HLSL Pixel Shader

305890
2009년 봄학기
6/10/2009
박경신

픽셀 셰이더

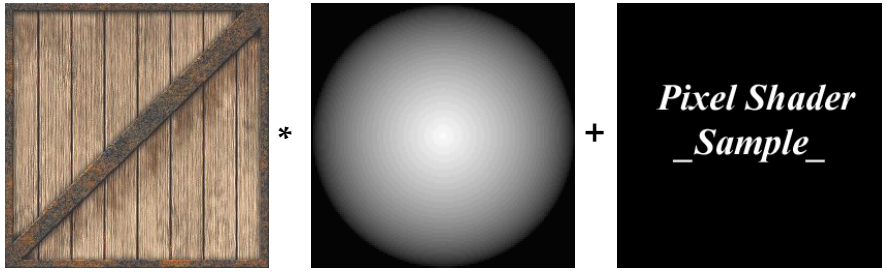
- 각 픽셀의 래스터라이즈 과정을 위해 그래픽 카드의 GPU에서 실행되는 프로그램
  - Direct3D는 소프트웨어적으로 픽셀 셰이더 기능을 에뮬레이트 하지 않음
- 픽셀과 텍스처 좌표에 대한 직접적인 접근, 처리
  - 멀티 텍스처링, 픽셀 당 조명, 필드 깊이, 구름 시뮬레이션, 불 시뮬레이션, 복잡한 그림자 테크닉
- GPU가 지원하는 픽셀 셰이더의 버전 체크

```
// 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



기반
(Base)

스포트 라이트
(Spot Light)

텍스트
(Text)

픽셀 셰이더를 이용한 멀티 텍스처링

기반
(Base)

텍스트
(Text)

멀티 텍스처링 결과

멀티 텍스처링의 개요

- 텍스처들을 블렌딩함

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 정적이여야 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 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 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 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)

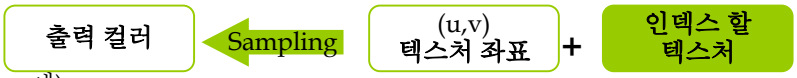
```
Device->BeginScene();

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

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

Device->EndScene();
```

- 이용이 끝난 뒤에는 픽셀 셰이더를 제거
  - 예)

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

HLSL 샘플러 객체

- (u,v) 텍스처 좌표로 인덱스 하려는 텍스처
  - 텍스처와 샘플러 상태를 식별하는 객체



  - 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);
```

Sample: Multitexturing

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;
```

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();
```

Compiling and Creating a PS

```
    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);

    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);
```

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 timeDelta)
{
    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);
```

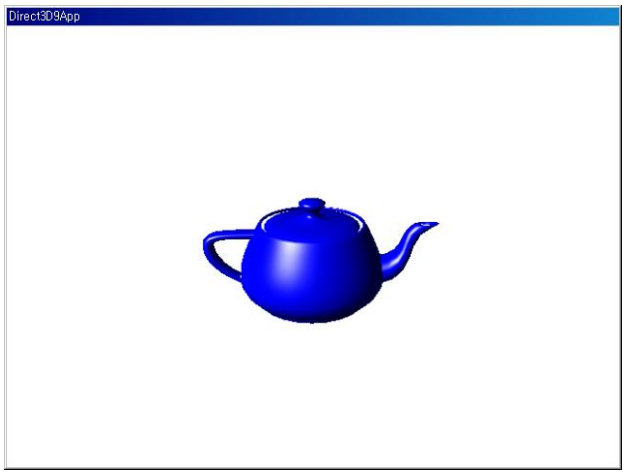
Display ( ) (2)

```
        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;
};
```

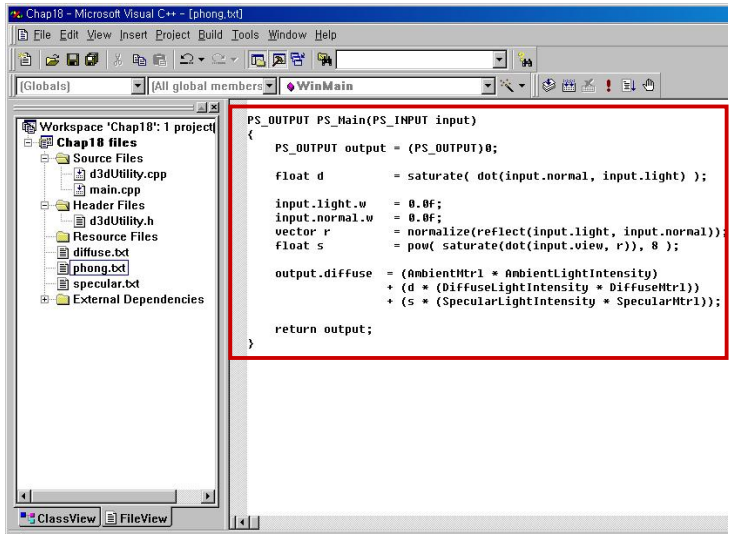
Shader: "phong.txt"

```
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;
}
```

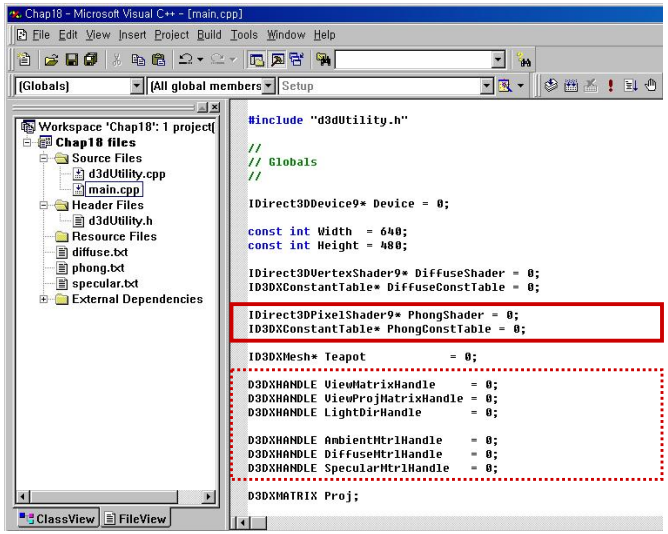
Shader: "phong.txt"

```
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;
}
```

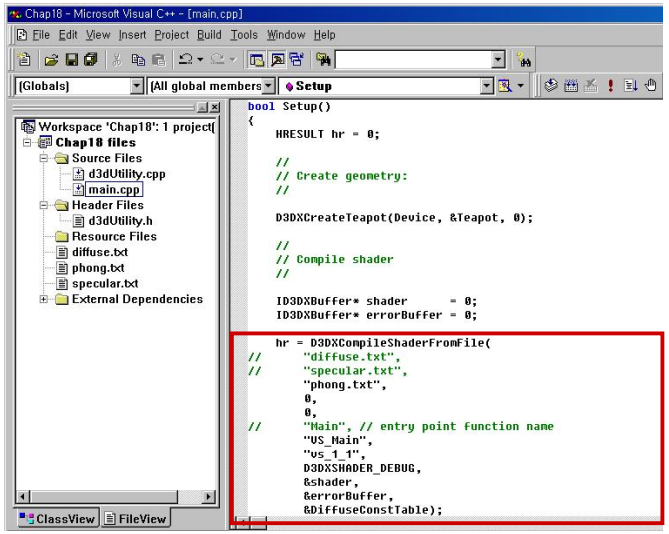
Shader: phong.txt (3)



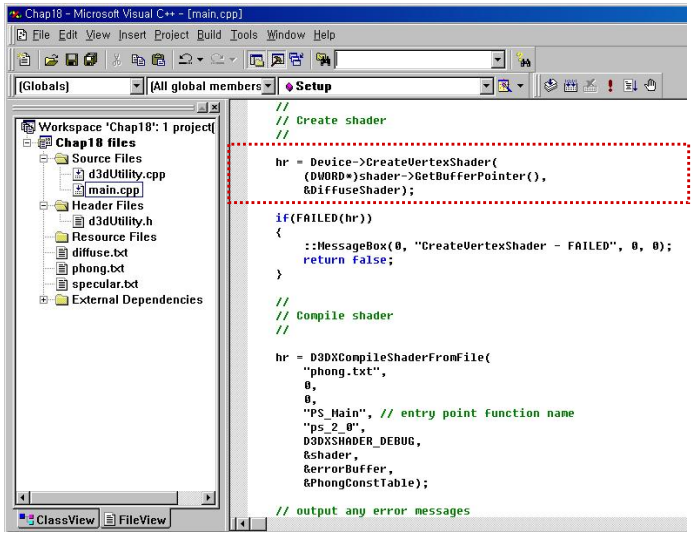
Global Variables



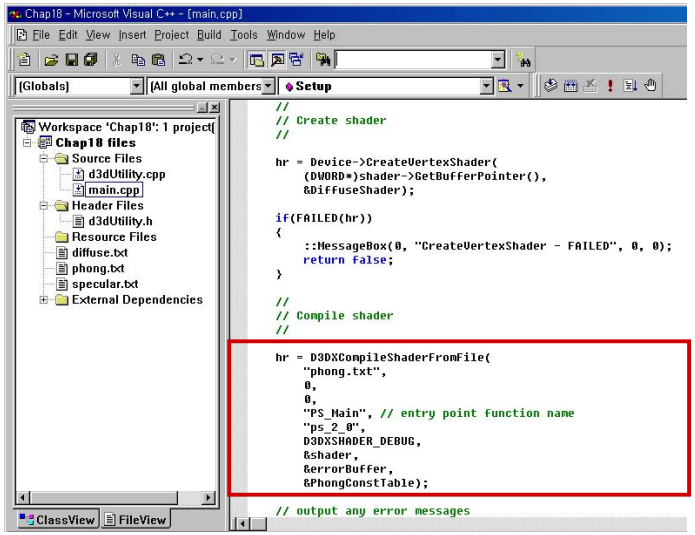
Compiling a Vertex Shader



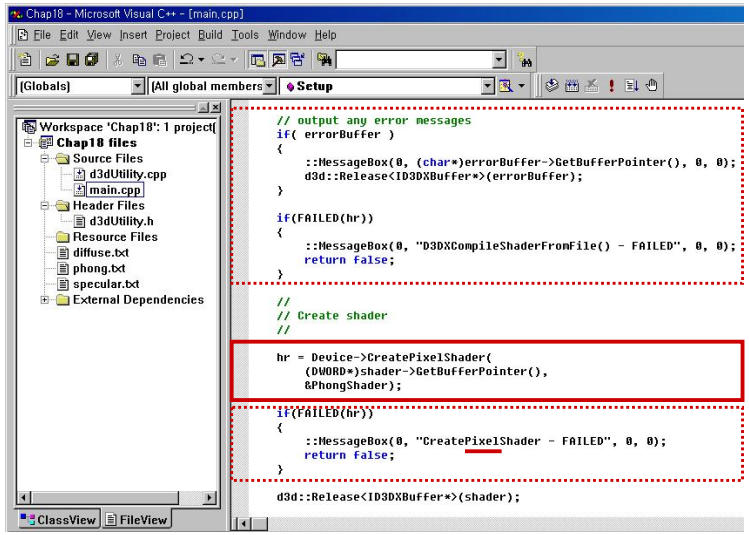
Creating a Vertex Shader



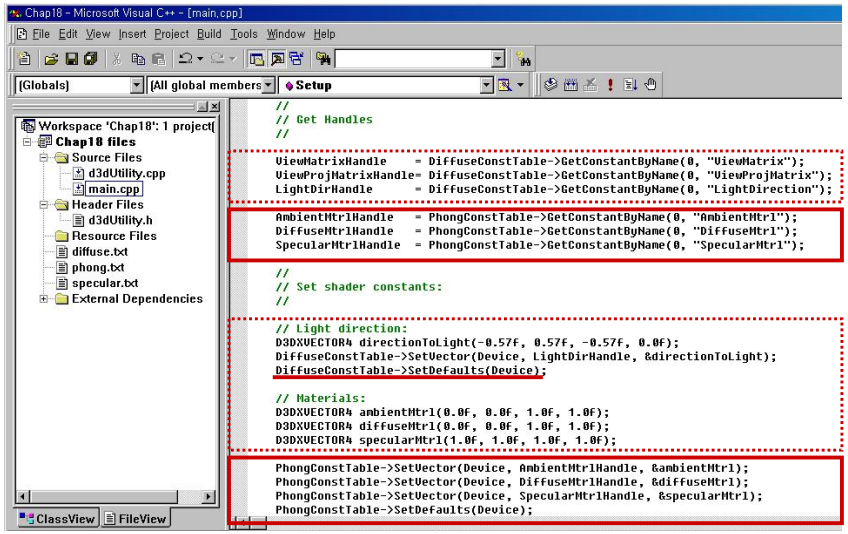
Compiling a Pixel Shader



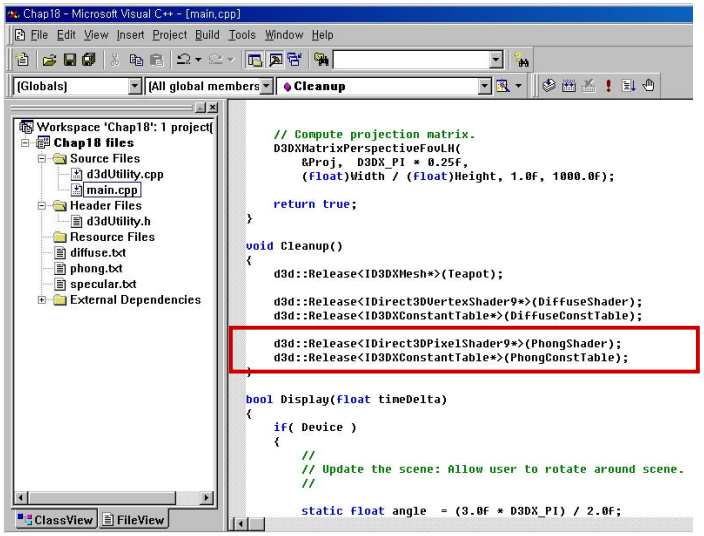
Creating a Pixel Shader



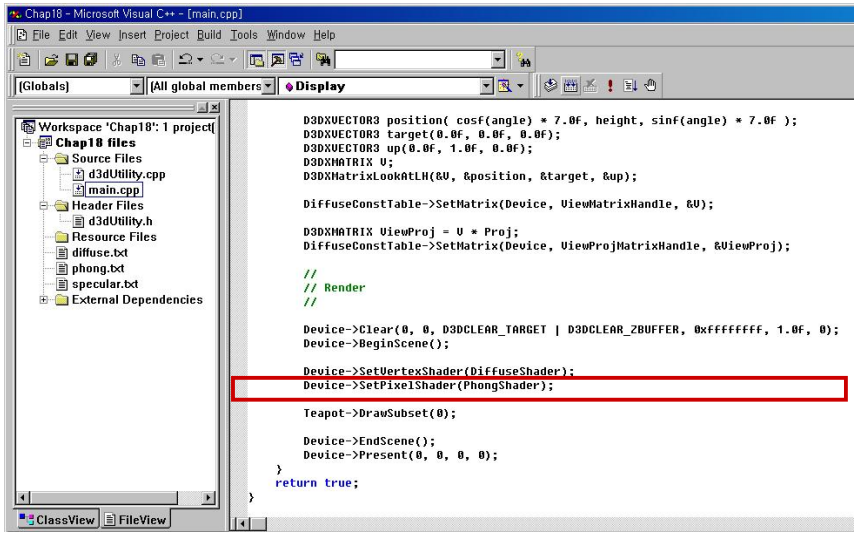
Obtaining Handles and Initializing Variables



Cleanup ( )



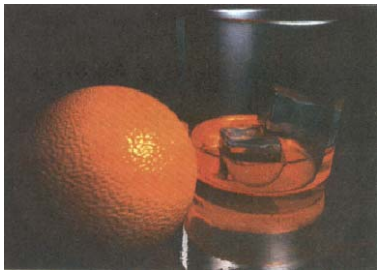
Display ( )



Exercise – Bump Mapping

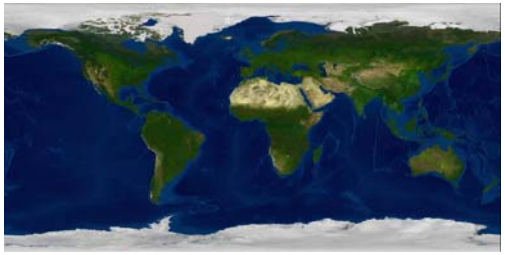
범프 매핑

- 표면이 렌더링 될 때, 법선 벡터에 잡음을 넣어 왜곡시키는 기법
  - ➔ 표면 색상에 작은 변동이 생겨 울퉁불퉁해 보임

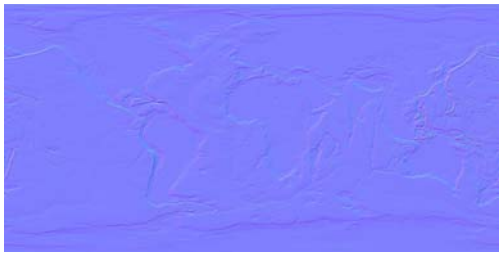


텍스처 맵

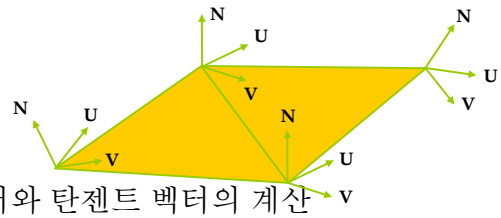
- 컬러 맵



- 범프 맵



탄젠트 좌표계

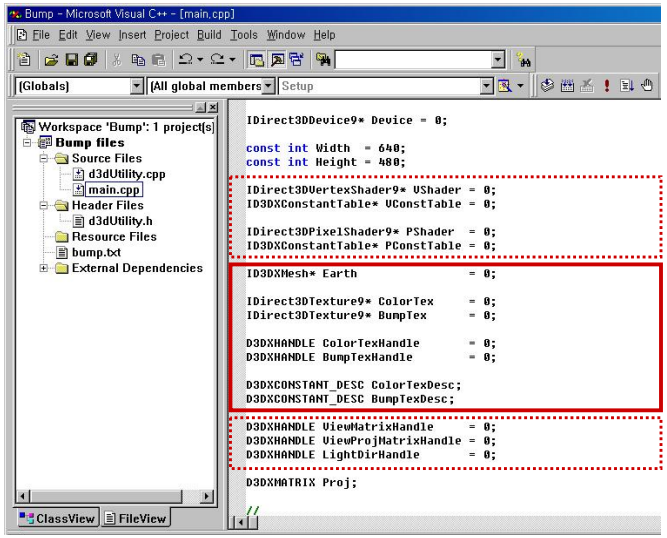


- 법선 벡터와 탄젠트 벡터의 계산

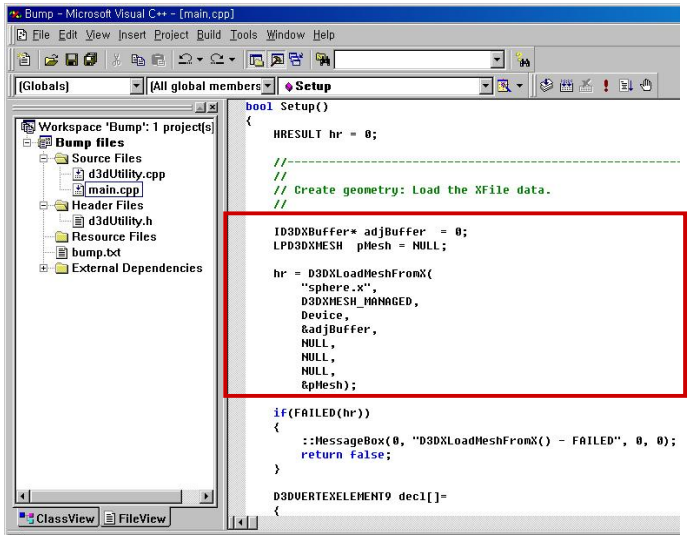
```
HRESULT D3DXComputeNormals(
            LPD3DXBASEMESH pMesh,
            CONST DWORD * pAdjacency );
```

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

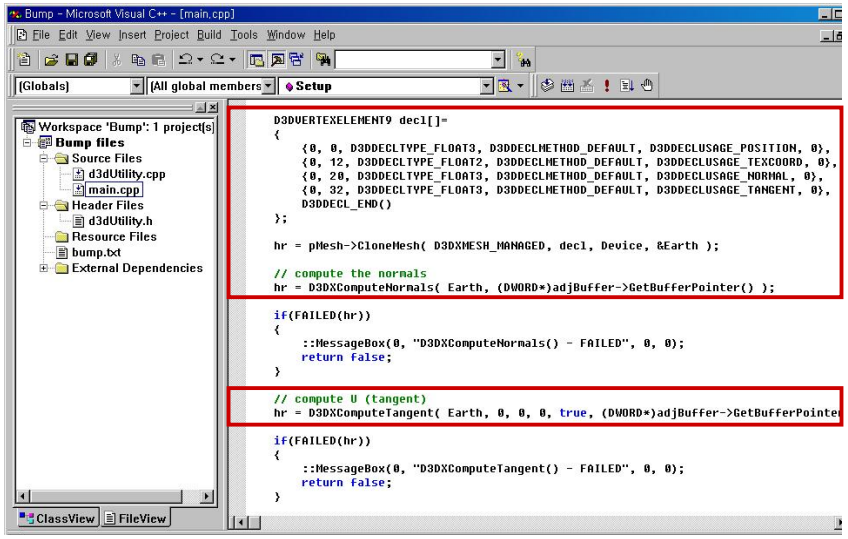
Global Variables



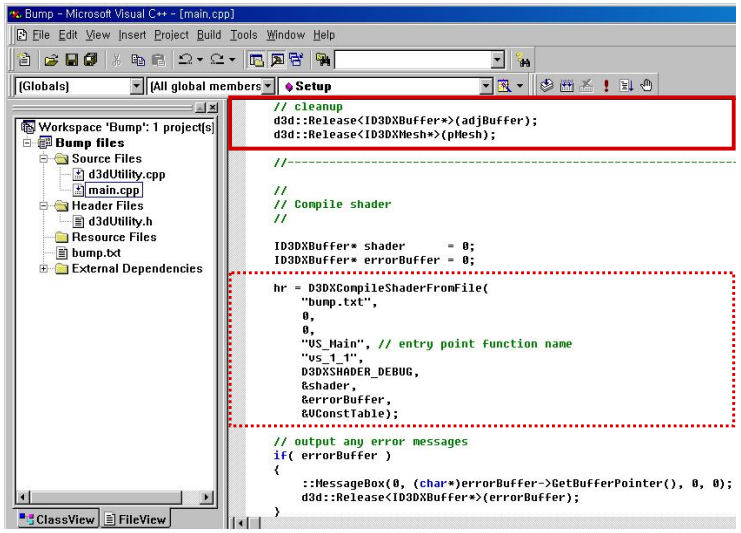
Loading a Mesh from X-File



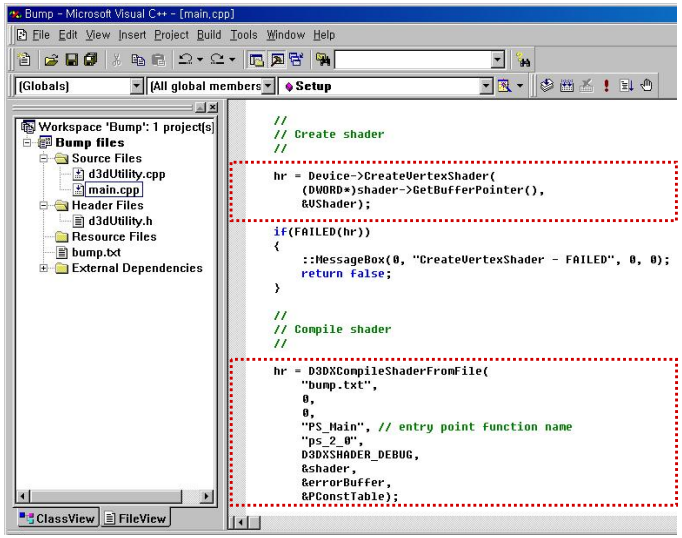
Computing Normal & Tangent Vectors



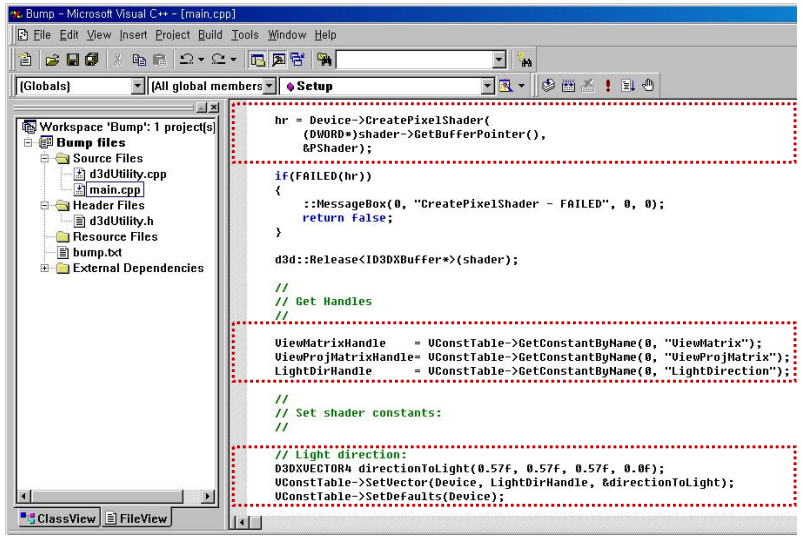
Cleanup & Compiling VS



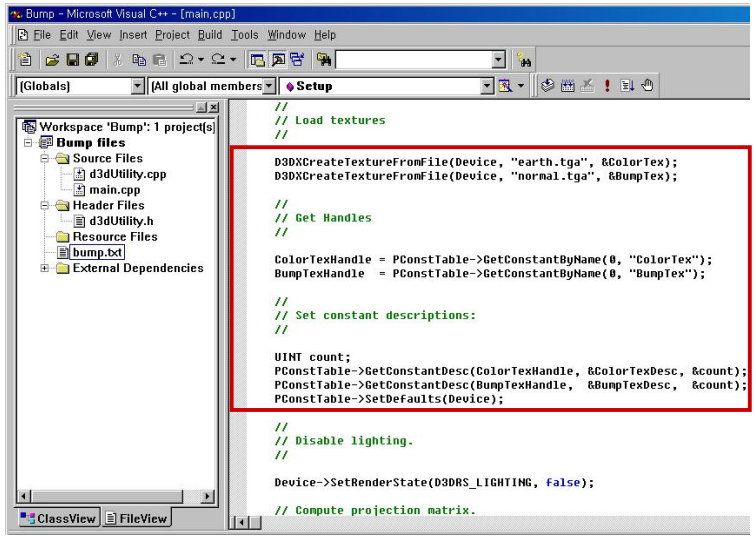
Creating VS & Compiling PS



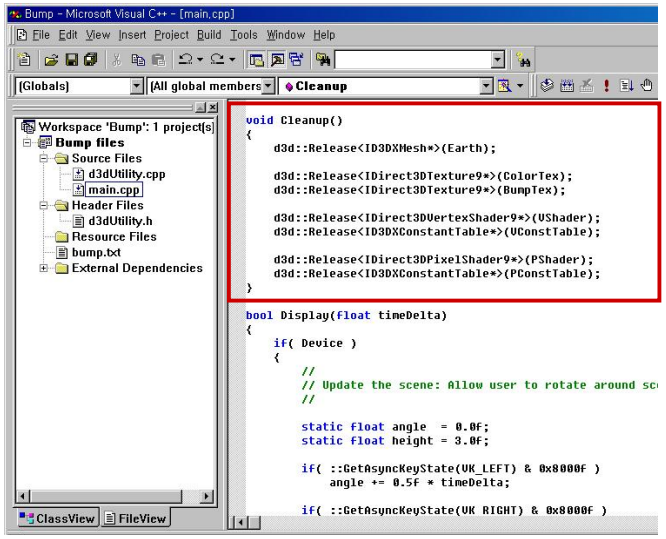
Creating PS & Getting Handles



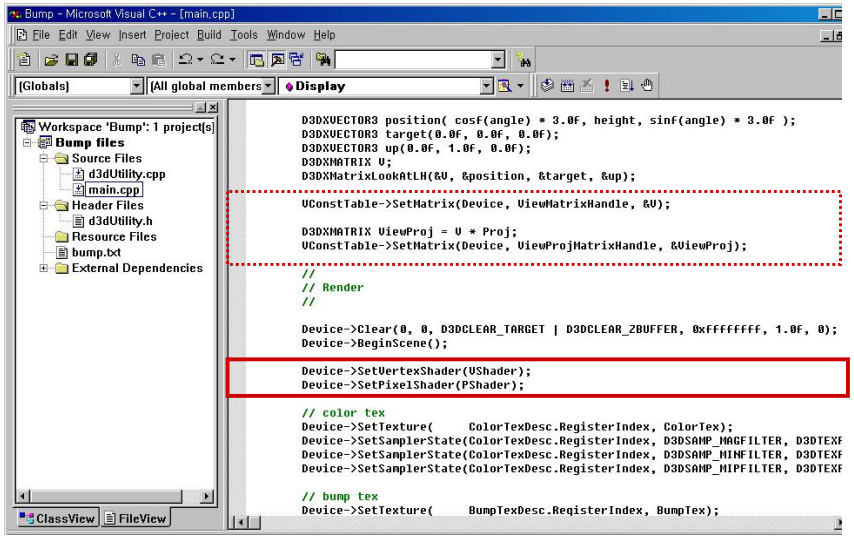
Getting Handles for Samplers



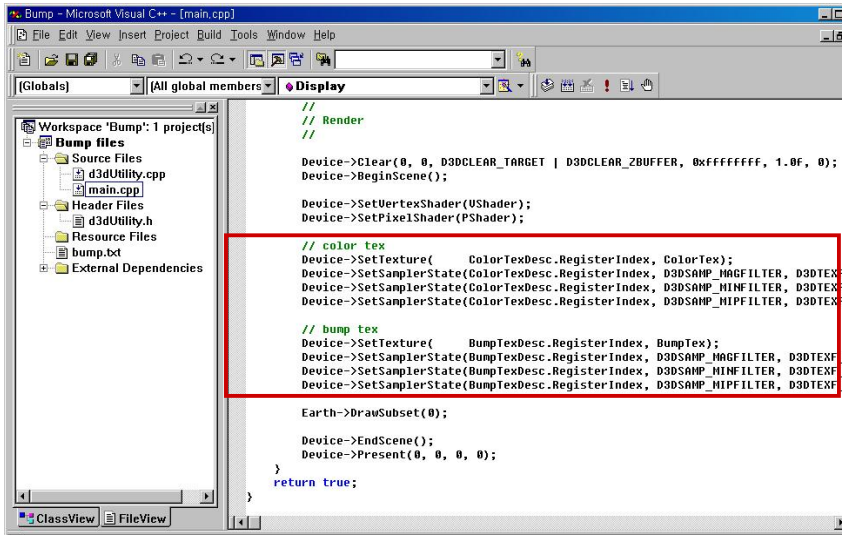
Cleanup ( )



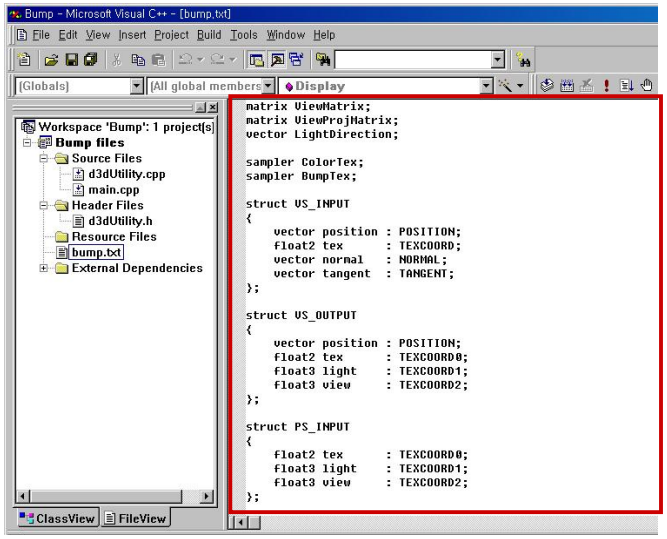
Enabling VS and PS



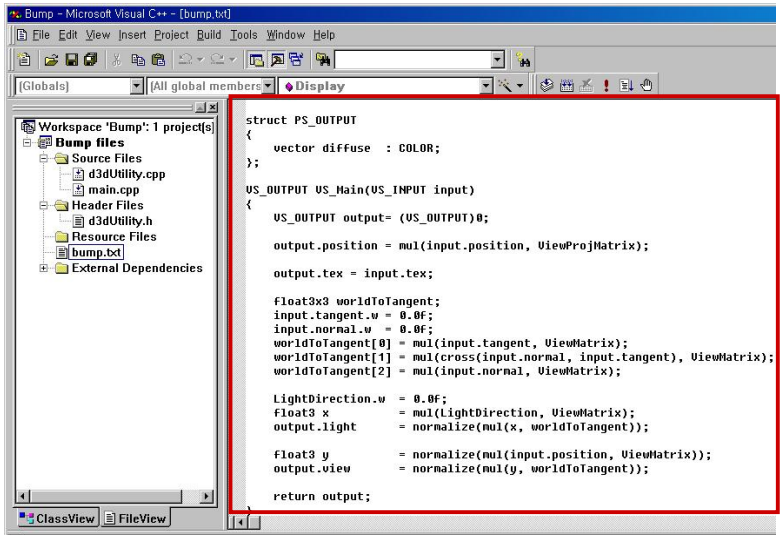
Bump Mapping



Shader: bump.txt (1)



Shader: bump.txt (2)



Shader: bump.txt (3)