ESライブラリ＆＆
ゲームプログラミング

○　ピクセルシェーダー編 - 第５回　ＨＤＲとトーンマッピング　○

HDR

・HDRはHigh Dynamic Rangeの略。従来のレンジをLDR(Low Dynamic Range)と呼ぶ
・RGB各成分0.0～1.0(0～255)を超えた幅広い表現域でレンダリングを行う
・明るすぎて白く飛んでしまったり、暗すぎて黒くつぶれてしまう場面もレンダリングできる
・ただし、ディスプレイは16,777,216色しか表示できないので、トーンマッピングという技法でディスプレイに表示する
・トーンマッピングを行うと色味が変わるので注意

概要

HDRとトーンマッピングは、もともとは写真で用いられていた技術で、3Dグラフィクスに取り入れられたものです。

通常、ディスプレイで表現できる色は、RGB各成分に８ビットを使っているので計２４ビットの最大16,777,216色です。現実世界の色を再現するためには、各成分が１０の１２乗必要といわれているので、ディスプレイではまったく足りないことになります。そのため、ライトやマテリアルを多少、明るくすると、各成分が上限の1.0(255)になってしまい、レンダリング画像が白く飛んでしまいます。暗い部分も同様に、黒でつぶれてしまいます。よって、明るすぎるところや暗すぎるところは、微妙な表現ができません。そこで、現実世界の光を扱えるようにした技法のひとつがHDRです。

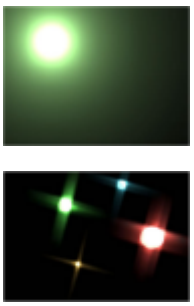
HDR(High Dynamic Range)

3DグラフィクスのHDRは、RGB各成分に浮動小数点を使い、幅広い表現域を再現できるようにしたものです。DirectX Graphicsでは、各成分を16ビットまたは32ビットの浮動小数点(Floting Point:FP)で扱うことができるようになっています。16ビットの場合、１ピクセルあたりRGBAのため16ビット×4＝64ビット、８バイト必要ですが、こうすると現実世界と同じぐらいの表現域を扱えます。

HDRを用いると、陰影のメリハリがつき、よりリアルになります。また、HDRを応用してブルーム、グレア、ゴーストといった光の効果を表現する技法も考え出されています。


▲通常のレンダリング


▲HDRレンダリング


▲上：ブルーム、下：グレア

トーンマッピング

HDRを使ってレンダリングしても、最終的にはディスプレイに表示しなければなりません。HDRをLDRのディスプレイにそのまま表示すると、上限1.0(255)を超えた色は白になってしまいます。実数である光を適正な明るさにし、各成分256段階にする処理がトーンマッピングです。

トーンマッピングはさまざまな方法が考え出されており、直線的なものからより自然に見える曲線、画像の解像度を落としながら適正輝度を算出する方法、数フレーム測定して平均輝度を算出する方法などがあります。基準となる輝度によって、同じ画像でも見え方が変わります。

▲どの輝度を基準にするかによって見え方がかわる

なお、DirectX Graphicsではトーンマッピング機能は提供していないため、ピクセルシェーダーでプログラムを作成しなければなりません。また、基本的にトーンマッピングを行うと色味が変わります。

ESライブラリでの対応

ESライブラリでは、GraphicsDevice.CreateHDRRenderTarget関数という、HDRフォーマットのレンダリングターゲットを生成する関数が用意されています。HDRであっても、通常どおりレンダリングを行えます。また、トーンマッピングはシェーダーで作成します。

課　題

HDRを使ってレンダリングし、トーンマッピングを使って結果を表示してみましょう。

(１)トーンマッピングをピクセルシェーダーで作成します。次のプログラムを"ToneMap.fx"として作成し、プロジェクトの"Shader"フォルダに保存しましょう。

－ToneMap.fx－

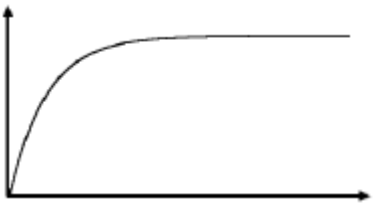
```
//-----------------------------------------------------------------------------
// File: ToneMap.fx
//
// The effect file for the Basic Tone Mapping sample.
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Global variables
//-----------------------------------------------------------------------------
sampler tex0 : register(s0);
float   exposure = 1.0;

//-----------------------------------------------------------------------------
// pass0 PixelShader Main Function
//-----------------------------------------------------------------------------
float4 PS_P0_Main(float2 UV : TEXCOORD0) : COLOR0
{
    float4   color = 1.0 - exp(-tex2D(tex0, UV) * exposure);
    return color;
}

//-----------------------------------------------------------------------------
// Techniques
//-----------------------------------------------------------------------------
technique ToneMap
{
    pass P0
    {
        VertexShader = NULL;
        PixelShader  = compile ps_2_0 PS_P0_Main();
    }
}
```

ピクセルシェーダーでは、HDRテクスチャから色値を取得し、0.0から1.0の値に変換しています。式は、



１ － 底がｅの指数（－色値 × 露光値）

というものです。右図のような非線形のトーンカーブが得られ、簡単な式にもかかわらず、それなりに自然に見えます。

(２)(１)で作成したエフェクトファイルを読み込むように、プログラムを変更しましょう。

(３)HDRレンダリングターゲット生成の前に、変数の宣言が必要です。ヘッダーファイルに以下のプログラムを追加しましょう。

```
RENDERTARGET    hdr;
```

(４)HDRレンダリングのターゲットを生成します。以下のプログラムを適切な場所に追加しましょう。

```
// オフスクリーンターゲット生成(HDRフォーマット)
hdr = GraphicsDevice.CreateHDRRenderTarget(1280, 720, DepthFormat_Unknown);
```

(５)「作業領域へのレンダリング→ピクセルシェーダーを用いて画面へ転送」を行うようにプログラムを変更します。CGameMain::Draw関数を以下のように変更しましょう。

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

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

    GraphicsDevice.SetRenderTarget(hdr);                    // HDRへターゲットを設定
    GraphicsDevice.Clear(Color_Black);                      // ターゲットのクリア

    ※ここで、レンダリングを行います※

    GraphicsDevice.RenderTargetToBackBuffer(HDR, effect);   // HDRターゲット→スクリーン

    GraphicsDevice.EndScene();
}
```

(６)プログラムを実行し、結果を確認しましょう。

(７)"ToneMap.fx"のexposure値を変更することにより、トーンマッピングの具合を変更できます。値を変更し、どのように変わるか確認しましょう。

(８)キー入力により、exposure値を増減できるようにしましょう。

ヒント：effect->SetFloat("exposure", 変数または値)
　　とすることにより、プログラム側からシェーダーのグローバル変数を変更できます

(９)式を変更し、実行結果を確認しましょう。

```
float4    color = tex2D(tex0, UV) * exp2(exposure);
```

２を底とする指数関数を用いた式です。

(10)式を変更し、実行結果を確認しましょう。

```
    float4   color = tex2D(tex0, UV);

    color *= exp2(exposure);
    color /= 1.0 + color;
```

トーンマッピングでよく使われるReinhardという計算式です。

(11)式を変更し、実行結果を確認しましょう。

```
    float4   color = tex2D(tex0, UV);

    color *= exp2(exposure);
    color  = max(color - 0.004, 0.0);
    color  = (color * (6.2 * color + 0.5)) / (color * (6.2 * color + 1.7) + 0.06);
```

映画フィルムのように見えるとのこと。