オブジェクト指向とゲームプログラミング

▶ DirectX Graphics 2D編 － 第２回　スプライトの生成 ◀

スプライトとテクスチャ

Direct3DXのスプライトは、ポリゴンにテクスチャを貼ったもので擬似的に実現しています。そのため、スプライトを描画する場合は、テクスチャを生成し、そこに画像を読み込まなければなりません。
Direct3DXには、スプライトを生成する関数はありません。テクスチャを生成して画像を読み込み、それをID3DXSpriteインタフェースで描画する、という流れになります。

テクスチャの生成

テクスチャは、Direct3DTexture9オブジェクトが管理します。また、テクスチャの画像は、IDirect3DSurface9インタフェースをとおして操作を行います。Direct3DTexture9オブジェクトを生成すると、画像を管理するDirect3DSurface9オブジェクトも内部で自動的に生成されます。

Direct3DXでは、テクスチャサーフェスを生成する関数を提供しています。空のテクスチャを生成するD3DXCreateTexture関数や、画像ファイルからテクスチャを生成するD3DXCreateTextureFromFile関数のほかに、メモリ、リソースにある画像からテクスチャを生成することが簡単にできます。

DirectX Graphicsでは、テクスチャのサイズは２の累乗に制限されます(ただし、ビデオカードが２の累乗でないテクスチャをサポートしており、かつフラグによって指定した場合を除く)。また、仕様上の最大サイズは4,096×4,096ピクセルですが、環境によってはこれ以下になります。
Direct3DXのテクスチャ生成関数は、この差異を吸収してテクスチャを生成しますが、任意のサイズのテクスチャを指定したとしても、２の累乗に拡大されます。そのため、読み込んだ画像がサイズ補正フィルタリングをされ、一部のピクセルがぼやけたり、意図しない大きさになる場合があります。

D3DXCreateTexture関数

－説明－
D3DXCreateTexture関数は、空のテクスチャを生成します。

－書式－
```
HRESULT D3DXCreateTexture(LPDIRECT3DDEVICE9 pDevice, UINT Width, UINT Height,
                          UINT MipLevels, DWORD Usage, D3DFORMAT Format, D3DPOOL Pool,
                          LPDIRECT3DTEXTURE9* ppTexture);
```

－パラメータ－
１つ目の引数(pDevice)は、テクスチャに関連づけるデバイスのIDirect3DDevice9インタフェースです。

２つ目の引数(Width)は、生成するテクスチャの幅です。必ず２の累乗に切り上げられます。

３つ目の引数(Height)は、生成するテクスチャの高さです。必ず２の累乗に切り上げられます。

４つ目の引数(MipLevels)は、ミップマップの数です。この値が０またはD3DX_DEFAULTの場合、テクスチャのサイズから１×１までの完全なミップマップチェーンが作成されます。

５つ目の引数(Usage)は、テクスチャの使い方を指定します。０または以下のフラグを指定します。

D3DUSAGE_RENDERTARGET	レンダリングターゲット
D3DUSAGE_DYNAMIC	動的テクスチャ

６つめの引数(Format)は、要求するフォーマットを記述します。要求したフォーマットをデバイスがサポートしていない場合は、指定したものと異なるテクスチャが生成される場合があります。

７つ目の引数(Pool)は、テクスチャを配置する以下のメモリクラスフラグを指定します。

D3DPOOL_DEFAULT	テクスチャを格納するのに最も適したメモリ(VRAMまたはAGPメモリ)に配置されます。このフラグを使って生成されたリソースは、IDirect3DDevice9::Resetメソッドを呼び出す前に、解放しなければなりません
D3DPOOL_MANAGED	DirectX Graphicsがアクセスできるメモリにテクスチャを生成し、さらにシステムメモリにバックアップを作成して自動メモリ管理を行います。ほとんどの場合、このフラグを用います
D3DPOOL_SYSTEMMEM	システムメモリに生成されます

７つ目の引数(ppTexture)は、生成されるテクスチャオブジェクトのインタフェースを格納する変数(IDirect3DTexture9*型またはLPDIRECT3DTEXTURE9型)のアドレスです。

－戻り値－
成功した場合はD3D_OK、それ以外の場合はエラーコードを返します。

```
// 1,024×512の空のテクスチャを生成(pD3DDeviceは、Direct3DDevice9オブジェクト)
IDirect3DTexture9*   pD3DTexture;          // テクスチャオブジェクトのインタフェース
D3DXCreateTexture(pD3DDevice, 1024, 512, 1, 0,
                  D3DFMT_A8R8G8B8, D3DPOOL_MANAGED, &pD3DTexture);
```

D3DXCreateTextureFromFileEx関数

－説明－
D3DXCreateTextureFromFileEx関数は、ファイルを基にテクスチャを作成します。この関数がサポートしているファイルフォーマットは、.bmp、.dds、.dib、.hdr、.jpg、.pfm、.png、.ppm、.tgaです。

－書式－
```
HRESULT D3DXCreateTextureFromFileEx(LPDIRECT3DDEVICE9 pDevice, LPCTSTR pSrcFile,
                                    UINT Width, UINT Height, UINT MipLevels,
                                    DWORD Usage, D3DFORMAT Format, D3DPOOL Pool,
                                    DWORD Filter, DWORD MipFilter, D3DCOLOR ColorKey,
                                    D3DXIMAGE_INFO *pSrcInfo, PALETTEENTRY *pPalette,
                                    LPDIRECT3DTEXTURE9 *ppTexture);
```

－パラメータ－
１つ目の引数(pDevice)は、テクスチャに関連づけるデバイスのIDirect3DDevice9インタフェースを指定します。

２つめの引数(pSrcFile)は、読み込むファイル名です。

３つ目の引数(Width)は、生成するテクスチャの幅です。この値が０またはD3DX_DEFAULTの場合は、ファイルからサイズが取得され、２の累乗に切り上げられます。デバイスが２の累乗でないテクスチャをサポートしていて、D3DX_DEFAULT_NONPOW2が指定されている場合は、サイズは切り上げられません。

４つ目の引数(Height)は、生成するテクスチャの高さです。この値が０またはD3DX_DEFAULTの場合は、ファイルからサイズが取得され、２の累乗に切り上げられます。デバイスが２の累乗でないテクスチャをサポートしていて、D3DX_DEFAULT_NONPOW2が指定されている場合は、サイズは切り上げられません。

５つ目の引数(MipLevels)は、ミップマップの数です。この値が０またはD3DX_DEFAULTの場合、テクスチャのサイズから１×１までの完全なミップマップチェーンが作成されます。D3DX_FROM_FILEの場合は、サイズはファイルのとおりに取得され、これがデバイス能力を侵害する場合は関数が失敗します。

６つ目の引数(Usage)は、テクスチャの使い方を指定します。０または以下のフラグを指定します。

D3DUSAGE_RENDERTARGET	レンダリングターゲット
D3DUSAGE_DYNAMIC	動的テクスチャ

７つめの引数(Format)は、要求するフォーマットを記述します。要求したフォーマットをデバイスがサポートしていない場合は、指定したものと異なるテクスチャが生成される場合があります。D3DFMT_UNKNOWNを指定すると、ファイルからフォーマットを取得します。D3DFMT_FROM_FILEの場合は、フォーマットはファイルのとおりに取得され、これがデバイス能力を侵害する場合は関数が失敗します。

８つ目の引数(Pool)は、テクスチャを配置する以下のメモリクラスフラグを指定します。

D3DPOOL_DEFAULT	テクスチャを格納するのに最も適したメモリ(VRAMまたはAGPメモリ)に配置されます。このフラグを使って生成されたリソースは、IDirect3DDevice9::Resetメソッドを呼び出す前に、解放しなければなりません
D3DPOOL_MANAGED	DirectX Graphicsがアクセスできるメモリにテクスチャを生成し、さらにシステムメモリにバックアップを作成して自動メモリ管理を行います。ほとんどの場合、このフラグを用います
D3DPOOL_SYSTEMMEM	システムメモリに生成されます

９つ目の引数(Filter)は、イメージをリサイズするときのフィルタフラグです。D3DX_DEFAULTを指定すると、「D3DX_FILTER_TRIANGLE | D3DX_FILTER_DITHER」と同じになります。

10個目の引数(MipFilter)は、ミップマップイメージのフィルタフラグです。D3DX_DEFAULTを指定すると、D3DX_FILTER_BOX と同じになります。

11個目の引数(ColorKey)は、透明となる色(D3DCOLOR型)の値。カラーキーを無効にする場合は0を指定します。入力イメージのフォーマットとは関係なく、常にD3DCOLOR型(ARGB)で指定します。アルファが有意であり、通常はカラーキーを不透明にする場合は0xffを指定します。したがって、不透明な黒の場合、値は0xff000000となります。ここで指定した色をもつすべてのピクセルが、アルファ値0(完全な透明)になります。

12個目の引数(pSrcInfo)は、入力イメージファイル内のデータの記述を格納するD3DXIMAGE_INFO構造体のアドレスを指定します。NULLをできます。

13個目の引数(pPalette)は、テクスチャがパレットを持つ場合に、それを格納するPALETTEENTRY構造体のアドレスを指定します。NULLをできます。

14個目の引数(ppTexture)は、生成されるテクスチャオブジェクトのインタフェースを格納する変数(IDirect3DTexture9*型またはLPDIRECT3DTEXTURE9型)のアドレスです。

－戻り値－
成功した場合はD3D_OK、それ以外の場合はエラーコードを返します。

```
// 画像ファイルからテクスチャを生成
IDirect3DTexture9*   pD3DTexture;          // テクスチャオブジェクトのインタフェース
D3DXCreateTextureFromFileEx(pD3DDevice, "Graphics\\Chara.tga",
                            D3DX_DEFAULT, D3DX_DEFAULT, 1, 0, D3DFMT_UNKNOWN,
                            D3DPOOL_MANAGED, D3DX_FILTER_NONE, D3DX_DEFAULT,
                            D3DCOLOR_RGBA(0, 0, 0, 255), NULL, NULL, &pD3DTexture);
```

課　　題

CDXGraphics9クラスに、スプライトを生成および解放する機能を追加しましょう。

(1)スプライトをカプセル化するクラスを作成します。

スプライトはID3DXSpriteインタフェース、イメージ(テクスチャ)はIDirect3DTexture9インタフェースをとおして操作を行います。これらのインタフェースは、生成に失敗した場合、NULLを保持します。ほとんどの場合、スプライトやテクスチャが無いということは、描画処理が行えないということになり、プログラムを終了させます。しかし、場合によっては、実行を続けたいこともあります。

この場合、インタフェースがNULLの場合とそうでない場合が考えられます。NULLは具体的なインスタンスを参照しておらず、そもそも読むことも書くこともできない領域を指しているので、NULLを介してメソッドを呼び出すと、Windowsは例外を発生させてプログラムを強制終了します。例外が発生しないようにするためには、以下のように、その都度インタフェースがNULLかどうかを調べなければなりません。

```
// pTextureは、Direct3DTexture9オブジェクトのインタフェース
if(pTexture != NULL) {
    // pTextureを使った処理
}
```

しかし、プログラムによってはこのようなチェックが非常に多くなってしまい、コードが煩雑になってしまう危険性があります。そこで、デザインパターンのNull Object(Null Device)パターンを適用します。
Null Objectパターンは、インタフェースやポインタがインスタンスを参照していないことを表現するのに、NULLを用いるのではなく、明示的に「何もしないオブジェクト」(Null Object)を使うようにする方法です。そのNull Objectは、呼び出される可能性のあるメソッドはすべて持ちますが、すべて「何もしない」処理として実装します。このようなNull Objectを導入すると、NULLチェックを省くことができ、簡潔なコードになります。

Null Objectパターンを適用したスプライトクラスのヘッダファイル(Sprite.hpp)を次のように作成しましょう。

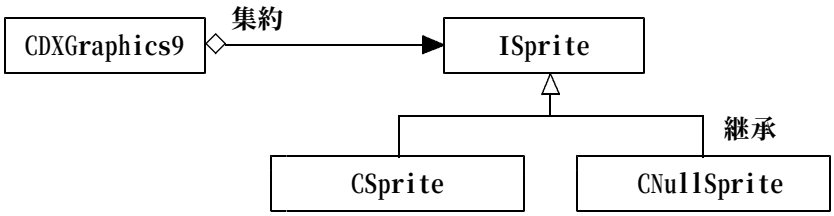
―Sprite.hpp―

```
/*
================================================================================
                        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】
    Sprite.hpp
                スプライトクラスヘッダ

  【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>

/*****************************************************************************************************/
/*                                      スプライトインタフェース定義                                 */
/*****************************************************************************************************/
class ISprite {
public:
    virtual ~ISprite() {}
};

/*****************************************************************************************************/
/*                                          スプライトクラス定義                                     */
/*****************************************************************************************************/
class CSprite : public ISprite {
public:

private:
};

/*****************************************************************************************************/
/*                                        NULLスプライトクラス定義                                   */
/*****************************************************************************************************/
class CNullSprite : public ISprite {
public:
    virtual ~CNullSprite() {}
};
```

ISpriteクラスは、CSpriteクラスとCNullSpriteクラスのインタフェースとして使用したいので、それらのクラスの基底クラスにし、publicな関数を純粋仮想関数で宣言します。CDXGraphics9クラスがスプライトクラスのオブジェクトを生成しようとしたとき、その成否によってCSpriteオブジェクトまたはCNullSpriteオブジェクトを生成(するように設計)します。アップキャストにより、ISpriteへのポインタには、そのどちらも代入することができます。ISpriteポインタをとおせば、そのインスタンスがCSpriteオブジェクトでもCNullSpriteオブジェクトでも、意識することなく操作することができるというわけです。

CSpriteクラスがスプライトをカプセル化するクラスになります。このクラスに、スプライト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。
CNullSpriteクラスがNull Object役になります。今後、CNullSpriteクラスには、ISprite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、次のようになります。



(２)スプライトクラスのソースファイル(Sprite.cpp)を以下のように作成しましょう。

－Sprite.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】
    Sprite.hpp
                スプライトクラス

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

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

#pragma once

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

(３)CSpriteクラスに、スプライト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。

```
    ID3DXSprite*        m_pD3DXSprite;
    IDirect3DTexture9*  m_pTexture;
```

スプライトを描画するためにID3DXSpriteインタフェース、イメージを管理するのにIDirect3DTexture9インタフェースが必要なので、これらをメンバ変数に加えておきます。

(４)CSpriteクラスのコンストラクタとデストラクタを作成します。以下のプロトタイプをpublicに追加しましょう。

```
    CSprite(ID3DXSprite* pD3DXSprite, IDirect3DTexture9* pD3DTexture);
    virtual ~CSprite();
```

コンストラクタでは、メンバ変数を初期化しますが、初期化に必要な情報のうち、外部から受け取らなければならない値を引数としています。コンストラクタの仕様は、以下のとおりです。

CSprite　　コンストラクタ

CSpriteオブジェクトを構築します。

試CSprite(ID3DXSprite* pD3DXSprite, IDirect3DTexture9* pD3DTexture);

pD3DXSprite	D3DXSpriteオブジェクトへのインタフェース
pD3DTexture	Direct3DTexture9オブジェクトへのインタフェース

(５)コンストラクタを実装します。以下のプログラムをソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          コンストラクタ                                           */
/*****************************************************************************************************/
CSprite::CSprite(ID3DXSprite* pD3DXSprite, IDirect3DTexture9* pD3DTexture)
    : m_pD3DXSprite(pD3DXSprite), m_pD3DTexture(pD3DTexture)
{
    assert(m_pD3DXSprite != NULL && m_pD3DTexture != NULL);

    m_pD3DXSprite->AddRef();        // 参照カウンタインクリメント
    m_pD3DTexture->AddRef();
}
```

コンストラクタでは、受け取ったインタフェースのAddRefメソッドを呼び出し、参照カウンタを増やしています。COMインタフェースを別の変数にコピーした場合、その期間が一時的でない場合は、このようにした方が安全です。
この例では、CSpriteオブジェクトが存在している間は、各インタフェースが参照しているオブジェクトが解放されなくなります(ただし、不正にReleaseメソッドが呼び出された場合はこの限りではありません)。もちろん、このままでは参照カウンタが増えたままになり、解放されなくなるので、CSpriteオブジェクトが破棄されるときに、Releaseメソッドを呼び出して参照カウンタを減らします。

(６)デストラクタを実装します。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。

```
/*****************************************************************************************************/
/*                                          デストラクタ                                             */
/*****************************************************************************************************/
CSprite::~CSprite()
{
    ※ここは各自考えましょう(m_pD3DTextureを解放します)
    ※ここは各自考えましょう(m_pD3DXSpriteを解放します)
}
```

デストラクタでは、コンストラクタで増やされたインタフェースの参照カウンタを減らし、正しく解放されるようにします。

(７)スプライトオブジェクトの解放漏れを防ぐため、すべてのスプライトオブジェクトを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<ISprite*>   m_SpriteList;      // スプライトリスト
```

(８)CDXGraphics9クラスに、空のテクスチャイメージを持つスプライトオブジェクトを生成するCreateSprite関数を追加します。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。

CreateSprite 生成

指定されたサイズおよびフォーマットのイメージを持つスプライトオブジェクトを生成し、そのインタフェースを返します。

```
訳ISprite* CreateSprite(const UINT inWidth, const UINT inHeight, const D3DFORMAT inFormat);
```

Return	生成されたスプライト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	スプライトイメージの幅。２の累乗に切り上げられます
inHeight	スプライトイメージの高さ。２の累乗に切り上げられます
inFormat	D3DFORMAT型のイメージのピクセルフォーマット

```
/*****************************************************************************************************/
/*                                          スプライト生成                                           */
/*****************************************************************************************************/
ISprite* CDXGraphics9::CreateSprite(const UINT inWidth, const UINT inHeight, const D3DFORMAT inFormat)
{
    ISprite*   pSprite;

    try {
        if(m_pD3DDevice == NULL)
            throw "DirectX Graphics未初期化";
```

```
        // テクスチャ生成
        IDirect3DTexture9*   pTexture = NULL;
        if(::????????????????????(m_pD3DDevice, inWidth, inHeight, 1, 0, inFormat,
                                   D3DPOOL_MANAGED, &pTexture) != D3D_OK)
            throw "ファイル読み込み失敗";

        pSprite = new CSprite(m_pD3DXSprite, pTexture);
        pTexture->Release();
    } catch(LPCTSTR ErrorString) {
        // エラーメッセージ生成
        TCHAR   msg[128];
        ::wsprintf(msg, "*** Error - %s(CDXGraphics9_CreateSrpite)\n", ErrorString);
        ::OutputDebugString(msg);

        // 例外が発生した場合は、NULLオブジェクトを生成
        pSprite = new ???????????();
    }

    m_SpriteList.push_back(pSprite);    // スプライトリストへ追加

    return pSprite;
}
```

(９)CDXGraphics9クラスに、ファイルからスプライトオブジェクトを生成するCreateSpriteFromFile関数を追加します。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。

CreateSpriteFromFile　生成

指定された画像ファイルとフォーマットからスプライトオブジェクトを生成し、そのインタフェースを返します。

```
書ISprite* CreateSpriteFromFile(LPCTSTR inFileName,
                                const D3DFORMAT inFormat = D3DFMT_UNKNOWN,
                                const D3DCOLOR inColorKey = 0);
```

Return	生成されたスプライト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
inFileName	読み込むファイルの名前
inFormat	D3DFORMAT型のイメージのピクセルフォーマット。省略するとファイルから取得します
inColorKey	D3DCOLOR型の透明色の色値を指定

```
/*****************************************************************************************************/
/*                                          スプライト生成                                           */
/*****************************************************************************************************/
ISprite* CDXGraphics9::CreateSpriteFromFile(LPCTSTR inFileName, const D3DFORMAT inFormat,
                                            const D3DCOLOR inColorKey)
{
    ISprite*   pSprite;

    try {
        if(m_pD3DDevice == NULL)
            throw "DirectX Graphics未初期化";

        // テクスチャ生成
        IDirect3DTexture9*   pTexture;
        if(::??????????????????????????????(m_pD3DDevice, inFileName, D3DX_DEFAULT, D3DX_DEFAULT, 1, 0,
                                             inFormat, D3DPOOL_MANAGED, D3DX_FILTER_NONE, D3DX_DEFAULT,
                                             inColorKey, NULL, NULL, &pTexture)
           != D3D_OK)
            throw "ファイル読み込み失敗";

        pSprite = new ???????(m_pD3DXSprite, pTexture);
        pTexture->Release();
    } catch(LPCTSTR ErrorString) {
        // エラーメッセージ生成
        TCHAR   msg[128];
        ::wsprintf(msg, "*** Error - %s(CDXGraphics9_CreateSrpiteFromFile)\n", ErrorString);
        ::OutputDebugString(msg);

        // 例外が発生した場合は、NULLオブジェクトを生成
```

```
        pSprite = new ???????????();
    }

    m_SpriteList.push_back(pSprite);    // スプライトリストへ追加

    return pSprite;
}
```

(10)CDXGraphics9クラスに、スプライトオブジェクトを解放するReleaseSprite関数を追加します。この関数の引数は、解放するスプライトのインタフェースへのポインタです。
以下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。

```
/*****************************************************************************************************/
/*                                          スプライト解放                                           */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseSprite(ISprite*& pSprite)
{
    m_SpriteList.??????(pSprite);       // スプライトリストから削除
    delete pSprite;                     // スプライトオブジェクト解放
    pSprite = NULL;                     // NULLを代入し、無効なポインタであることを明示する
}
```

(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スプライトオブジェクトを解放するReleaseAllSprites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。

```
/*****************************************************************************************************/
/*                                          全スプライト解放                                         */
/*****************************************************************************************************/
void CDXGraphics9::ReleaseAllSprites()
{
    for(std::list<ISprite*>::iterator it = m_SpriteList.?????(); it != m_SpriteList.???(); it++)
        delete *it;
    m_SpriteList.clear();
}
```

(12)CDXGraphics9クラスの解放時に、すべてのスプライト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。

```
    ReleaseAllSprites();        // 全スプライト解放
```