Chapter18-7 テクスチャバッファ　[応用]

シェーダーでテクスチャはテクスチャオブジェクトとして変数・定数とは別に独立して定義される。

DirectX11 シェーダーバージョン４．０以降ではテクスチャオブジェクトは Texture 型のオブジェクトとして下記の型が定義されている。

型	内容
Texture1D	1 次元テクスチャー
Texture1DArray	1 次元テクスチャー配列
Texture2D	2 次元テクスチャー
Texture2DArray	2 次元テクスチャー配列
Texture3D	3 次元テクスチャー
TextureCube	キューブ テクスチャー

テクスチャオブジェクトは通常の変数のように宣言できる。
　オブジェクトの型< type > 名前 : register(レジスタ番号)
type は省略可能でテクスチャからの取得する値の型を指定できる。
register（レジスタ番号)は省略可能で番号を t#でテクスチャオブジェクトがどのレジスタにテクスチャを展開するかを指定できる。

Program18-7 テクスチャオブジェクト作成例

```
//! テクスチャオブジェクト
Texture2D txDiffuse : register( t0 );
```

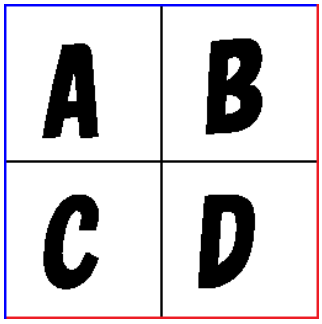
テクスチャオブジェクトを描画するためにはテクスチャオブジェクトからデータを取得する際にどのように取得するかを指定するサンプラーステートの作成が必要になる。
サンプラーステートはゲーム側から受け取るための作成のみの場合と、シェーダー内で中のステートを指定して作成する方法の 2 種類が存在する。

ゲーム側から受け取る場合は通常の変数のように宣言できる。
　SamplerState 名前 : register(レジスタ番号)
register（レジスタ番号)は省略可能で番号を s#でサンプラーステートをどのレジスタに作成するかを指定できる。

シェーダー内で作成する場合は宣言の後に｛　～　｝で作成するサンプラーステートのパラメーターを指定する。
　SamplerState 名前 : register(レジスタ番号) { ステート }
ステートはパラメーターからデフォルト値から変更したいものを設定する。
よく変更するパラメーターとよく設定する値は下記のようなデータがある。

パラメーター	内容
Filter	テクスチャからサンプリング（色を取得）する際に指定するテクスチャの UV（位置）は小数であり明確なピクセルではない。指定されたサンプリング位置からサンプリングする際に色を一つまたは周囲の複数のピクセルからどのように取得するかの指定 デフォルト値は MIN_MAG_MIP_POINT よく使用される代表的な値 ・MIN_MAG_MIP_POINT 　縮小、拡大、ミップレベルのサンプリングでポイント サンプリング（補間なし） ・MIN_MAG_MIP_LINEAR 　縮小、拡大、およびミップレベルのサンプリングに線形補間
AddressU AddressV AddressW	テクスチャのサンプリング位置（UV）を０～１以外の値で指定された場合の解決方法の指定 デフォルト値は CLAMP

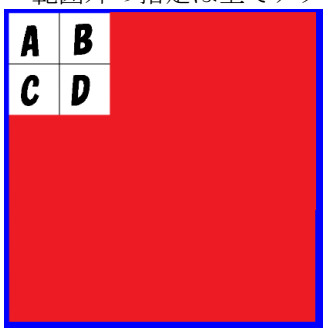
例：貼り付け画像



よく使用される代表的な値（画像の例は全て０～３を使用した場合）

・CLAMP

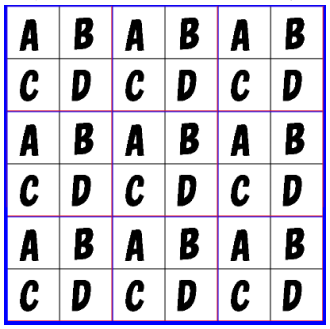
範囲外の指定は全てテクスチャの端の色を使用する



・WRAP

範囲外の指定は全て繰り返しでテクスチャの色を参照する。

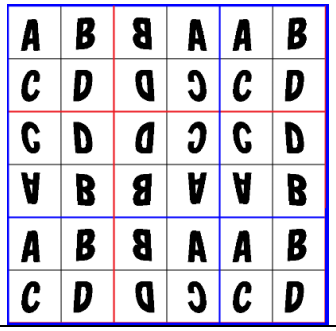
例えば uv を 0～２で指定した場合はテクスチャが 2 回繰り返される。



・MIRROR

範囲外の指定は全て繰り返しでテクスチャの色を参照する。

ただし奇数の値の場合テクスチャが反転する。



Program18-7 サンプラーステート作成例

```
//! サンプラーステート
SamplerState samLinear : register(s0)
{
```

```
        Filter = MIN_MAG_MIP_LINEAR;
        AddressU = MIRROR;
        AddressV = MIRROR;
};
```

二つのテクスチャオブジェクトを作成してブレンドするプログラムを紹介する。

Program18-7 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
{
        float4      pos             : POSITION;
        float3      normal          : NORMAL;
        float2      Tex             : TEXCOORD;
        float4      color           : COLOR0;
};
struct OutputVS
{
        float4      pos             : SV_POSITION;
        float2      Tex             : TEXCOORD0;
};

//! 新規のコンスタントバッファ作成
cbuffer cbGameParam : register(b4)
{
        float4      cbTime          : packoffset(c0);
};

//新規のテクスチャオブジェクトの作成
```

```
Texture2D txImage1 : register( t1 );
Texture2D txImage2 : register( t2 );

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

	matrix		mtxVP = mul( mtxView, mtxProj );
	float4 Pos = mul( inVert.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
{
	//フレンド率（U 位置 + sin(time))をそのままブレンド率とする）
	float bld = clamp(inPixel.Tex.x + sin(cbTime.x), 0, 1);
	//2 枚のテクスチャを線形補間で色を求める
	return lerp(txImage1.Sample(samLinear, inPixel.Tex),
		txImage2.Sample(samLinear, inPixel.Tex), bld);
}

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

ゲーム側でシェーダにテクスチャを設定する。

Program18-7 Sample.hlsl

```
//INCLUDE
#include	"GameApp.h"

//カメラ
CCamera				gCamera;
//メッシュ
LPGeometry			pGeometry;
//テクスチャ
CTexture			gTexture1;
CTexture			gTexture2;
//シェーダー
CShader				gShader;
CShaderBind_3DPrimitiveBase	gShaderBind;

//シェーダー側のコンスタントバッファと同様の構造体
struct cbGameParam
{
	Vector4	cbTime;
};
//時間
float				gTime = 0.0f;
```

```
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
	pGeometry = CGraphicsUtilities::CreatePlaneGeometry(3, 3, 1, 1, TRUE, Vector3(0, 0, 0));
	//テクスチャの読み込み
	gTexture1.Load("terrain1.jpg");
	gTexture2.Load("terrain2.jpg");

	//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("txImage1");
	gShaderBind.CreateShaderResource("txImage2");

	//シェーダーとのバッファの連携
	gShaderBind.CreateShaderBuffer("cbGameParam", sizeof(cbGameParam));
	return TRUE;
}

MofBool CGameApp::Update(void){
	//キーの更新
	g_pInput->RefreshKey();

	return TRUE;
}

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);

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

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

	//プログラム側で時間経過を実行
	gTime += CUtilities::GetFrameSecond();
	//シェーダーに送るバッファを作成
	cbGameParam sb;
	sb.cbTime.x = gTime;
	//シェーダーにバッファを送る
	gShaderBind.GetShaderBuffer(0)->SetBuffer(&sb);

	//テクスチャの設定
	gShaderBind.GetShaderResource(0)->SetResource(&gTexture1);
	gShaderBind.GetShaderResource(1)->SetResource(&gTexture2);

	//メッシュの描画
	CMatrix44 matWorld;
	pGeometry->Render(matWorld, &gShader, &gShaderBind);
```

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

MofBool CGameApp::Release(void){
	MOF_SAFE_DELETE(pGeometry);
	gTexture1.Release();
	gTexture2.Release();
	//シェーダーの解放
	gShaderBind.Release();
	gShader.Release();
	return TRUE;
}
```