ESライブラリ＆＆
ゲームプログラミング

○　２Ｄ編 - 第１回　スプライトで画像の描画　○

スプライト

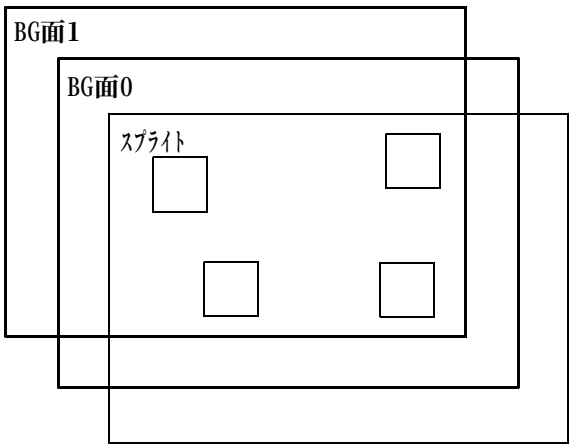
・DirectX9では、2Dグラフィクスの描画はすべてスプライト(またはポリゴン＋テクスチャ)で行う
※2D専用の機能があったDirectX7までは、2Dグラフィクスは画像として表示していた
・本来のスプライトはキャラクターをハードウェアで高速に表示する機能
・ほとんどの3D環境では、スプライトを「ポリゴン」＋「テクスチャ」で擬似的に再現
・3Dの機能を用いるので「拡大縮小」「回転」「半透明」「ｚバッファによる重なり」をサポート

概要

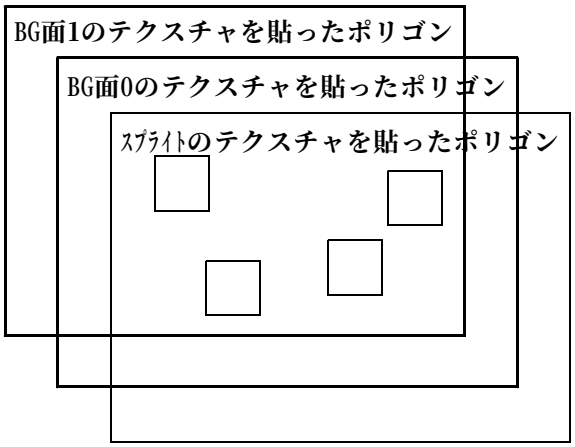
スプライトとは、キャラクターを高速に表示するための機能のことで、背景などほかの画像を壊すことなく独立して制御でき、透過色や重なり順序を指定することができます。

本来のスプライトは、ハードウェアで実現される機能で、背景画像(BG面)とは別に、多数の小さな画像を用意しておき、画面に合成して表示するものです。複数の背景やスプライトと重ね合わせて合成表示ができるようになっており、ほかのスプライトと重なったときに、どれを前面に表示するかが指定できるようになっています。CPUに負担をかけずにキャラクターを表示することができました。
現在では、CPUが高速化され、グラフィック機能も3Dに傾斜しているため、ハードウェアでスプライトをサポートしている機種はほとんどなく、ポリゴンとテクスチャを用いたり、SDKやライブラリでスプライトを擬似的に扱えるようにしています。

DirectX9でも、2Dグラフィクスの機能(スーパーファミコンやゲームボーイアドバンスのような旧来のスプライトおよびBG機能)は実装されていないため、2D表示はポリゴンを平面的に並べ、そこにテクスチャを貼り付けて擬似的に再現する疑似スプライトとなります。内部では面倒な作業を行っているように感じますが、3Dの機能を用いて描画(レンダリング)されるので「スケーリング」「回転」「アルファブレンディング」といった機能をGPUで高速に行うことができます。



▲スーパーファミコンやセガサターンはBG面とスプライトをハードウェアで合成して画面に表示



▲DirectX9ではすべて「ポリゴン」＋「テクスチャ」で表示。重なり順序は描画する順序やｚバッファで指定

スプライトの読み込み

・Direct3D + ESライブラリでスプライトを簡単に扱える
・読み込める画像の形式は .bmp .png .jpg .tga .dds .dib
・読み込むための関数は「GraphicsDevice.CreateSpriteFromFile関数」

概要

DirectX9に含まれるDirect3DXライブラリは、画像を読み込んでテクスチャにし、スプライトとして描画できる機能を備えています。さらにESライブラリにより、スプライトが簡単に扱えるようになっています。

スプライト描画は、DirectX9本来の流れでは、テクスチャ画像を読み込んで描画領域の頂点データを宣言し、テクスチャを貼り付けるように設定したポリゴンを描画することで行います。これらの作業をDirect3D + ESライブラリにより、いつくかの関数を実行するだけで行うことができます。

たとえば、画像を読み込み、スプライトとして扱えるようにするには、ESライブラリの「GraphicsDevice.CreateSpriteFromFile関数」にファイル名を指定するだけで行うことができます。この関数を実行すると、スプライトを制御するための変数が戻ってくるようになっているので、以下のようにSPRITE型の変数に保存しておきます(関数を実行しただけでは、スプライトがなくなってしまいます)。

```
SPRITE   haikei = GraphicsDevice.CreateSpriteFromFile( TEXT("bg000.bmp") );
```

スプライトの描画

・ESライブラリでのスプライトの描画は「SpriteBatch.Draw関数」で行う
・描画の前にSpriteBatch.Begin関数、描画後にはEnd関数を呼び出す必要がある
・Begin～End内で何度でもDrawを呼び出すことができる
・Direct3Dでは、スプライトの描画は3D機能が使われるので、BeginSceneとEndSceneメソッドの実行が必要
・ESライブラリでは、「GraphicsDevice.BeginScene関数」と「GraphicsDevice.EndScene関数」

概要

ESライブラリでのスプライトの描画は、SpriteBatch.Draw関数で行います。このときDraw関数の前にSpriteBatch.Begin関数を実行し、グラフィクスデバイスをスプライト(平面ポリゴン＋テクスチャ)の描画に適した状態にする必要があります。描画終了後にはSpriteBatch.End関数を実行し、描画が終了したことをグラフィクスデバイスに通知します。Begin～End関数内では、何度でもDraw関数の実行ができます。スプライトが実際に描画されるのはEnd関数を実行したときで、まとめてスプライトが描画(レンダリング)されるようになっています。

Draw関数は、スプライトを描画せず、スプライト描画リスト(スプライトバッチ)に登録するという作業をしています。End関数を実行したときに、リストに登録されたスプライトを指定順に並び替え、まとめて描画しています。

ESライブラリを使わず、DirectX9のみの場合も同じ流れになります。関数の名前が異なりますが、スプライトのBegin、Draw、Endという手順は同じです。

BeginSceneとEndScene

Direct3Dを用いて3Dグラフィクスを描画する場合、シーンの開始と終了をグラフィクスデバイスに通知する必要があります。シーンとは、映画などの「何々シーン」や「何々場面」と同じような、ゲーム内のワンシーン(世界観の一部)です。
3Dグラフィクスでは、3Dモデルを多数、空間内に置いて独自の世界観を表現します。この「モデルを空間内に置く」作業を始める宣言をするのが「シーンの開始」となります。作業終了を宣言するのが「シーンの終了」です。シーンの開始を宣言すると、グラフィクスデバイスが3Dグラフィクスを表現するのに適した状態になります。シーンの終了を宣言をすると、レンダリングされた仮想世界をディスプレイなどに表示できる状態にグラフィクスデバイスを設定します。

ESライブラリもDirect3Dを使用しているので、シーンの開始・終了宣言が必要となります。それぞれ「GraphicsDevice.BeginScene関数」「GraphicsDevice.EndScene関数」が対応しています。スプライトも3Dの機能を使っているので、これらの関数の実行が必要です。

コーディング例

```
// 変数の宣言(全体で使う変数の宣言は、基本的にヘッダーファイルで行います)
SPRITE   BG;     // 背景スプライト
                 ⋮

void CGameMain::LoadContent()
{
    // TODO: use this.Content to load your game content here
    // 背景読み込み
    BG = GraphicsDevice.CreateSpriteFromFile( TEXT("BG.jpg") );
                 ⋮

void CGameMain::Draw()
{
    GraphicsDevice.Clear(Color_CornflowerBlue);

    // TODO: Add your drawing code here
    GraphicsDevice.BeginScene();

    // スプライト描画
    SpriteBatch.Begin();                                  // スプライト描画開始
    SpriteBatch.Draw(BG, Vector3(0.0f, 0.0f, 1.0f));      // スプライト描画
    SpriteBatch.End();                                    // スプライト描画終了

    GraphicsDevice.EndScene();
}
```

課　　題

画像を読み込んで描画してみましょう。

(１)背景用の画像を準備しましょう。ファイルはプロジェクトの"Content"フォルダに保存してください。
ヒント：画面のサイズは1280×720の16:9、ハイビジョン解像度です

(２)スプライトを管理するための変数の宣言をします。以下のプログラムをヘッダーファイル"GameMain.h"の"private:"の下に追加しましょう。

```
SPRITE   BG;
```

(３)画像を読み込み、スプライトとして扱えるようにします。以下のプログラムをソースファイル"GameMain.cpp"の"LoadContent関数"に追加しましょう。

```
void CGameMain::LoadContent()
{
    // TODO: use this.Content to load your game content here
    // 背景読み込み
    BG = GraphicsDevice.CreateSpriteFromFile( TEXT("画像のファイル名") );
```

「GraphicsDevice.CreateSpriteFromFile関数」を実行すると、指定された画像ファイルが読み込まれ、テクスチャに変換し、スプライトとして扱えるように管理されます。変数BGには、その情報が格納されます。

(４)スプライトを表示します。以下のプログラムをソースファイル"GameMain.cpp"の"Draw関数"に追加しましょう。

```
void CGameMain::Draw()
{
    GraphicsDevice.Clear(Color_CornflowerBlue);

    // TODO: Add your drawing code here
    GraphicsDevice.BeginScene();

    // スプライト描画
    SpriteBatch.Begin();
    SpriteBatch.Draw(BG, Vector3(0.0f, 0.0f, 1.0f));
    SpriteBatch.End();

    GraphicsDevice.EndScene();
}
```

(５)プログラムをビルドし、実行してみましょう。