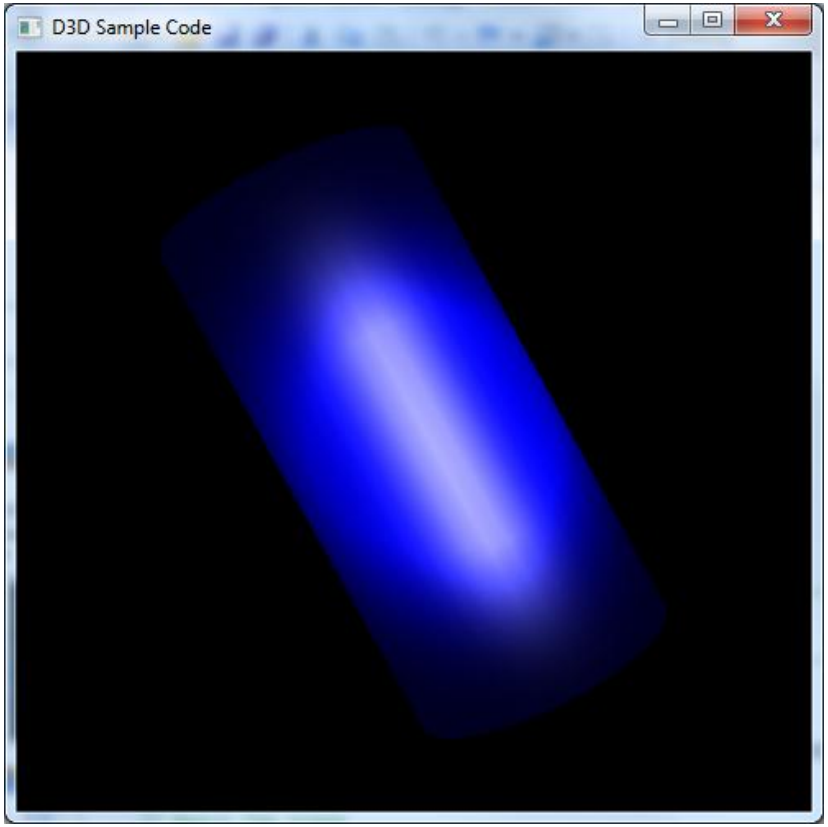
DirectX Programming #5

Kang, Seongtae

Computer Graphics, 2009 Spring

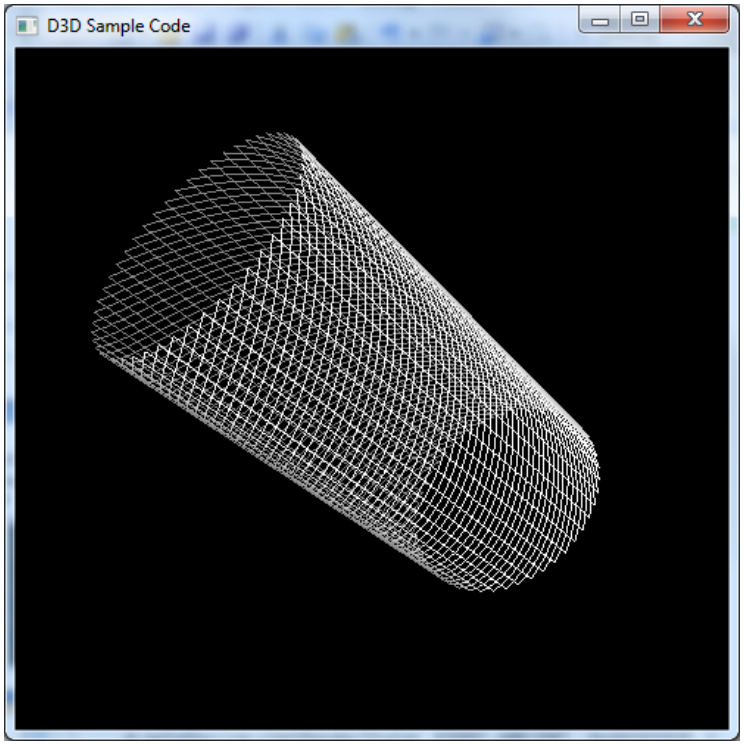
The Sample Program

- A blue cylinder with white specular color
- Gouraud shading with one spotlight



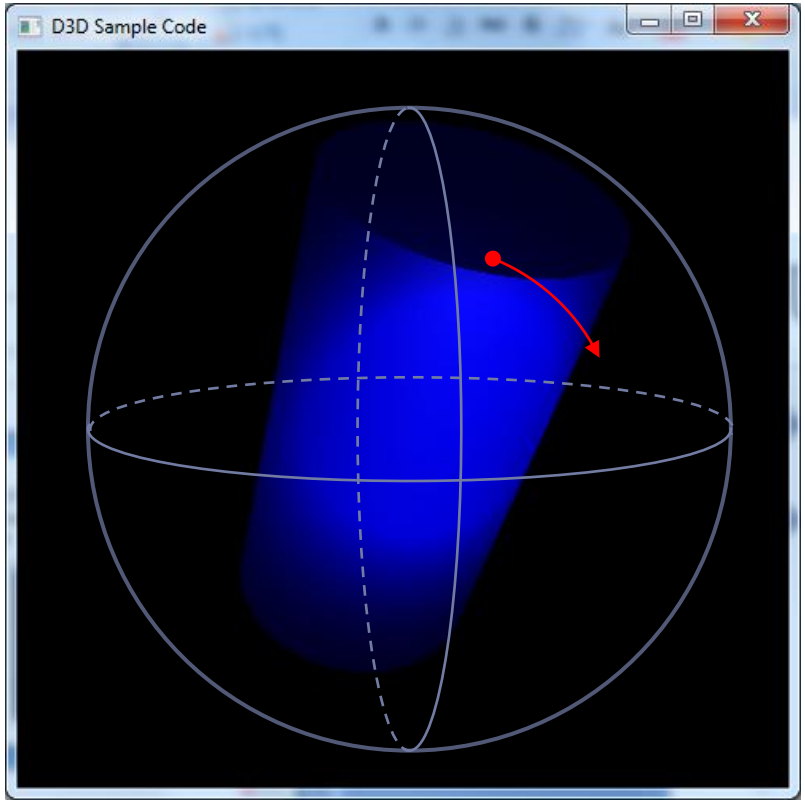
Primitives

- const int CIRCLE_DIVISION = 50;
- const int PILLAR_DIVISION = 50;



User Interface

- Rotation and panning using mouse
- ArcBall system

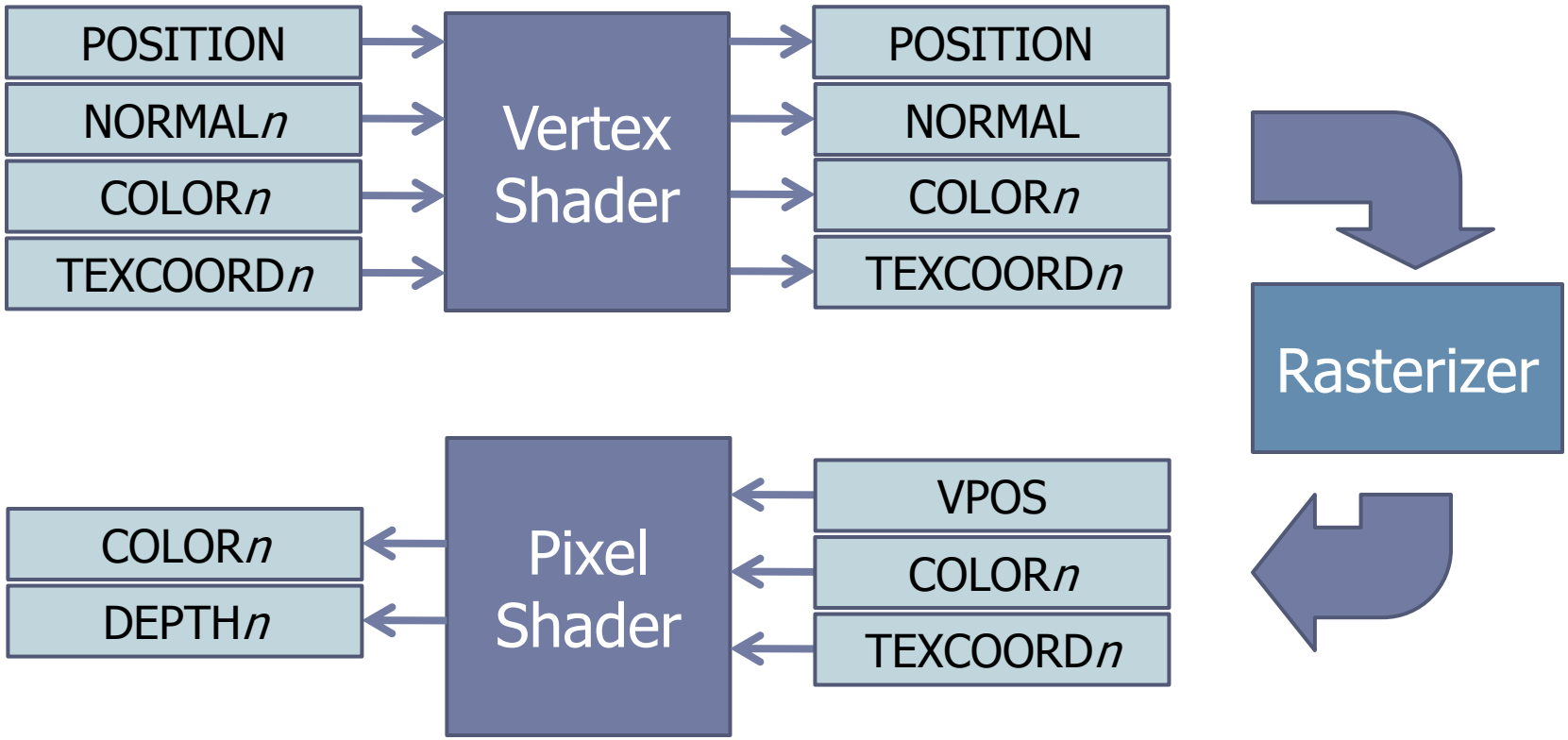


We Will…

- Modify the sample code to use HLSL
  - Creating an effect file which describes functions for
    - Spotlight lighting
    - Gouraud shading
  - Passing variable to the effect
  - Rendering with the effect

Semantics for VS/PS Input and Output

- VS and PS 2.0 model
- Not all semantics are described here



Type Definition

- Structures for input and output parameters of VS/PS
  - Usual strucutre definition + semantic description
  - Vertex shader input must be matched to the CPU-side vertex description (FVF)

```
struct VS_INPUT
{
        float4 Position : POSITION;
        float3 Normal : NORMAL;
};

struct Gouraud_VS_OUTPUT
{
        float4 Position : POSITION;
        float4 Color : COLOR0;
};
```

Type Definition

- D3D object mappers
  - Just for convenience
  - We will pass D3DLIGHT9 / D3DMATERIAL9 object directly to the effect from CPU-side using this mapping structure

```
struct LIGHT                        // Light structure which corresponds to D3DLIGHT9 on CPU-side
{
        int         Type;           // This component will be ignored. We consider the spotlight type only
        float4      Diffuse;
        float4      Specular;
        float4      Ambient;
        float3      Position;
        float3      Direction;
        float       Range;
        float       Falloff;
        float       Att0;
        float       Att1;
        float       Att2;
        float       Theta;
        float       Phi;
};
```

Variable Definition

- Define parameters passed from CPU-side as global variable form
  - Viewing parameter
    - WVP matrices
  - Shading parameter
    - Material
    - Light and Global ambient factor
    - Eye position

```
// variable definition

MATERIAL material;
LIGHT light;
float4x4 matWorld, matView, matProj;
float3 eyePosition;
float globalAmbient;
```

Functions : Vertex Shader

- Gouraud shading model : Per-vertex lighting

```
float4 Lighting(float3 position, float3 normal)
{
                                        // Actual lighting routine here
}
```

```
Gouraud_VS_OUTPUT GouraudShadingVS(VS_INPUT i)
{
        Gouraud_VS_OUTPUT o;

        i.Position = mul(i.Position, matWorld);
        i.Normal = mul(i.Normal, matWorld);          // Normal is needed to be world-transformed

        o.Color = Lighting(i.Position, i.Normal);    // Lighting in the world coordinate

        o.Position = mul(mul(i.Position, matView), matProj);

        return o;
}
```

Functions : Pixel Shader

- In Gouraud shading, PS just passes rasterized color value
- We use enumeration of primitive types and semantics instead of structures for parameter passing
  - It's just a matter of choice

```
float4 GouraudShadingPS(float4 inColor : COLOR0) : COLOR0
{
        return inColor;
}
```

Functions : Lighting

- Spotlight with Phong illumination model
  - See DirectX Document
    - Mathematics of Lighting (Direct3D 9)
    - Attenuation and Spotlight Factor (Direct3D 9)

```
float4 Lighting(float3 position, float3 normal)
{
                                                    :
                                                    :
}
```

Functions : Lighting

- Calculating atten and spot

```
float3 Ldcs = -light.Direction;
float3 Ldir = light.Position - position;
float d = length(Ldir);
float rho = dot(normalize(Ldcs), normalize(Ldir));
float atten, spot;

if(d > light.Range) {
    atten = 0.0;
    spot = 0.0;
}
else {
    atten = 1.0 / (light.Att0 + light.Att1 * d + light.Att2 * d * d);
    spot = saturate(pow((rho - cos(light.Phi / 2)) / (cos(light.Theta / 2) - cos(light.Phi / 2)), light.Falloff));
}
```

$$Atten = 1/(att0_i + att1_i * d + att2_i * d^2)$$

$$spot_i = \begin{cases} 1 & \text{for non- spotlights or if } rho_i > \cos(\frac{theta_i}{2}) \\ 0 & \text{if } rho_i \le \cos(\frac{phi_i}{2}) \\ \left[\frac{rho_i - \cos(\frac{phi_i}{2})}{\cos(\frac{theta_i}{2}) - \cos(\frac{phi_i}{2})} \right]^{falloff} & \text{otherwise} \end{cases}$$

Functions : Lighting

- Calculate illuminations

```
float3 N = normalize(normal);
Ldir = normalize(Ldir);
float3 H = normalize(normalize(eyePosition - position) + Ldir);

float4 ambient = saturate(material.Ambient * (globalAmbient + light.Ambient * spot * atten));
float4 diffuse = saturate(material.Diffuse * light.Diffuse * dot(N, Ldir) * spot * atten);
float4 specular = saturate(material.Specular * light.Specular * pow(dot(N, H), material.Power) * atten * spot);

return saturate(ambient + diffuse + specular);
```

Ambient Lighting = $C_a*[G_a + \Sigma Att*Spot*L_a]$

Diffuse Lighting = $\Sigma\ C_d*L_d*(N \cdot L_{dir})*Atten*Spot$

Specular Lighting = $C_s * \Sigma\ L_s * (N \bullet H)^P * Atten * Spot]$

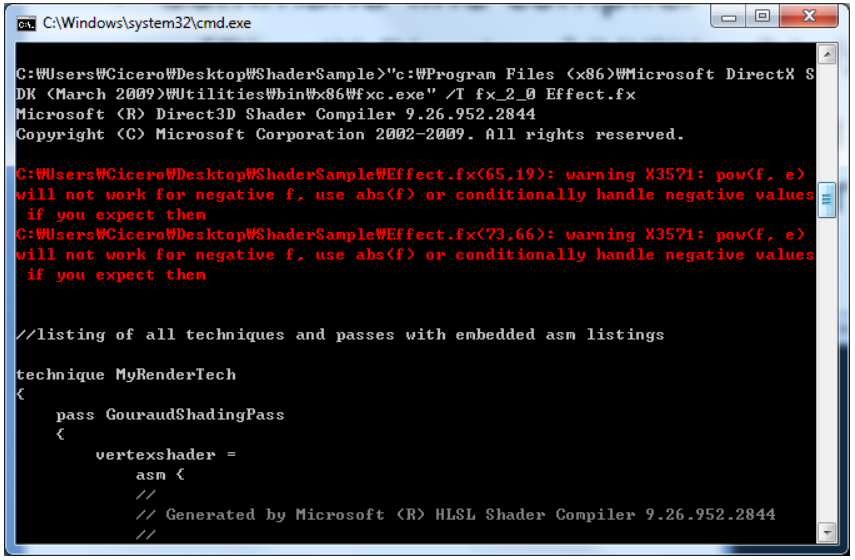
Technique and Pass Description

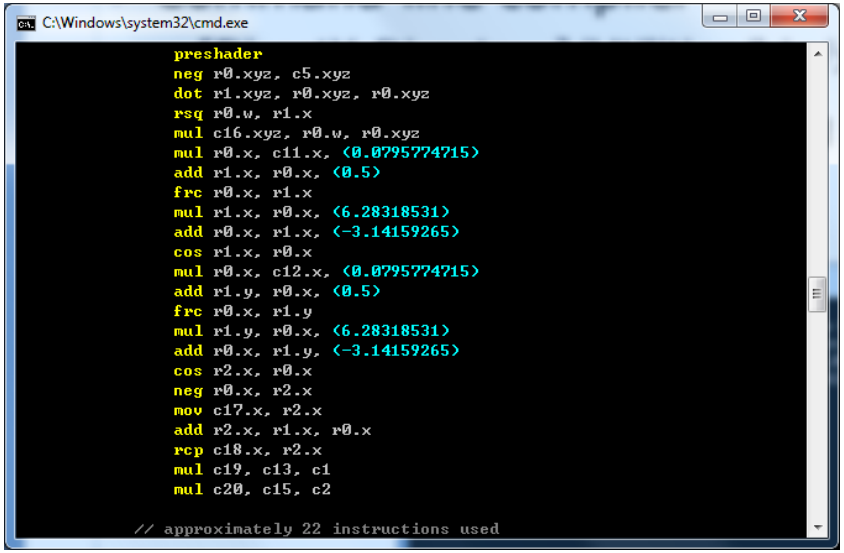
- Describe render states and compilation options

```
// technique definition
technique MyRenderTech
{
    pass GouraudShadingPass {
        ZEnable = TRUE;
        CullMode = NONE;

        VertexShader = compile vs_2_0 GouraudShadingVS();
        PixelShader = compile ps_2_0 GouraudShadingPS();
    }
}
```

Syntax Checking

- Command line compiler
  - [DirectX Directory]/Utilities/bin/x86/fxc.exe
- If there's no syntax error, you can see shader assembly mnemonics for your HLSL code
- Semantic errors are not catched on fxc compiler!





CPU-Side : Creating an Effect

- D3DXCreateEffectFromFile
- Look out for the file path

```
LPD3DXEFFECT        g_pEffect = NULL;

// Create an Effect
if (FAILED( D3DXCreateEffectFromFile( g_pd3dDevice, _T("Effect.fx"), NULL, NULL, NULL,
                                                                    NULL, &g_pEffect, NULL) ) )
{
            return E_FAIL;
}
```

CPU-Side : Passing Variables

- Set[T] for [T] type
  - Matrix, Float, Vector …
- SetValue for custom data type, such as structs

```
g_pEffect->SetMatrix("matWorld", &matWorld);

g_pEffect->SetValue("eyePosition", &vEyePt, sizeof(D3DXVECTOR3));

g_pEffect->SetValue("material", &mtrl, sizeof(mtrl));

g_pEffect->SetFloat("globalAmbient", 0.125f);
```

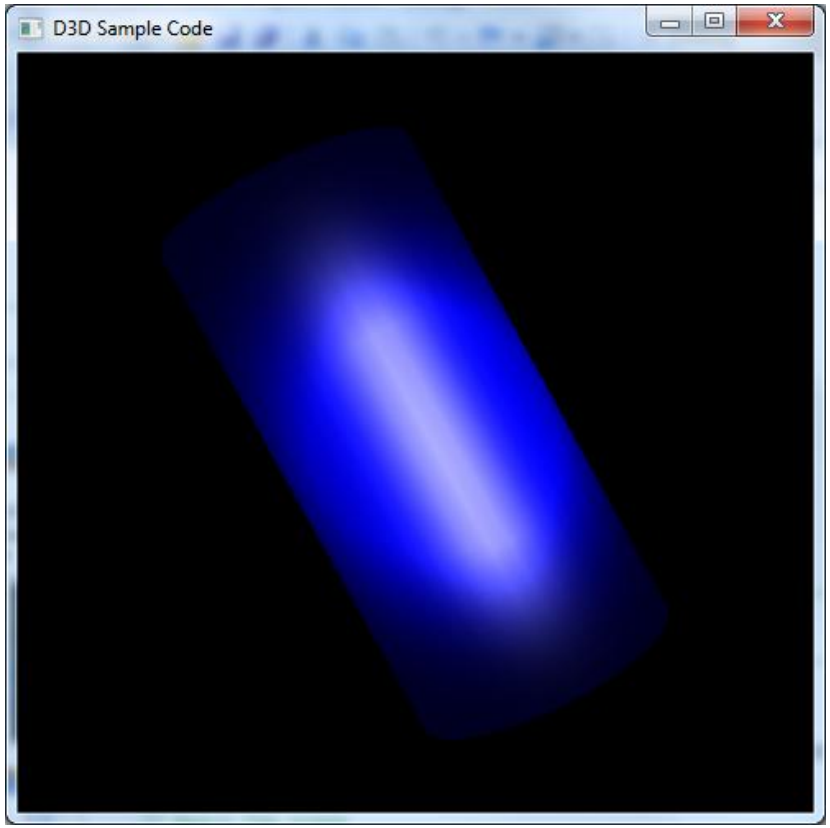
CPU-Side : Rendering

- Set the technique and begin a pass

```
g_pEffect->SetTechnique("MyRenderTech");

UINT nPasses;

g_pEffect->Begin(&nPasses, 0);

g_pEffect->BeginPass(0);

for(int i=0; i<PILLAR_DIVISION; i++)
        g_pd3dDevice->DrawPrimitive( D3DPT_TRIANGLESTRIP,
                                     i * (CIRCLE_DIVISION * 2), CIRCLE_DIVISION * 2 - 2);

g_pEffect->EndPass();

g_pEffect->End();
```

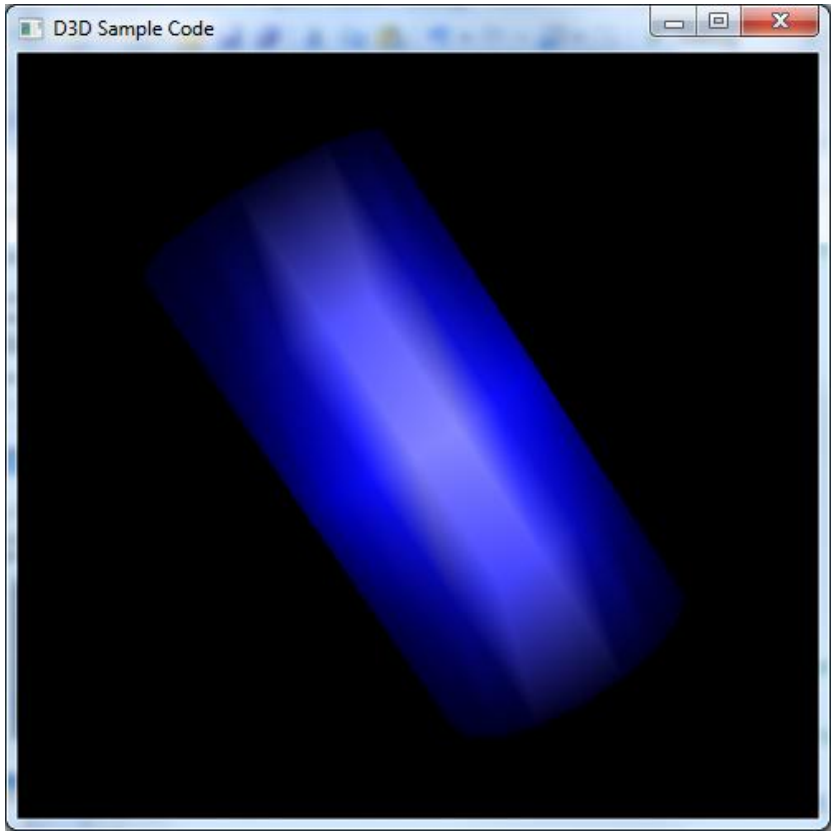
See Your Result

- It may not be completely identical to the result of fixed-pipeline sample, for some implementation issues



Cons. of Gouraud Shading Model

- Reduce # of polygon and see the result



const int CIRCLE_DIVISION = 20;
const int PILLAR_DIVISION = 4;

DX Assignment

- Implement Phong shading using HLSL
  - You can reuse your lighting function
  - Make an additional pass on your effect file
  - Test your code with *pEffect->BeginPass(1);*
- Implement the point and directional light source
  - Optional

- Commit your .fx file via e-mail
  - If you use additional parameters to be passed from CPU-side, describe them on e-mail content
  - cicero@cglab.snu.ac.kr
- Due : 6/11 (Thu.), no delay