Chapter18-17 ディスプレイスメントマッピング　[応用]

法線テクスチャを利用することで細かな凹凸を見せることができるが、あくまで陰影のみであり、水平にカメラを設置した場合には凹凸が一切ないことがわかってしまう。

そこで凹凸を実際につける方法として考えられた手法がディスプレイスメントマッピングである。ディスプレイすマッピングは法線テクスチャのようにモデルの凹凸を表す、法線マップや高さマップを頂点シェーダーで参照することで実際にジオメトリの表面の凹凸をつける手法。

Program18-17 Sample.hlsl

```
//! コンスタントバッファ
cbuffer cbSceneParam : register( b0 )
{
	float4		vecViewPos		: packoffset( c0 );
	matrix		mtxView			: packoffset( c1 );
	matrix		mtxProj			: packoffset( c5 );
};

cbuffer cbMeshParam : register( b1 )
{
	matrix		mtxWorld		: packoffset(c0);
	float4		colRevise		: packoffset(c4);
	float4		CoordsRevise		: packoffset(c5);
};

cbuffer cbMaterialParam : register( b2 )
{
	float4		matDiffuse		: packoffset( c0 );
	float4		matAmbient		: packoffset( c1 );
	float4		matSpeculer		: packoffset( c2 );
	float4		matEmissive		: packoffset( c3 );
	float		matPower		: packoffset( c4 );
};

cbuffer cbLightParam : register( b3 )
{
	float3		litDirection		: packoffset( c0 );
	float4		litDiffuse		: packoffset( c1 );
	float4		litAmbient		: packoffset( c2 );
	float4		litSpeculer		: packoffset( c3 );
};

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

//! 頂点属性
struct InputVS
{
	float4pos		: POSITION;
	float3normal		: NORMAL;
	float2Tex		: TEXCOORD;
	float4color		: COLOR0;
};
struct OutputVS
{
	float4pos		: SV_POSITION;
	float2Tex		: TEXCOORD0;
};

//新規のテクスチャオブジェクトの作成
Texture2D txNormal : register( t1 );

//! 頂点シェーダ
OutputVS RenderVS( InputVS inVert )
```

```
{
	OutputVS	outVert;

	matrix		mtxVP = mul( mtxView, mtxProj );

	//頂点シェーダーで法線マップを参照して法線方向に頂点座標を移動させる
	float4 Pos = inVert.pos;
	Pos.xyz += txNormal.SampleLevel(samLinear, inVert.Tex, 0).xyz * 0.05f;
	Pos = mul(Pos, mtxWorld);

	outVert.pos = mul( Pos , mtxVP );

	outVert.Tex = inVert.Tex;
	return outVert;
}

//! ピクセルシェーダ
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
	//法線をテクスチャから取得する
	//テクスチャの色では 0 ～ 1 なのでー 1 ～ 1 の範囲に変更する
	float4 n = normalize((2.0f * txNormal.Sample(samLinear, inPixel.Tex)) - 1.0f);

	//ハーフランバートライティング
	float l = dot(n, -litDirection);
	l = l * 0.5f + 0.5f;
	l *= l;

	//2 枚のテクスチャを線形補間で色を求める
	float3 diff = l * litDiffuse.xyz * matDiffuse.xyz;
	float4 LP = float4( saturate( diff ) , matDiffuse.w );
	return LP * txDiffuse.Sample(samLinear, inPixel.Tex);
}

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

しかしディスプレイスメントマッピングは頂点を実際に移動させる手法であるために、凹凸を正確に表現するためには細かなポリゴンの分割が必要である。そこで DirectX11 で追加されたテッセレーターを使用して分割を動的に変化させる手法が最新の環境では使われている。

Chapter18-18 ディスプレイスメントマッピングの活用　［応用］

ディスプレイスメントマッピングはその性質を利用してリアルタイムのジオメトリの変形にも利用される。ここでは高さマップを２Ｄの描画でリアルタイムに編集をおこない描画する手法を紹介する。

Program18-18 Sample.hlsl

```
//! コンスタントバッファ
cbuffer cbSceneParam : register( b0 )
{
	float4	vecViewPos		: packoffset( c0 );
	matrix	mtxView			: packoffset( c1 );
	matrix	mtxProj			: packoffset( c5 );
};

cbuffer cbMeshParam : register( b1 )
{
	matrix	mtxWorld		: packoffset(c0);
	float4	colRevise		: packoffset(c4);
	float4	CoordsRevise		: packoffset(c5);
};

cbuffer cbMaterialParam : register( b2 )
{
	float4	matDiffuse		: packoffset( c0 );
	float4	matAmbient		: packoffset( c1 );
	float4	matSpeculer		: packoffset( c2 );
	float4	matEmissive		: packoffset( c3 );
	float	matPower		: packoffset( c4 );
};

cbuffer cbLightParam : register( b3 )
{
	float3	litDirection		: packoffset( c0 );
	float4	litDiffuse		: packoffset( c1 );
	float4	litAmbient		: packoffset( c2 );
	float4	litSpeculer		: packoffset( c3 );
};

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

//! 頂点属性
struct InputVS
{
	float4pos		: POSITION;
	float3normal		: NORMAL;
	float2Tex		: TEXCOORD;
	float4color		: COLOR0;
};
struct OutputVS
{
	float4pos		: SV_POSITION;
	float2Tex		: TEXCOORD0;
};

//新規のテクスチャオブジェクトの作成
Texture2D txNormal : register( t1 );
Texture2D txHeight : register( t2 );

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

```
	OutputVS	outVert;

	matrix		mtxVP = mul( mtxView, mtxProj );

	//頂点シェーダーで法線マップを参照して法線方向に頂点座標を移動させる
	float4 Pos = inVert.pos;
	Pos.y += txHeight.SampleLevel(samLinear, inVert.Tex, 0).x * 0.05f;
	Pos = mul(Pos, mtxWorld);

	outVert.pos = mul( Pos , mtxVP );

	outVert.Tex = inVert.Tex;
	return outVert;
}

//! ピクセルシェーダ
float4 RenderPS( OutputVS inPixel ) : SV_TARGET
{
	//法線をテクスチャから取得する
	//テクスチャの色では 0 ～ 1 なのでー 1 ～ 1 の範囲に変更する
	float4 n = normalize((2.0f * txNormal.Sample(samLinear, inPixel.Tex)) - 1.0f);

	//ハーフランバートライティング
	float l = dot(n, -litDirection);
	l = l * 0.5f + 0.5f;
	l *= l;

	//2 枚のテクスチャを線形補間で色を求める
	float3 diff = l * litDiffuse.xyz * matDiffuse.xyz;
	float4 LP = float4( saturate( diff ) , matDiffuse.w );
	return LP * txDiffuse.Sample(samLinear, inPixel.Tex);
}

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

ゲーム側では描画ターゲットとしてテクスチャを生成して、そのテクスチャに高さ（凹凸）を描画する。作成した高さテクスチャをディスプレイスメントマッピングで頂点座標の移動にサンプリングするテクスチャとして利用するようにシェーダー側に設定を行う。

Program18-18 GameApp.cpp

```
//カメラ
CCamera			gCamera;
//ライト
CDirectionalLight		gLight;
//メッシュ
LPGeometry		pGeometry;
//テクスチャ
CTexture		gTexture;
//描画ターゲット
CTexture		gTarget;
//シェーダー
CShader			gShader;
```

```
CShaderBind_3DPrimitiveBase		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);

	//ライト初期化
	gLight.SetDirection(CVector3(0, -1, -1));
	CGraphicsUtilities::SetDirectionalLight(&gLight);

	//メッシュの読み込み
	pGeometry = CGraphicsUtilities::CreatePlaneGeometry(3, 3, 100, 100, TRUE, Vector3(0, 0, 0));
	//テクスチャの読み込み
	gTexture.Load("isi2_n.png");
	LPTexture pTex = new CTexture();
	pTex->Load("isi2.png");
	pGeometry->GetMaterial()->GetTextureArray()->AddLast(pTex);

	//シェーダーの読み込み
	gShader.Load("Shader.hlsl");
	gShaderBind.Create(&gShader);

	//テクスチャオブジェクトを設定
	gShaderBind.CreateShaderResource("txNormal");
	gShaderBind.CreateShaderResource("txHeight");

	//描画ターゲットを作成する
	MofU32 sw = 512;
	MofU32 sh = 512;
	gTarget.CreateTarget(sw, sh, PIXELFORMAT_R32_FLOAT,
BUFFERACCESS_GPUREADWRITE);
	return TRUE;
}
```

Program18-18 GameApp.cpp

```
MofBool CGameApp::Render(void){
	//描画処理
	g_pGraphics->RenderStart();
	//画面のクリア
	g_pGraphics->ClearTarget(0.0f,0.0f,1.0f,0.0f,1.0f,0);

	//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());

	if (g_pInput->IsMouseKeyHold(MOFMOUSE_LBUTTON))
	{
		MofFloat mx, my;
		g_pInput->GetMousePos(mx, my);
		g_pGraphics->SetBlending(BLEND_ADD);
		CGraphicsUtilities::RenderFillCircle(mx, my, 10.0f,
			MOF_ARGB(255, 255, 0, 0), MOF_ARGB(0, 0, 0, 0));
		g_pGraphics->SetBlending(BLEND_NORMAL);
```

```
	}

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

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

	//カメラを設定
	gShaderBind.SetCamera(&gCamera);

	//ライトを設定
	gShaderBind.SetDirectionalLight(&gLight);

	//テクスチャの設定
	gShaderBind.GetShaderResource(0)->SetResource(&gTexture);
	gShaderBind.GetShaderResource(1)->SetResource(&gTarget);

	//メッシュの描画
	CMatrix44 matWorld;
	matWorld.RotationX(MOF_ToRadian(90));
	if (g_pInput->IsKeyHold(MOFKEY_SPACE))
	{
		pGeometry->Render(matWorld, &gShader, &gShaderBind);
	}
	else
	{
		pGeometry->Render(matWorld);
	}


	g_pGraphics->SetDepthEnable(FALSE);

	//高さテクスチャを描画
	if (g_pInput->IsKeyHold(MOFKEY_F1))
	{
		gTarget.Render(0,0);
	}
	CGraphicsUtilities::RenderString(10, 10, MOF_COLOR_WHITE, "SPACE キーでシェーダー使用");

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