

HLSL Pixel Shader

305890
2009년 봄학기¹
6/10/2009
박경신

픽셀 셰이더

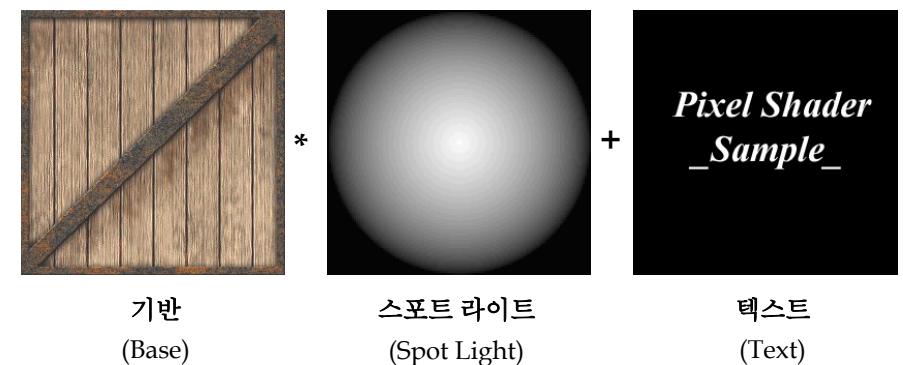
- ▣ 각 픽셀의 레스터라이즈 과정을 위해 그래픽 카드의 GPU에서 실행되는 프로그램
 - Direct3D는 소프트웨어적으로 픽셀 셰이더 기능을 에뮬레이트하지 않음
- ▣ 픽셀과 텍스처 좌표에 대한 직접적인 접근, 처리
 - 멀티 텍스처링, 픽셀 당 조명, 필드 깊이, 구름 시뮬레이션, 불 시뮬레이션, 복잡한 그림자 테크닉
- ▣ GPU가 지원하는 픽셀 셰이더의 버전 체크

```
// Step 2: Check for hardware vp.  
D3DCAPS9 caps;  
d3d9->GetDeviceCaps(D3DADAPTER_DEFAULT, deviceType, &caps);  
// If the device's supported version is less than version 2.0  
if( caps.PixelShaderVersion < D3DPS_VERSION(2,0) )  
    // Then pixel shader version 2.0 is not supported on this device
```

Overview

- ▣ 멀티 텍스처링 개념에 대한 기본적인 이해
- ▣ 픽셀 셰이더의 작성법과 생성법 및 사용법
- ▣ 픽셀 셰이더를 이용한 멀티 텍스처링의 구현 방법

픽셀 셰이더를 이용한 멀티 텍스처링



픽셀 셰이더를 이용한 멀티 텍스처링



멀티 텍스처링의 개요

- 텍스처들을 블렌딩 함

고정 기능
파이프라인



픽셀 셰이더

- 라이트 맵(light maps)이라는 특수한 텍스처 맵을 이용하면 얻게 되는 장점

- 조명이 미리 계산됨 → 전반적인 처리 속도 향상
 - 단, 물체와 조명이 모두 정적이어야 함
- Direct3D 조명 모델보다 훨씬 정확하고 정교한 광원 모델의 이용 가능
- 난반사 라이트 맵, 정반사 라이트 맵, 안개 맵, 디테일 맵

멀티 텍스처의 활성화

- 텍스처 지정 함수와 샘플러 상태 지정 함수

```
HRESULT SetTexture(  
    DWORD Stage, // specifies the texture stage index  
    IDirect3DBaseTexture9 *pTexture );  
  
HRESULT SetSamplerState(  
    DWORD Sampler, // specifies the texture stage index  
    D3DSAMPLERSTATETYPE Type,  
    DWORD Value  
);  
  
// Set first texture and corresponding sampler states  
Device->SetTexture( 0, BaseTex );  
Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR );  
  
// Set second texture and corresponding sampler states  
Device->SetTexture( 1, SpotLightTex );  
Device->SetSamplerState(1, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(1, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(1, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR );  
  
// Set third texture and corresponding sampler states  
Device->SetTexture( 2, StringTex );  
Device->SetSamplerState(2, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(2, D3DSAMP_MINFILTER, D3DTEXF_LINEAR );  
Device->SetSamplerState(2, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR );
```

멀티 텍스처 좌표

- 각 버텍스는 활성화된 텍스처 개수와 같은 수의 텍스처 좌표 집합을 가져야 함

- 최대 8개의 텍스처 좌표 지원

```
■ 예) struct MultiTexVertex {  
    MultiTexVertex(float x, float y, float z,  
                  float u0, float v0,  
                  float u1, float v1,  
                  float u2, float v2)  
    {  
        _x = x; _y = y; _z = z;  
        _u0 = u0; _v0 = v0;  
        _u1 = u1; _v1 = v1;  
        _u2 = u2; _v2 = v2;  
    }  
    float _x, _y, _z;  
    float _u0, _v0;  
    float _u1, _v1;  
    float _u2, _v2;  
    static const DWORD FVF;  
};  
const DWORD MultiTexVertex::FVF = D3DFVF_XYZ | D3DFVF_TEX3;
```

픽셀 셰이더의 입출력

□ 입력 - 픽셀의 컬러와 텍스처 좌표

```
■ 예) struct PS_INPUT
{
    vector c0      : COLOR0;
    vector c1      : COLOR1;
    float2 t0      : TEXCOORD0;
    float2 t1      : TEXCOORD1;
    float2 t2      : TEXCOORD2;
};
```

■ 픽셀 셰이더의 입력 ← 버텍스 셰이더의 출력

□ 출력 - 픽셀의 하나의 컬러

```
■ 예) struct VS_OUTPUT
{
    vector finalPixelColor : COLOR0;
};
```

픽셀 셰이더의 작성과 컴파일

□ HLSL을 이용해 픽셀 셰이더 프로그램 작성

■ ASCII 텍스트 파일 (텍스트 편집기 이용)

□ 픽셀 셰이더 컴파일

■ [D3DXCompileShaderFromFile](#) 함수 이용

→ 컴파일된 셰이더 코드를 포함하는 [ID3DXBuffer](#) 포인터 리턴

```
hr = D3DXCompileShaderFromFile(
    "ps_multitex.txt",
    0,
    0,
    "Main", // entry point function name
    "ps_1_1",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &MultiTexCT);
```

픽셀 셰이더의 이용 단계

픽셀 셰이더를 작성하고 컴파일

컴파일된 셰이더 코드에 기반한 픽셀 셰이더를
나타내는 IDirect3DPixelShader9 인터페이스를
생성

IDirect3DDevice9::SetPixelShader 메서드를
이용해 픽셀 셰이더 활성화

이용이 끝난 뒤, 픽셀 셰이더 제거

픽셀 셰이더 만들기

```
HRESULT IDirect3DDevice9::CreatePixelShader(
    const DWORD *pFunction,
    IDirect3DPixelShader9** ppShader
);
```

```
■ 예) IDirect3DPixelShader9* MultiTexPS = 0;
ID3DXConstantTable* MultiTexCT = 0;
ID3DXBuffer* shader = 0;
ID3DXBuffer* errorBuffer = 0;
hr = D3DXCompileShaderFromFile(
    "ps_multitex.txt",
    0,
    0,
    "Main",           // entry point function name
    "ps_1_1",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &MultiTexCT);
hr = Device->CreatePixelShader(
    (DWORD*)shader->GetBufferPointer(),
    &MultiTexPS);
```

픽셀 셰이더의 활성화와 제거

▣ 픽셀 셰이더의 활성화

```
HRESULT IDirect3DDevice9::SetPixelShader(  
    IDirect3DPixelShader9* pShader  
)
```

■ 예)
Device->BeginScene();

```
Device->SetPixelShader(MultiTexPS);  
Device->SetFVF(MultiTexVertex::FVF);  
Device->SetStreamSource(0, QuadVB, 0, sizeof(MultiTexVertex));
```

```
Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);
```

```
Device->EndScene();
```

▣ 이 용이 끝난 뒤에는 픽셀 셰이더를 제거

■ 예)
d3d::Release<IDirect3DPixelShader9*>(MultiTexPS);
d3d::Release<ID3DXConstantTable*>(MultiTexCT);

Sample: Multitexturing



HLSL 샘플러 객체

▣ (u,v) 텍스처 좌표로 인덱스 하려는 텍스처

- 텍스처와 샘플러 상태를 식별하는 객체

출력 컬러

Sampling

(u,v)
텍스처 좌표

+
인덱스 할 텍스처

■ 예)

```
sampler BaseTex;  
sampler SpotLightTex;  
sampler StringTex;
```

픽셀 셰이더 코드

```
IDirect3DTexture9* BaseTex = 0;  
D3DXHANDLE BaseTexHandle = 0;  
D3DXCONSTANT_DESC BaseTexDesc;
```

```
D3DXCreateTextureFromFile(Device, "crate.bmp", &BaseTex);
```

```
BaseTexHandle = MultiTexCT->GetConstantByName(0, "BaseTex");
```

```
MultiTexCT->GetConstantDesc(BaseTexHandle, &BaseTexDesc, &count);
```

```
Device->SetTexture(_BaseTexDesc.RegisterIndex, BaseTex);  
Device->SetSamplerState(BaseTexDesc.RegisterIndex,
```

D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);

```
Device->SetSamplerState(BaseTexDesc.RegisterIndex,
```

D3DSAMP_MINFILTER, D3DTEXF_LINEAR);

```
Device->SetSamplerState(BaseTexDesc.RegisterIndex,  
D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
```

Shader: ps_multitex.txt

```
sampler BaseTex;  
sampler SpotLightTex;  
sampler StringTex;  
  
struct PS_INPUT  
{  
    float2 base : TEXCOORD0;  
    float2 spotlight : TEXCOORD1;  
    float2 text : TEXCOORD2;  
};  
  
struct PS_OUTPUT  
{  
    vector diffuse : COLOR0;  
};  
  
PS_OUTPUT Main(PS_INPUT input){  
    PS_OUTPUT output = (PS_OUTPUT)0;  
  
    vector b = tex2D(BaseTex, input.base);  
    vector s = tex2D(SpotLightTex, input.spotlight);  
    vector t = tex2D(StringTex, input.text);  
  
    vector c = b * s + t;  
    c += 0.1f;  
    output.diffuse = c;  
    return output;  
}
```

Global Variables

```
IDirect3DPixelShader9* MultiTexPS = 0;
ID3DXConstantTable* MultiTexCT = 0;
IDirect3DVertexBuffer9* QuadVB = 0;
IDirect3DTexture9* BaseTex = 0;
IDirect3DTexture9* SpotLightTex = 0;
IDirect3DTexture9* StringTex = 0;
D3DXHANDLE BaseTexHandle = 0;
D3DXHANDLE SpotLightTexHandle = 0;
D3DXHANDLE StringTexHandle = 0;
D3DXCONSTANT_DESC BaseTexDesc;
D3DXCONSTANT_DESC SpotLightTexDesc;
D3DXCONSTANT_DESC StringTexDesc;
struct MultiTexVertex
{
    MultiTexVertex(float x, float y, float z,
                  float u0, float v0,
                  float u1, float v1,
                  float u2, float v2)
    {
        _x = x; _y = y; _z = z;
        _u0 = u0; _v0 = v0;
        _u1 = u1; _v1 = v1;
        _u2 = u2; _v2 = v2;
    }
    float _x, _y, _z;
    float _u0, _v0;
    float _u1, _v1;
    float _u2, _v2;
    static const DWORD FVF;
};
const DWORD MultiTexVertex::FVF = D3DFVF_XYZ | D3DFVF_TEX3;
```

Compiling and Creating a PS

```
ID3DXBuffer* shader = 0;
ID3DXBuffer* errorBuffer = 0;
hr = D3DXCompileShaderFromFile(
    "ps_multitex.txt",
    0,
    "Main", // entry point function name
    "ps_1",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &MultiTexCT);
if( errorBuffer ){
    ::MessageBox(0, (char*)errorBuffer->GetBufferPointer(), 0, 0);
    d3d::Release<ID3DXBuffer*>(errorBuffer);
}
if(FAILED(hr)){
    ::MessageBox(0, "D3DXCompileShaderFromFile() - FAILED", 0, 0);
    return false;
}
hr = Device->CreatePixelShader(
    (DWORD*)shader->GetBufferPointer(),
    &MultiTexPS);
if(FAILED(hr)){
    ::MessageBox(0, "CreateVertexShader - FAILED", 0, 0);
    return false;
}
d3d::Release<ID3DXBuffer*>(shader);
```

Creating a Vertex Buffer

```
bool Setup()
{
    HRESULT hr = 0;
    Device->CreateVertexBuffer(
        6 * sizeof(MultiTexVertex),
        D3DUSAGE_WRITEONLY,
        MultiTexVertex::FVF,
        D3DPOOL_MANAGED,
        &QuadVB,
        0);
    MultiTexVertex* v = 0;
    QuadVB->Lock(0, 0, (void**)&v, 0);
    v[0] = MultiTexVertex(-10.0f, -10.0f, 5.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    v[1] = MultiTexVertex(-10.0f, 10.0f, 5.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f);
    v[2] = MultiTexVertex( 10.0f, 10.0f, 5.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);
    v[3] = MultiTexVertex(-10.0f, -10.0f, 5.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f);
    v[4] = MultiTexVertex( 10.0f, 10.0f, 5.0f, 1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);
    v[5] = MultiTexVertex( 10.0f, -10.0f, 5.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f, 1.0f);
    QuadVB->Unlock();
```

Obtaining Handles and Initializing PS's Variables

```
D3DXCreateTextureFromFile(Device, "crate.bmp", &BaseTex);
D3DXCreateTextureFromFile(Device, "spotlight.bmp", &SpotLightTex);
D3DXCreateTextureFromFile(Device, "text.bmp", &StringTex);

D3DXMATRIX P;
D3DXMatrixPerspectiveFovLH(&P, D3DX_PI * 0.25f,
                           (float)Width / (float)Height, 1.0f, 1000.0f);
Device->SetTransform(D3DTS_PROJECTION, &P);

Device->SetRenderState(D3DRS_LIGHTING, false);

BaseTexHandle = MultiTexCT->GetConstantByName(0, "BaseTex");
SpotLightTexHandle = MultiTexCT->GetConstantByName(0, "SpotLightTex");
StringTexHandle = MultiTexCT->GetConstantByName(0, "StringTex");

UINT count;

MultiTexCT->GetConstantDesc(BaseTexHandle, &BaseTexDesc, &count);
MultiTexCT->GetConstantDesc(SpotLightTexHandle, &SpotLightTexDesc, &count);
MultiTexCT->GetConstantDesc(StringTexHandle, &StringTexDesc, &count);

MultiTexCT->SetDefaults(Device);

return true;
```

Display () (1)

```
bool Display(float deltaTime)
{
    if( Device )
    {
        // Update the scene: code snipped ...

        Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
                      0xffffffff, 1.0f, 0);
        Device->BeginScene();

        Device->SetPixelShader(MultiTexPS);
        Device->SetFVF(MultiTexVertex::FVF);
        Device->SetStreamSource(0, QuadVB, 0, sizeof(MultiTexVertex));

        Device->SetTexture( BaseTexDesc.RegisterIndex, BaseTex);
        Device->SetSamplerState(BaseTexDesc.RegisterIndex,
                               D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
        Device->SetSamplerState(BaseTexDesc.RegisterIndex,
                               D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
        Device->SetSamplerState(BaseTexDesc.RegisterIndex,
                               D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

        Device->SetTexture( SpotLightTexDesc.RegisterIndex,
                           SpotLightTex);
        Device->SetSamplerState(SpotLightTexDesc.RegisterIndex,
                               D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
        Device->SetSamplerState(SpotLightTexDesc.RegisterIndex,
                               D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
        Device->SetSamplerState(SpotLightTexDesc.RegisterIndex,
                               D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
    }
}
```

Display () (2)

```
Device->SetTexture( StringTexDesc.RegisterIndex, StringTex);
Device->SetSamplerState(StringTexDesc.RegisterIndex,
                       D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(StringTexDesc.RegisterIndex,
                       D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(StringTexDesc.RegisterIndex,
                       D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);

Device->EndScene();
Device->Present(0, 0, 0, 0);
}

return true;
}
```

Cleanup ()

```
void Cleanup()
{
    d3d::Release<IDirect3DVertexBuffer9*>(QuadVB);

    d3d::Release<IDirect3DTexture9*>(BaseTex);
    d3d::Release<IDirect3DTexture9*>(SpotLightTex);
    d3d::Release<IDirect3DTexture9*>(StringTex);

    d3d::Release<IDirect3DPixelShader9*>(MultiTexPS);
    d3d::Release<ID3DXConstantTable*>(MultiTexCT);
}
```

연습 문제 - Phong Shading



Shader: "phong.txt"

```
struct VS_INPUT {
    vector position : POSITION;
    vector normal  : NORMAL;
};

struct VS_OUTPUT {
    vector position : POSITION;
    vector light   : TEXCOORD0;
    vector normal  : TEXCOORD1;
    vector view    : TEXCOORD2;
};

struct PS_INPUT {
    vector light   : TEXCOORD0;
    vector normal  : TEXCOORD1;
    vector view    : TEXCOORD2;
};

struct PS_OUTPUT {
    vector diffuse : COLOR;
};
```

Shader: "phong.txt"

```
PS_OUTPUT PS_Main(PS_INPUT input)
{
    PS_OUTPUT output = (PS_OUTPUT)0;

    float d = saturate(dot(input.normal, input.light));

    input.light.w = 0.0f;
    input.normal.w = 0.0f;
    vector r = normalize(reflect(input.light, input.normal));
    float s = pow(saturate(dot(input.view, r)), 8);

    output.diffuse = (AmbientMtrl * AmbientLightIntensity)
        + (d * (DiffuseLightIntensity * DiffseMtrl))
        + (s * (SpecularLightIntensity * SpecularMtrl));

    return output;
}
```

Shader: "phong.txt"

```
VS_OUTPUT VS_Main(VS_INPUT input) {
    VS_OUTPUT output = (VS_OUTPUT)0;

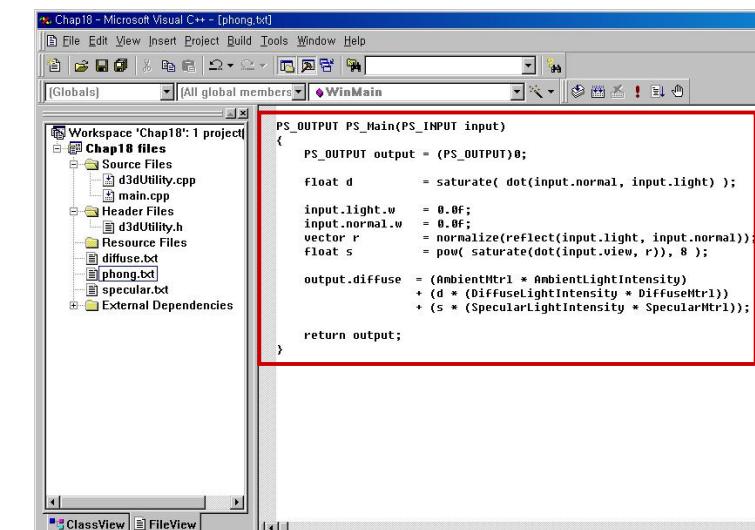
    output.position = mul(input.position, ViewProjMatrix);

    LightDirection.w = 0.0f;
    input.normal.w = 0.0f;
    output.light = mul(LightDirection, ViewMatrix);
    output.normal = mul(input.normal, ViewMatrix);

    output.view = normalize(mul(input.position, ViewMatrix));

    return output;
}
```

Shader: phong.txt (3)



Global Variables

Chap18 - Microsoft Visual C++ - [main.cpp]

```
#include "d3dUtility.h"

// // Globals
//

IDirect3DDevice9* Device = 0;
const int Width = 640;
const int Height = 480;

IDirect3DVertexShader9* DiffuseShader = 0;
ID3DXConstantTable* DiffuseConstTable = 0;

IDirect3DPixelShader9* PhongShader = 0;
ID3DXConstantTable* PhongConstTable = 0;

ID3DXMesh* Teapot = 0;
D3DXHANDLE ViewMatrixHandle = 0;
D3DXHANDLE ViewProjMatrixHandle = 0;
D3DXHANDLE LightDirHandle = 0;

D3DXHANDLE AmbientMtrlHandle = 0;
D3DXHANDLE DiffuseMtrlHandle = 0;
D3DXHANDLE SpecularMtrlHandle = 0;

D3DXMATRIX Proj;
```

Compiling a Vertex Shader

Chap18 - Microsoft Visual C++ - [main.cpp]

```
bool Setup()
{
    HRESULT hr = S_OK;

    // // Create geometry:
    //

    D3DXCreateTeapot(Device, &Teapot, 0);

    // // Compile shader
    //

    ID3DXBuffer* shader = 0;
    ID3DXBuffer* errorBuffer = 0;

    hr = D3DXCompileShaderFromFile(
        "diffuse.txt",
        "specular.txt",
        "phong.txt",
        0,
        0,
        // "Main", // entry point function name
        "VS_Main",
        "ps_1_1",
        D3DXSHADER_DEBUG,
        &shader,
        &errorBuffer,
        &DiffuseConstTable);
```

Creating a Vertex Shader

Chap18 - Microsoft Visual C++ - [main.cpp]

```
// // Create shader
//

hr = Device->CreateVertexShader(
    (DWORD*)shader->GetBufferPointer(),
    &DiffuseShader);

if(FAILED(hr))
{
    ::MessageBox(0, "CreateVertexShader - FAILED", 0, 0);
    return false;
}

// // Compile shader
//

hr = D3DXCompileShaderFromFile(
    "phong.txt",
    0,
    0,
    "PS_Main", // entry point function name
    "ps_2_0",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &PhongConstTable);

// output any error messages
```

Compiling a Pixel Shader

Chap18 - Microsoft Visual C++ - [main.cpp]

```
// // Create shader
//

hr = Device->CreateVertexShader(
    (DWORD*)shader->GetBufferPointer(),
    &DiffuseShader);

if(FAILED(hr))
{
    ::MessageBox(0, "CreateVertexShader - FAILED", 0, 0);
    return false;
}

// // Compile shader
//

hr = D3DXCompileShaderFromFile(
    "phong.txt",
    0,
    0,
    "PS_Main", // entry point function name
    "ps_2_0",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &PhongConstTable);

// output any error messages
```

Creating a Pixel Shader

This screenshot shows the Microsoft Visual Studio IDE with the project 'Chap18' open. The code editor displays the main.cpp file, which contains C++ code for creating a pixel shader. A red box highlights the section where the shader is created:

```
// Create shader
//
hr = Device->CreatePixelShader(
    (DWORD*)shader->GetBufferPointer(),
    &phongShader);

if(FAILED(hr))
{
    ::MessageBox(0, "CreatePixelShader - FAILED", 0, 0);
    return false;
}
```

Obtaining Handles and Initializing Variables

This screenshot shows the Microsoft Visual Studio IDE with the project 'Chap18' open. The code editor displays the main.cpp file, which contains C++ code for obtaining handles and initializing variables for the pixel shader. A red box highlights the section where handles are obtained:

```
// Get Handles
//
ViewMatrixHandle = DiffuseConstTable->GetConstantByName(0, "ViewMatrix");
ViewProjMatrixHandle = DiffuseConstTable->GetConstantByName(0, "ViewProjMatrix");
LightDirHandle = DiffuseConstTable->GetConstantByName(0, "LightDirection");

AmbientMtrlHandle = PhongConstTable->GetConstantByName(0, "AmbientMtr1");
DiffuseMtrlHandle = PhongConstTable->GetConstantByName(0, "DiffuseMtr1");
SpecularMtrlHandle = PhongConstTable->GetConstantByName(0, "SpecularMtr1");

// Set shader constants:
//

// Light direction:
D3DXVECTOR3 directionToLight(-0.5f, 0.5f, -0.5f, 0.0f);
DiffuseConstTable->SetVector(Device, LightDirHandle, &directionToLight);
DiffuseConstTable->SetDefaults(Device);

// Materials:
D3DXVECTOR4 ambientMtr1(0.0f, 0.0f, 1.0f, 1.0f);
D3DXVECTOR4 diffuseMtr1(0.0f, 0.0f, 1.0f, 1.0f);
D3DXVECTOR4 specularMtr1(1.0f, 1.0f, 1.0f, 1.0f);

PhongConstTable->SetVector(Device, AmbientMtr1Handle, &ambientMtr1);
PhongConstTable->SetVector(Device, DiffuseMtr1Handle, &diffuseMtr1);
PhongConstTable->SetVector(Device, SpecularMtr1Handle, &specularMtr1);
PhongConstTable->SetDefaults(Device);
```

Cleanup ()

This screenshot shows the Microsoft Visual Studio IDE with the project 'Chap18' open. The code editor displays the main.cpp file, which contains C++ code for the cleanup process. A red box highlights the section where resources are released:

```
// Compute projection matrix.
D3DXMatrixPerspectiveFovLH(
    &proj, D3DX_PI * 0.25f,
    (float)Width / (float)Height, 1.0f, 1000.0f);

return true;

void Cleanup()
{
    d3d->Release<ID3DXMesh*>(Teapot);

    d3d->Release<IDirect3DVertexShader9*>(DiffuseShader);
    d3d->Release<ID3DXConstantTable*>(DiffuseConstTable);

    d3d->Release<IDirect3DPixelShader9*>(PhongShader);
    d3d->Release<ID3DXConstantTable*>(PhongConstTable);
}

bool Display(float timeDelta)
{
    if( Device )
    {
        // Update the scene: Allow user to rotate around scene.
        //

        static float angle = (3.0f * D3DX_PI) / 2.0f;
```

Display ()

This screenshot shows the Microsoft Visual Studio IDE with the project 'Chap18' open. The code editor displays the main.cpp file, which contains C++ code for the display process. A red box highlights the section where rendering commands are issued:

```
D3DXVECTOR3 position( cosf(angle) * 7.0f, height, sinf(angle) * 7.0f );
D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
D3DXMATRIX up(0.0f, 1.0f, 0.0f);
D3DXMATRIX V;
D3DXMatrixLookAtLH(&V, &position, &target, &up);

DiffuseConstTable->SetMatrix(Device, ViewMatrixHandle, &V);

D3DXMATRIX ViewProj = V * Proj;
DiffuseConstTable->SetMatrix(Device, ViewProjMatrixHandle, &ViewProj);

//
// Render
//

Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
Device->BeginScene();

Device->SetVertexShader(DiffuseShader);
Device->SetPixelShader(PhongShader);

Teapot->DrawSubset(0);

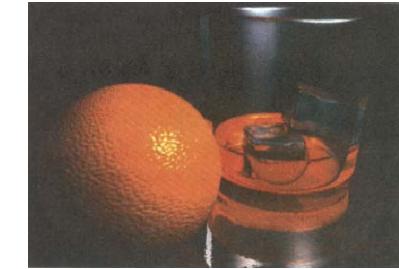
Device->EndScene();
Device->Present(0, 0, 0, 0);
}
return true;
```

Exercise – Bump Mapping



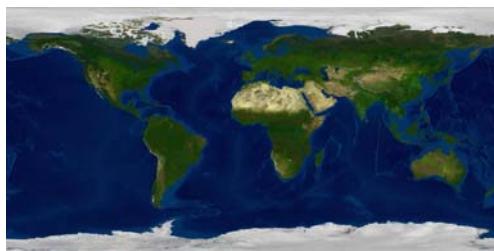
범프 매핑

- ▣ 표면이 렌더링 될 때, 법선 벡터에 잡음을 넣어 왜곡시키는 기법
→ 표면 색상에 작은 변동이 생겨 유통불통해 보임

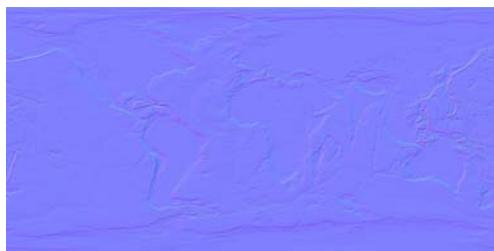


텍스처 맵

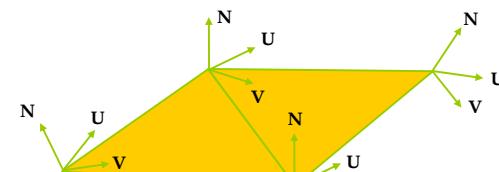
- ▣ 컬러 맵



- ▣ 범프 맵



탄젠트 좌표계



- ▣ 법선 벡터와 탄젠트 벡터의 계산

```
HRESULT D3DXComputeNormals(
    LPD3DXBASEMESH pMesh,
    CONST DWORD * pAdjacency );
```

```
HRESULT WINAPI D3DXComputeTangent(
    LPD3DXMESH Mesh,
    DWORD TexStageIndex,
    DWORD TangentIndex,
    DWORD BinormIndex,
    DWORD Wrap,
    const DWORD *pAdjacency );
```

Global Variables

Bump - Microsoft Visual C++ - [main.cpp]

```
File Edit View Insert Project Build Tools Window Help
[Globals] [All global members] Setup
Workspace 'Bump': 1 project(s)
Bump files
Source Files
d3dUtility.cpp
main.cpp
Header Files
d3dUtility.h
Resource Files
bump.txt
External Dependencies

IDirect3DDevice9* Device = 0;
const int Width = 640;
const int Height = 480;

IDirect3DVertexShader9* Ushader = 0;
ID3DXConstantTable* UConstTable = 0;

IDirect3DPixelShader9* PShader = 0;
ID3DXConstantTable* PConstTable = 0;

ID3DXMesh* Earth = 0;

IDirect3DTexture9* ColorTex = 0;
IDirect3DTexture9* BumpTex = 0;

D3DXHANDLE ColorTexHandle = 0;
D3DXHANDLE BumpTexHandle = 0;

D3DXCONSTANT_DESC ColorTexDesc;
D3DXCONSTANT_DESC BumpTexDesc;

D3DXHANDLE ViewMatrixHandle = 0;
D3DXHANDLE ViewProjMatrixHandle = 0;
D3DXHANDLE LightDirHandle = 0;

D3DXMATRIX Proj;
```

//

Loading a Mesh from X-File

Bump - Microsoft Visual C++ - [main.cpp]

```
File Edit View Insert Project Build Tools Window Help
[Globals] [All global members] Setup
Workspace 'Bump': 1 project(s)
Bump files
Source Files
d3dUtility.cpp
main.cpp
Header Files
d3dUtility.h
Resource Files
bump.txt
External Dependencies

bool Setup()
{
    HRESULT hr = S_OK;

    //-----
    // Create geometry: Load the XFile data.
    //

    ID3DXBuffer* adjBuffer = 0;
    LPD3DXMESH pMesh = NULL;

    hr = D3DXLoadMeshFromX(
        "sphere.x",
        D3DXMESH_MANAGED,
        Device,
        &adjBuffer,
        NULL,
        NULL,
        NULL,
        &pMesh);

    if(FAILED(hr))
    {
        ::MessageBox(0, "D3DXLoadMeshFromX() - FAILED", 0, 0);
        return false;
    }

    D3DUERTEXELEMENT9 decl[] =
```

Computing Normal & Tangent Vectors

Bump - Microsoft Visual C++ - [main.cpp]

```
File Edit View Insert Project Build Tools Window Help
[Globals] [All global members] Setup
Workspace 'Bump': 1 project(s)
Bump files
Source Files
d3dUtility.cpp
main.cpp
Header Files
d3dUtility.h
Resource Files
bump.txt
External Dependencies

D3DUERTEXELEMENT9 decl[] =
{
    {0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0},
    {0, 12, D3DDECLTYPE_FLOAT2, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0},
    {0, 20, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0},
    {0, 32, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TANGENT, 0},
    D3DDECL_END()
};

hr = pMesh->CloneMesh( D3DXMESH_MANAGED, decl, Device, &Earth );

// compute the normals
hr = D3DXComputeNormals( Earth, (DWORD*)adjBuffer->GetBufferPointer() );

if(FAILED(hr))
{
    ::MessageBox(0, "D3DXComputeNormals() - FAILED", 0, 0);
    return false;
}

// compute U (tangent)
hr = D3DXComputeTangent( Earth, 0, 0, 0, true, (DWORD*)adjBuffer->GetBufferPointer() );

if(FAILED(hr))
{
    ::MessageBox(0, "D3DXComputeTangent() - FAILED", 0, 0);
    return false;
}
```

Cleanup & Compiling VS

Bump - Microsoft Visual C++ - [main.cpp]

```
File Edit View Insert Project Build Tools Window Help
[Globals] [All global members] Setup
Workspace 'Bump': 1 project(s)
Bump files
Source Files
d3dUtility.cpp
main.cpp
Header Files
d3dUtility.h
Resource Files
bump.txt
External Dependencies

// cleanup
d3d::Release<ID3DXBuffer*>(adjBuffer);
d3d::Release<ID3DXMesh*>(pMesh);

//-----
// Compile shader
//

ID3DXBuffer* shader = 0;
ID3DXBuffer* errorBuffer = 0;

hr = D3DXCompileShaderFromFile(
    "bump.txt",
    0,
    0,
    "VS_Main", // entry point function name
    "vs_1_1",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &UConstTable);

// output any error messages
if( errorBuffer )
{
    ::MessageBox(0, (char*)errorBuffer->GetBufferPointer(), 0, 0);
    d3d::Release<ID3DXBuffer*>(errorBuffer);
}
```

Creating VS & Compiling PS

Bump - Microsoft Visual C++ - [main.cpp]

```
// Create shader
//
hr = Device->CreateVertexShader(
    (DWORD*)shader->GetBufferPointer(),
    &VShader);

if(FAILED(hr))
{
    ::MessageBox(0, "CreateVertexShader - FAILED", 0, 0);
    return false;
}

// Compile shader
//
hr = D3DXCompileShaderFromFile(
    "bump.txt",
    0,
    0,
    "PS_Main", // entry point function name
    "ps_2_0",
    D3DXSHADER_DEBUG,
    &shader,
    &errorBuffer,
    &pConstTable);
```

Creating PS & Getting Handles

Bump - Microsoft Visual C++ - [main.cpp]

```
hr = Device->CreatePixelShader(
    (DWORD*)shader->GetBufferPointer(),
    &PShader);

if(FAILED(hr))
{
    ::MessageBox(0, "CreatePixelShader - FAILED", 0, 0);
    return false;
}

d3d::Release<ID3DXBuffer*>(shader);

// Get Handles
//
ViewMatrixHandle = UConstTable->GetConstantByName(0, "ViewMatrix");
ViewProjMatrixHandle = UConstTable->GetConstantByName(0, "ViewProjMatrix");
LightDirHandle = UConstTable->GetConstantByName(0, "LightDirection");

// Set shader constants:
//
// Light direction:
D3DXVECTOR4 directionToLight(0.57F, 0.57F, 0.57F, 0.0F);
UConstTable->SetVector(Device, LightDirHandle, directionToLight);
UConstTable->SetDefaults(Device);
```

Getting Handles for Samplers

Bump - Microsoft Visual C++ - [main.cpp]

```
// Load textures
//
D3DXCreateTextureFromFile(Device, "earth.tga", &ColorTex);
D3DXCreateTextureFromFile(Device, "normal.tga", &BumpTex);

// Get Handles
//
ColorTexHandle = PConstTable->GetConstantByName(0, "ColorTex");
BumpTexHandle = PConstTable->GetConstantByName(0, "BumpTex");

// Set constant descriptions:
//
UINT count;
PConstTable->GetConstantDesc(ColorTexHandle, &ColorTexDesc, &count);
PConstTable->GetConstantDesc(BumpTexHandle, &BumpTexDesc, &count);
PConstTable->SetDefaults(Device);

// Disable lighting.
//
Device->SetRenderState(D3DRS_LIGHTING, false);

// Compute projection matrix.
```

Cleanup ()

Bump - Microsoft Visual C++ - [main.cpp]

```
void Cleanup()
{
    d3d::Release<ID3DXMesh*>(Earth);

    d3d::Release<IDirect3DTexture9*>(ColorTex);
    d3d::Release<IDirect3DTexture9*>(BumpTex);

    d3d::Release<IDirect3DVertexShader9*>(VShader);
    d3d::Release<ID3DXConstantTable*>(UConstTable);

    d3d::Release<IDirect3DPixelShader9*>(PShader);
    d3d::Release<ID3DXConstantTable*>(PConstTable);
}

bool Display(float deltaTime)
{
    if( Device )
    {
        //
        // Update the scene: Allow user to rotate around sc
        //

        static float angle = 0.0F;
        static float height = 3.0F;

        if( ::GetAsyncKeyState(VK_LEFT) & 0x8000F )
            angle += 0.5F * deltaTime;

        if( ::GetAsyncKeyState(VK_RIGHT) & 0x8000F )
```

Enabling VS and PS

```
D3DXVECTOR3 position( cosf(angle) * 3.0f, height, sinf(angle) * 3.0f );
D3DXVECTOR3 target(0.0f, 0.0f, 0.0f);
D3DXVECTOR3 up(0.0f, 1.0f, 0.0f);
D3DXMATRIX U;
D3DXMatrixLookAtLH(&U, &position, &target, &up);

UConstTable->SetMatrix(Device, ViewMatrixHandle, &U);

D3DXMATRIX ViewProj = U * Proj;
UConstTable->SetMatrix(Device, ViewProjMatrixHandle, &ViewProj);

// Render
//

Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);

Device->SetVertexShader(UShader);
Device->SetPixelShader(PShader);

// color tex
Device->SetTexture(    ColorTexDesc.RegisterIndex, ColorTex);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

// bump tex
Device->SetTexture(    BumpTexDesc.RegisterIndex, BumpTex);
```

Bump Mapping

```
// Render
//
Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, 0xffffffff, 1.0f, 0);
Device->BeginScene();

Device->SetVertexShader(UShader);
Device->SetPixelShader(PShader);

// color tex
Device->SetTexture(    ColorTexDesc.RegisterIndex, ColorTex);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(ColorTexDesc.RegisterIndex, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

// bump tex
Device->SetTexture(    BumpTexDesc.RegisterIndex, BumpTex);
Device->SetSamplerState(BumpTexDesc.RegisterIndex, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(BumpTexDesc.RegisterIndex, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
Device->SetSamplerState(BumpTexDesc.RegisterIndex, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

Earth->DrawSubset(0);

Device->EndScene();
Device->Present(0, 0, 0, 0);
}

return true;
```

Shader: bump.txt (1)

```
matrix ViewMatrix;
matrix ViewProjMatrix;
vector LightDirection;

sampler ColorTex;
sampler BumpTex;

struct VS_INPUT
{
    vector position : POSITION;
    Float2 tex : TEXCOORD0;
    vector normal : NORMAL;
    vector tangent : TANGENT;
};

struct VS_OUTPUT
{
    vector position : POSITION;
    Float2 tex : TEXCOORD0;
    Float3 light : TEXCOORD1;
    Float3 view : TEXCOORD2;
};

struct PS_INPUT
{
    Float2 tex : TEXCOORD0;
    Float3 light : TEXCOORD1;
    Float3 view : TEXCOORD2;
};
```

Shader: bump.txt (2)

```
struct PS_OUTPUT
{
    vector diffuse : COLOR;
};

US_OUTPUT US_Main(US_INPUT input)
{
    US_OUTPUT output= (US_OUTPUT)0;
    output.position = mul(input.position, ViewProjMatrix);
    output.tex = input.tex;

    float3 worldToTangent;
    input.tangent.w = 0.0f;
    input.normal.w = 0.0f;
    worldToTangent[0] = mul(input.tangent, ViewMatrix);
    worldToTangent[1] = mul(cross(input.normal, input.tangent), ViewMatrix);
    worldToTangent[2] = mul(input.normal, ViewMatrix);

    LightDirection.w = 0.0f;
    float3 x = mul(LightDirection, ViewMatrix);
    output.light = normalize(mul(x, worldToTangent));

    float3 y = normalize(mul(input.position, ViewMatrix));
    output.view = normalize(mul(y, worldToTangent));
    return output;
}
```

Shader: bump.txt (3)

The screenshot shows the Microsoft Visual Studio interface with the project 'Bump' open. The 'Display' tab is selected in the top navigation bar. The left pane shows the project structure under 'Workspace "Bump": 1 project(s)'. The 'Bump files' folder contains 'Source Files' (main.cpp), 'Header Files' (d3dUtility.h), 'Resource Files' (bump.txt), and 'External Dependencies'. The right pane displays the contents of the 'bump.txt' file, which is highlighted with a red border. The code is a pixel shader (PS) main function:

```
PS_OUTPUT PS_Main(PS_INPUT input)
{
    PS_OUTPUT output = (PS_OUTPUT)0;
    vector color = tex2D(ColorTex, input.tex);
    vector normal = 2 * (tex2D(BumpTex, input.tex) - 0.5);

    float d = saturate(dot(normal, input.light));
    float3 r = normalize(reflect(normal, input.light));
    float s = min(pow(saturate(dot(input.view, r)), 8), color.w);

    output.diffuse = 0.2 * color + d * color + s;
    return output;
}
```