オブジェクト指向と
ゲームプログラミング

▶ DirectX Graphics 2D編 ― 第3回　スプライトの描画 ◀

シーンの開始と終了

スプライトは、3Dの機能を用いて描画(レンダリング)されます。そのため、スプライトの描画を行う前に、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。
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ます。

スプライト描画の開始と終了

スプライトは、通常の3Dオブジェクトと描画方法(レンダーステート)が異なります。スプライトが正しく描画されるように、レンダーステートを設定してから描画しなければなりません。ID3DXSprite::Beginメソッドは、スプライトが正しく描画できるように、いくつかのレンダーステートを設定します。
スプライトの描画が終わったら、ID3DXSprite::Endメソッドを呼び出し、レンダーステートを元に戻します。

ID3DXSprite::Beginメソッド

―説明―
Beginメソッドは、スプライトを描画できるようにデバイスを準備します。

―書式―
```
HRESULT Begin(DWORD Flags);
```

―パラメータ―
引数(Flags)は、スプライトのレンダリング方法です。0または以下のフラグを組み合わせます。

D3DXSPRITE_ALPHABLEND	アルファ ブレンディングを可能にします
D3DXSPRITE_SORT_DEPTH_FRONTTOBACK	描画の前にスプライトを前から後ろへ深度の順にソートします。不透明なスプライトを描画する際に使用します。
D3DXSPRITE_SORT_DEPTH_BACKTOFRONT	描画の前にスプライトを後ろから前へ深度の順にソートします。透明なスプライトを描画する際に使用します。
D3DXSPRITE_SORT_TEXTURE	描画の前にスプライトをテクスチャごとにソートします
D3DXSPRITE_BILLBOARD	各スプライトを、ビューアに向くように回転します
D3DXSPRITE_DONOTMODIFY_RENDERSTATE	レンダーステートを変更しません
D3DXSPRITE_DONOTSAVESTATE	レンダーステートの保存と復元を行いません
D3DXSPRITE_OBJECTSPACE	変換行列を変更しません

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

ID3DXSprite::Endメソッド

―説明―
Endメソッドは、Drawメソッドが呼び出されたスプライトをまとめてデバイスに送出し、デバイスの状態をID3DXSprite::Beginメソッドが呼び出される前の状態に復元します。

―書式―
```
HRESULT End();
```

―戻り値―
成功した場合はS_OK、それ以外の場合はD3DERR_INVALIDCALLを返します。

スプライトの描画

スプライトの描画は、ID3DXSprite::Drawメソッドで行います。

ID3DXSprite::Drawメソッド

－説明－

Drawメソッドは、描画スプライトのリストに、指定されたスプライトを追加します。

－書式－

```
HRESULT Draw(LPDIRECT3DTEXTURE9 pTexture, CONST RECT* pSrcRect,
             CONST D3DXVECTOR3* pCenter, CONST D3DXVECTOR3* pPosition, D3DCOLOR Color);
```

－パラメータ－

１つ目の引数(pTexture)は、スプライトのイメージが格納されたIDirect3DTexture9インターフェースへのポインタです。

２つ目の引数(pSrcRect)は、転送元の領域を格納したRECT構造体のアドレスです。NULLの場合、イメージ全体が描画されます。

３つ目の引数(pCenter)は、スプライトの中心を示すD3DXVECTOR3構造体のアドレスです。NULLの場合は、座標(0,0,0)となります。

４つ目の引数(pPosition)は、スプライトの描画位置を格納したD3DXVECTOR3構造体のアドレスです。NULLの場合は、座標(0,0,0)となります。

５つ目の引数(Color)は、スプライトと乗算するD3DCOLOR型の色値です。この値とイメージの色値が乗算されます。0xffffffffを指定すると、元の色が維持されます。

－戻り値－

成功した場合はS_OK、それ以外の場合はエラーコードを返します。

```
// ビューポートのクリア(pD3DDeviceは、Direct3DDevice9オブジェクト)
pD3DDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0, 1.0f, 0);

// シーン開始
pD3DDevice->BeginScene();

(…3Dオブジェクトの描画…)

// スプライト描画開始(pD3DXSpriteは、スプライトインタフェース)
pD3DXSprite->Begin(D3DXSPRITE_ALPHABLEND | D3DXSPRITE_SORT_DEPTH_FRONTTOBACK);

// スプライト描画(pD3DTextureは、イメージが格納されたDirect3DTexture9オブジェクト)
RECT          src = {0, 0, 120, 120};       // 転送元領域
D3DXVECTOR3   dest(320.0f, 240.0f, 0.0f);   // 転送先座標
pD3DXSprite->Draw(pD3DTexture, &src, NULL, &dest, 0xffffffff);

// スプライト描画終了
pD3DXSprite->End();

(…3Dオブジェクトの描画…)

// シーン終了
pD3DDevice->EndScene();
```

スプライト描画のしくみ

Direct3DXのスプライト描画は、以下のように行っています。

①IDirect3DDevice9::BeginScene
↓
②IDirect3DStateBlock9::Apply
↓
③IDirect3DDevice9::GetRenderState
↓
④IDirect3DDevice9::GetViewport
↓
⑤IDirect3DDevice9::GetTransform

↓
⑥IDirect3DTexture9::GetLevelDesc
↓
⑦IDirect3DDevice9::SetRenderState
↓
⑧IDirect3DDevice9::SetVertexDeclaration
↓
⑨IDirect3DDevice9::SetIndices
↓
⑩IDirect3DDevice9::SetStreamSource
↓
⑪IDirect3DDevice9::SetTexure
↓
⑫IDirect3DVertexBuffer9::Lock
↓
⑬IDirect3DVertexBuffer9::Unlock
↓
⑭IDirect3DDevice9::DrawIndexedPrimitive
↓
⑮IDirect3DDevice9::EndScene

スプライトは、ID3DXSprite::Drawメソッドが呼び出されたときに描画するのではなく、ID3DXSprite::Endメソッドが呼び出されたときにまとめて描画するようになっています。Drawメソッドは、描画用のスプライトデータをまとめておくリストに書き込むだけです。Endメソッドが呼び出されると、ｚ値などでソートした後、頂点バッファに書き込み、まとめて描画しています(⑫～⑭)。
スプライトを描画するための設定は、ID3DXSprite::Beginメソッドが呼ばれたとき行っています。このメソッドが呼び出されると､現在のレンダリングステートや行列などといった状態を保存し(③～⑥)、その後スプライトを描画するのに適した状態に設定します(⑦)。
描画速度の論理限界は、インデックス頂点バッファ(同じ頂点を共有する頂点バッファ)がもっとも高速といわれています。よってDirect3DXでは、DrawIndexedPrimitiveメソッドを用いて描画しているのです(⑭)。

課　題

CDXGraphics9クラスとスプライトクラスに、スプライトを描画する機能を追加しましょう。

(１)CDXGraphics9クラスに、スプライト描画の開始と終了を示す関数を追加します。以下のプロトタイプを適切な場所に追加しましょう。

```
bool BeginSprite(const DWORD inFlags);
void EndSprite();
```

(２)スプライトの描画を開始するBeginSprite関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          スプライト描画開始                                       */
/*****************************************************************************************************/
bool CDXGraphics9::BeginSprite(const DWORD inFlags)
{
#ifdef _DEBUG
    if(m_pD3DXSprite == NULL) {
        ::OutputDebugString("*** Error - スプライト未初期化(CDXGraphics9_BeginSprite)\n");
        return false;
    }
    if(m_pD3DXSprite->?????(inFlags) != D3D_OK) {
        ::OutputDebugString("*** Error - スプライト描画開始失敗(CDXGraphics9_BeginSprite)\n");
        return false;
    }
#else
    m_pD3DXSprite->?????(inFlags);
#endif
    return true;
}
```

(３)スプライトの描画を終了するEndSprite関数を実装します。以下のプログラムの足りない部分を補い、ソースファイルに追加しましょう。

```
/*****************************************************************************************************/
/*                                          スプライト描画終了                                         */
/*****************************************************************************************************/
void CDXGraphics9::EndSprite()
{
#ifdef _DEBUG
    if(m_pD3DXSprite == NULL) {
        ::OutputDebugString("*** Error - スプライト未初期化(CDXGraphics9_BeginSprite)\n");
        return;
    }
#endif
    m_pD3DXSprite->???();
}
```

(４)スプライトクラスに、スプライトを描画するDraw関数を作成します。以下のプロトタイプをISpriteクラス定義の適切な場所に追加しましょう。

```
    virtual void Draw(const RECT* pSrc, const D3DXVECTOR3& inPos, const float inAlpha) = 0;
```

Draw関数の仕様は、以下のとおりです。

Draw 描画

スプライトを描画します(実際は、描画スプライトリストへ登録するだけです)。

```
void Draw(const RECT* pSrc, const D3DXVECTOR3& inPos, const float inAlpha);
```

pSrc	転送元の領域を格納したRECT構造体のアドレス。NULLを指定すると、イメージ全体が描画される
inPos	スプライトの描画位置を格納したD3DXVECTOR3構造体のアドレス
inAlpha	透明度。範囲は0.0f(完全に透明)から1.0f(完全に不透明)

(５)CSpriteクラスにDraw関数を追加します。この関数のプロトタイプを(４)をもとに作成し、適切な場所に追加しましょう。

(６)Draw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。

```
/*****************************************************************************************************/
/*                                            描　画                                                 */
/*****************************************************************************************************/
void CSprite::Draw(const RECT* pSrc, const D3DXVECTOR3& inPos, const float inAlpha)
{
    // 色設定
    const D3DCOLOR   COLOR = ((DWORD)(inAlpha * 255.0f) << 24) | 0x00ffffff;

    // 変換行列設定
    D3DXMATRIX   mat;
    ::D3DXMatrixIdentity(&mat);
    m_pD3DXSprite->SetTransform(&mat);

    // 描画
    m_pD3DXSprite->Draw(※ここは各自考えましょう);
}
```

(７)CNullSpriteクラスに「何もしない」Draw関数を実装します。以下のプログラムを適切な場所に追加しましょう。

```
    virtual void Draw(const RECT* pSrc, const D3DXVECTOR3& inPos, const float inAlpha) {}
```

(８)構造体とフラグの省略名を作成します。以下のプログラムを適切な場所に追加しましょう。

```
/*****************************************************************************************************/
/*                                              型定義                                               */
/*****************************************************************************************************/
typedef D3DXVECTOR3  DXVEC3;

/*****************************************************************************************************/
/*                                          スプライト描画フラグ                                     */
/*****************************************************************************************************/
enum DXG_SPRITE {
    DXGSPR_ALPHA     = D3DXSPRITE_ALPHABLEND,
    DXGSPR_SORT_TEX  = D3DXSPRITE_SORT_TEXTURE,
    DXGSPR_SORT_FRBK = D3DXSPRITE_SORT_DEPTH_FRONTTOBACK,
    DXGSPR_SORT_BKFR = D3DXSPRITE_SORT_DEPTH_BACKTOFRONT
};
```

よく使う構造体やフラグが長い名前の場合、省略名を作成しておくと、入力するのが楽になります。

(９)正しく描画されるかを確認します。以下のような「背景ビットマップ」と「キャラクタービットマップ」を用意しましょう。

背景.BMP

キャラクター.BMP

(10)CTestSceneクラスを以下のように追加、変更しましょう。

・CTestSceneクラスのメンバに追加

```
ISprite*   m_pBG;          // 背景
ISprite*   m_pChara;       // キャラクター
int        m_CharaAnime;   // アニメーションカウンタ
```

・CTestScene::CTestScene関数を以下のように変更

```
CTestScene::CTestScene()
{
    // ここに、機能テストの初期化処理を記述します
    m_pBG    = DXGraphics().CreateSpriteFromFile("Graphics\\BG.bmp",    D3DFMT_UNKNOWN, 0);
    m_pChara = DXGraphics().CreateSpriteFromFile("Graphics\\Chara.bmp", D3DFMT_UNKNOWN,
                                                 D3DCOLOR_XRGB(0, 0, 0));

    m_CharaAnime = 0;

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

・CTestScene::~CTestScene関数を以下のように変更

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

・CTestScene::ActivateProc関数を以下のように変更

```
int CTestScene::ActiveProc()
{
    // ここに、機能テストのメイン処理を記述します
    // 内部処理
    // アニメーションカウンタ処理
    m_CharaAnime = (m_CharaAnime + 1) % 5;

    // 描画処理
    if(FPSTimer().IsSkip() == true)
        return 0;
```

```
        DXGraphics().Clear();

        DXGraphics().BeginScene();
        DXGraphics().BeginSprite(DXGSPR_ALPHA | DXGSPR_SORT_FRBK);

        RECT   src;

        // 背景描画
        ::SetRect(&src, 0, 0, 640, 480);
        m_pBG->Draw(&src, DXVEC3(0.0f, 0.0f, 1.0f), 1.0f);

        // キャラクター描画
        src.left   = 0;
        src.top    = 0;
        src.right  = 120;
        src.bottom = 120;
        m_pChara->Draw(&src, DXVEC3(0.0f, 0.0f, 0.5f), 1.0f);

        DXGraphics().EndSprite();
        DXGraphics().EndScene();

        // フレーム更新
        DXGraphics().UpdateFrame();

        return 0;
    }
```

(11)キャラクターをアニメーションさせましょう。