Update 関数

ワールド行列の更新やトラの移動などを記述する関数。

Render 関数

トラの描画処理を記述する。

Release 関数

メッシュ、エフェクト、テクスチャなどを破棄するコードを記述する。

・トラのクラスは下記のメンバ変数を最低限保持する。

```
ID3DXEffect*     pEffect;
D3DXMATRIX    worldMatrix;
LPD3DXMESH    pMEsh;
LPDIRECT3DTEXTURE9* pMeshTextures
DWORD numMaterial;
```

② トラを２体出してください。

③ トラのクラスからモデルの処理を抽出して、モデルクラスを作成してください。