Chapter18-8 レンダリングターゲットの利用 ［応用］

シェーダーの利用の代表的なものにポストエフェクトが存在する。ポストエフェクトとはレンダリングをおこなったイメージに対して様々なエフェクト（効果）を適用するもの。
ポストエフェクトはシーン内に表示するジオメトリの数に作用されず、一定の負荷でシーンの印象に大きな変化を付けられるためリアルタイムレンダリングでは重要なエフェクトの一つとして使用される。代表的なポストエフェクトとして HDR レンダリングや被写界深度によるぼかしなどが存在する。

ポストエフェクトを作成するためには、まず描画結果をテクスチャとして再利用できる形で保存する必要がある。ここではそのためのレンダリングターゲットの利用を紹介する。

レンダリングターゲットはその名前の通りレンダリング（描画）を実行するためのターゲット（対象）のこと。
通常の描画では画面のバックバッファ（オフスクリーンサーフェース）に描画を実行し、フロントバッファ（画面）に転送することで表示している。
レンダリングターゲットをテクスチャにするためには、まずそのためのテクスチャの生成が必要になる。通常のテクスチャ読み込みの手順と同じように CTexture クラスのオブジェクトを宣言して CreateTarget 関数で画面にあわせたテクスチャを生成する。

Program18-8

```
//カメラ
CCamera                         gCamera;
//メッシュ
CMeshContainer                  gMesh;
//描画ターゲット
CTexture                        gTarget;

MofBool CGameApp::Initialize(void){
        //リソース配置ディレクトリの設定
        CUtilities::SetCurrentDirectory("Resource");

        //カメラ初期化
        gCamera.SetViewPort();
        gCamera.LookAt(Vector3(-2.0f,2.0f,-2.0f),Vector3(0,0,0),Vector3(0,1,0));
        gCamera.PerspectiveFov(MOF_ToRadian(60.0f),1024.0f / 768.0f,0.01f,1000.0f);
        gCamera.Update();
        CGraphicsUtilities::SetCamera(&gCamera);

        //メッシュの読み込み
        gMesh.Load("player.mom");

        //描画ターゲットを作成する
        MofU32 sw = g_pGraphics->GetTargetWidth();
        MofU32 sh = g_pGraphics->GetTargetHeight();
        gTarget.CreateTarget(sw, sh, PIXELFORMAT_R8G8B8A8_UNORM,
BUFFERACCESS_GPUREADWRITE);

        return TRUE;
}
```

CTexture クラスのメソッド

MofBool CreateTarget(const MofU32 w, const MofU32 h, const PixelFormat Format, const BufferAccess Usage);			
描画ターゲットとして使用するテクスチャを生成する			
引数	[in]	w	生成するテクスチャの幅
	[in]	h	生成するテクスチャの高さ
	[in]	format	生成するテクスチャのフォーマット
	[in]	Usage	生成するテクスチャのアクセスタイプ
戻り値	生成に成功した場合 TRUE,生成失敗した場合 FALSE		

生成したテクスチャはそのままレンダリングターゲットとして指定ができる。描画処理でレンダリングターゲットに指定する。指定後は通常の描画と同じ手順で描画をおこない、必要な描画が終わったら元のターゲットに戻す。こうして作成したテクスチャは以降、通常のテクスチャと同様に描画をできる。下記のプログラムでは作成したテクスチャを画面へのそのままの表示と SPACE キーで png ファイルとして保存している。

Program18-8

```
MofBool CGameApp::Render(void){
	//描画処理
	g_pGraphics->RenderStart();

	//元の描画ターゲットを取得する
	LPRenderTarget pold = g_pGraphics->GetRenderTarget();

	//作成したテクスチャを描画ターゲットとして設定する
	//深度バッファは元の情報をそのまま使用する
	g_pGraphics->SetRenderTarget(gTarget.GetRenderTarget(), g_pGraphics->GetDepthTarget());

	//画面のクリア
	g_pGraphics->ClearTarget(0.0f,0.0f,1.0f,1.0f,1.0f,0);

	//深度バッファ有効化
	g_pGraphics->SetDepthEnable(TRUE);

	//メッシュの描画
	CMatrix44 matWorld;
	gMesh.Render(matWorld);

	//描画ターゲットを元に戻す
	g_pGraphics->SetRenderTarget(pold, g_pGraphics->GetDepthTarget());
	g_pGraphics->SetDepthEnable(FALSE);

	//描画したターゲットをテクスチャとして画面に描画する
	gTarget.Render(0, 0);

	//SPACE キーで作成したレンダリングターゲットを保存する
	if (g_pInput->IsKeyPush(MOFKEY_SPACE))
	{
		gTarget.Save("ScreenImage.png");
	}

	//描画の終了
	g_pGraphics->RenderEnd();
	return TRUE;
}

MofBool CGameApp::Release(void){
	gMesh.Release();
	gTarget.Release();
	return TRUE;
}
```

IGraphics クラスのメソッド

MofBool SetRenderTarget(LPRenderTarget pRenderTarget, LPDepthTarget pDepthTarget);	
描画ターゲットを設定する	
引数	[in] pRenderTarget 描画時のカラーバッファの保存先 [in] pDepthTarget 描画時の深度バッファの保存先
戻り値	設定に成功した場合 TRUE, 設定失敗した場合 FALSE

LPRenderTarget GetRenderTarget(void);	
現在設定されている描画ターゲットを取得する	
引数	なし
戻り値	現在設定されている描画ターゲット

LPDepthTarget GetDepthTarget(void);	
現在設定されている深度ターゲットを取得する	
引数	なし
戻り値	現在設定されている深度ターゲット

CTexture クラスのメソッド

LPRenderTarget GetRenderTarget(void);	
テクスチャを対象とするレンダリングターゲットを取得する CreateTarget で生成したテクスチャ以外の場合は NULL になる	
引数	なし
戻り値	テクスチャを対象とする、描画ターゲット

MofBool Save(LPCMofChar pName);	
テクスチャをファイルとして出力する	
引数	[in] pName 出力ファイル名
戻り値	保存に成功した場合 TRUE, 保存失敗した場合 FALSE

Chapter18-9 色調の変更 [応用]

簡単なポストエフェクトとして、画像の色調を変化させるエフェクトがある。サンプルとしてモノクロの表示をおこなうエフェクトを紹介する。

Program18-9 Shader.hlsl

```
//! コンスタントバッファ
cbuffer cbSceneParam : register(b0)
{
	matrix	mtxView		: packoffset( c0 );
	matrix	mtxProj		: packoffset( c4 );
};

cbuffer cbMeshParam : register( b1 )
{
	float4cbvOffset		: packoffset( c0 );
	float4cbvSize		: packoffset( c1 );
```

```
	float4cbtOffset		: packoffset( c2 );
	float4cbtSize	: packoffset( c3 );
	float4cbColor	: packoffset( c4 );
	matrix	mtxWorld	: packoffset( c5 );
};

Texture2D txDiffuse : register( t0 );
SamplerState samLinear : register( s0 );

//! 頂点属性
struct InputVS
{
	float4pos		: POSITION;
	float2Tex		: TEXCOORD;
};
struct OutputVS
{
	float4pos		: SV_POSITION;
	float2Tex		: TEXCOORD0;
	float4color		: COLOR0;
};

//! 頂点シェーダ
OutputVS RenderVS( InputVS inVert )
{
	OutputVS	outVert;

	float3 pv = mul( float4(inVert.pos * cbvSize.xyz + cbvOffset.xyz, 1), mtxWorld );
	float4 Pos = mul(float4(pv, 1), mtxView);
	outVert.pos = mul(Pos, mtxProj);

	outVert.color = cbColor;

	outVert.Tex = inVert.Tex * cbtSize.xy + cbtOffset.xy;

	return outVert;
}

//! ピクセルシェーダ
const float3  monochrome = float3(0.298912, 0.586611, 0.114478);
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
	float4 ocol = txDiffuse.Sample(samLinear, inPixel.Tex) * inPixel.color;
	float grayColor = dot(ocol.rgb, monochrome);
	return float4(grayColor, grayColor, grayColor, 1);
}

technique11 TShader
{
	pass P0
	{
		SetVertexShader(CompileShader(vs_4_0, RenderVS()));
		SetGeometryShader(NULL);
		SetHullShader(NULL);
		SetDomainShader(NULL);
		SetPixelShader(CompileShader(ps_4_0, RenderPS()));
		SetComputeShader(NULL);
	}
}
```

シェーダー側での変更はピクセルシェーダーのみ、ピクセルシェーダで色を出力する際にモノクロ変換用のベクトルと掛け合わせた値を RGB 全ての色に利用しているだけである。このモノクロ変換用の定数は NTSC 系加重平均法に則った固定値として定義している。

ゲーム側ではまずシェーダーファイルを読み込む。ただし今回の描画はテクスチャの描画のみということでシェーダーの関連付け用のクラスを CShaderBind_SpriteBase に変更している。基本の利用シェーダーとの関連づけはジオメトリを描画するときは CShaderBind_3DPrimitiveBase を、テクスチャを描画するときは CShaderBind_SpriteBase を使えば良い。

シェーダーを読み込み関連付けを作成した後は、２D 描画用のスクリーン変換行列を設定している。現在の描画では２D での描画も実は２D 用の変換行列を設定しただけで３D 描画と違いはない。そこで必要な２D 変換用の行列を設定している。

Program18-9 GameApp.cpp

```
//カメラ
CCamera                         gCamera;
//メッシュ
CMeshContainer                  gMesh;
//描画ターゲット
CTexture                        gTarget;
//シェーダー
CShader                         gShader;
CShaderBind_SpriteBase          gShaderBind;

MofBool CGameApp::Initialize(void){
        //リソース配置ディレクトリの設定
        CUtilities::SetCurrentDirectory("Resource");

        //カメラ初期化
        gCamera.SetViewPort();
        gCamera.LookAt(Vector3(-2.0f,2.0f,-2.0f),Vector3(0,0,0),Vector3(0,1,0));
        gCamera.PerspectiveFov(MOF_ToRadian(60.0f),1024.0f / 768.0f,0.01f,1000.0f);
        gCamera.Update();
        CGraphicsUtilities::SetCamera(&gCamera);

        //メッシュの読み込み
        gMesh.Load("player.mom");

        //描画ターゲットを作成する
        MofU32 sw = g_pGraphics->GetTargetWidth();
        MofU32 sh = g_pGraphics->GetTargetHeight();
        gTarget.CreateTarget(sw, sh, PIXELFORMAT_R8G8B8A8_UNORM,
BUFFERACCESS_GPUREADWRITE);

        //シェーダーの読み込み
        gShader.Load("Shader.hlsl");
        gShaderBind.Create(&gShader);
        //シェーダーに２D 描画用の行列を設定する
        CGraphicsUtilities::SetScreenSize(sw, sh, &gShaderBind);
        return TRUE;
}
```

描画処理ではレンダリングターゲットを利用してテクスチャに画面を描画した後に、そのテクスチャを利用してモノクロシェーダーを適用して描画している。

Program18-9 GameApp.cpp

```
MofBool CGameApp::Render(void){
        //描画処理
        g_pGraphics->RenderStart();

        //元の描画ターゲットを取得する
        LPRenderTarget pold = g_pGraphics->GetRenderTarget();

        //作成したテクスチャを描画ターゲットとして設定する
        //深度バッファは元の情報をそのまま使用する
        g_pGraphics->SetRenderTarget(gTarget.GetRenderTarget(), g_pGraphics->GetDepthTarget());
```

```
	//画面のクリア
	g_pGraphics->ClearTarget(0.0f,0.0f,1.0f,1.0f,1.0f,0);

	//深度バッファ有効化
	g_pGraphics->SetDepthEnable(TRUE);

	//メッシュの描画
	CMatrix44 matWorld;
	gMesh.Render(matWorld);

	//描画ターゲットを元に戻す
	g_pGraphics->SetRenderTarget(pold, g_pGraphics->GetDepthTarget());
	g_pGraphics->SetDepthEnable(FALSE);

	//描画したターゲットをテクスチャとして画面に描画する
	CGraphicsUtilities::RenderTexture(0, 0, &gTarget, &gShader, &gShaderBind);

	//描画の終了
	g_pGraphics->RenderEnd();
	return TRUE;
}

MofBool CGameApp::Release(void){
	gMesh.Release();
	gTarget.Release();
	//シェーダーの解放
	gShaderBind.Release();
	gShader.Release();
	return TRUE;
}
```

モノクロ描画を少し変更すれば、セピア調の描画を実現できる。セピア描画のサンプルを紹介する。

Program18-9 Shader.hlsl

```
//! ピクセルシェーダ
const float3  monochrome = float3(0.298912, 0.586611, 0.114478);
const float3  sepia = float3(1.07, 0.74, 0.43);
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
	float4 ocol = txDiffuse.Sample(samLinear, inPixel.Tex) * inPixel.color;
	float grayColor = dot(ocol.rgb, monochrome);
	return float4(float3(grayColor, grayColor, grayColor) * sepia, 1.0);
}
```

元の状態	モノクロ	セピア調