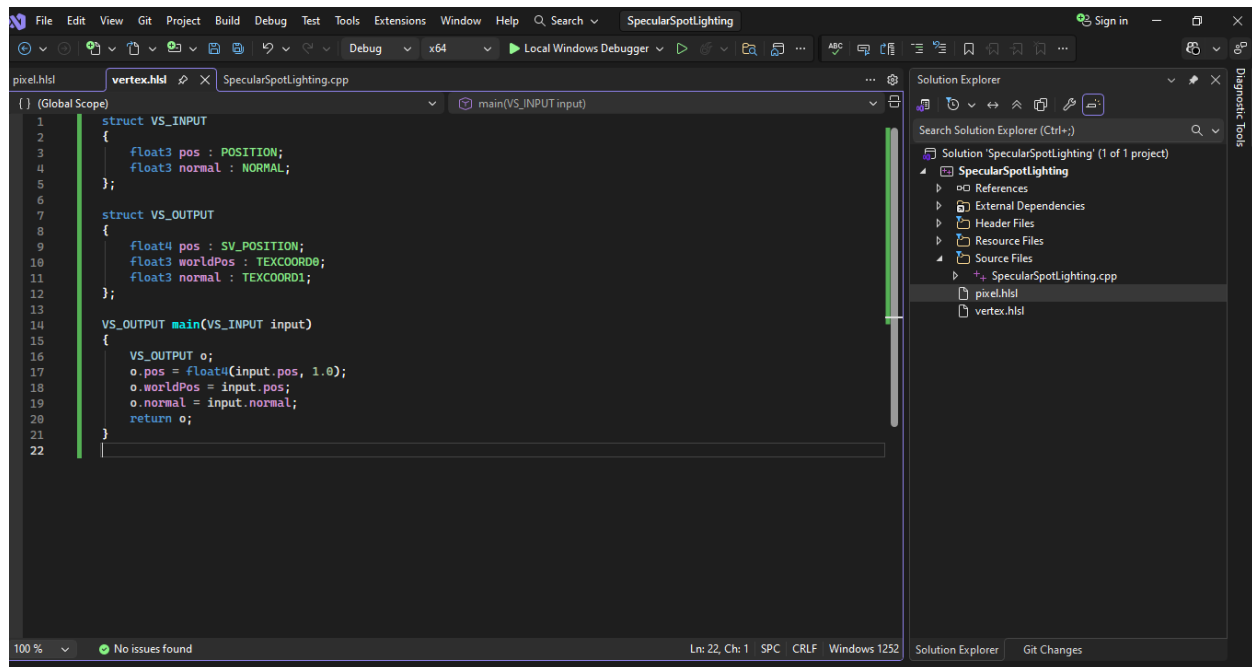


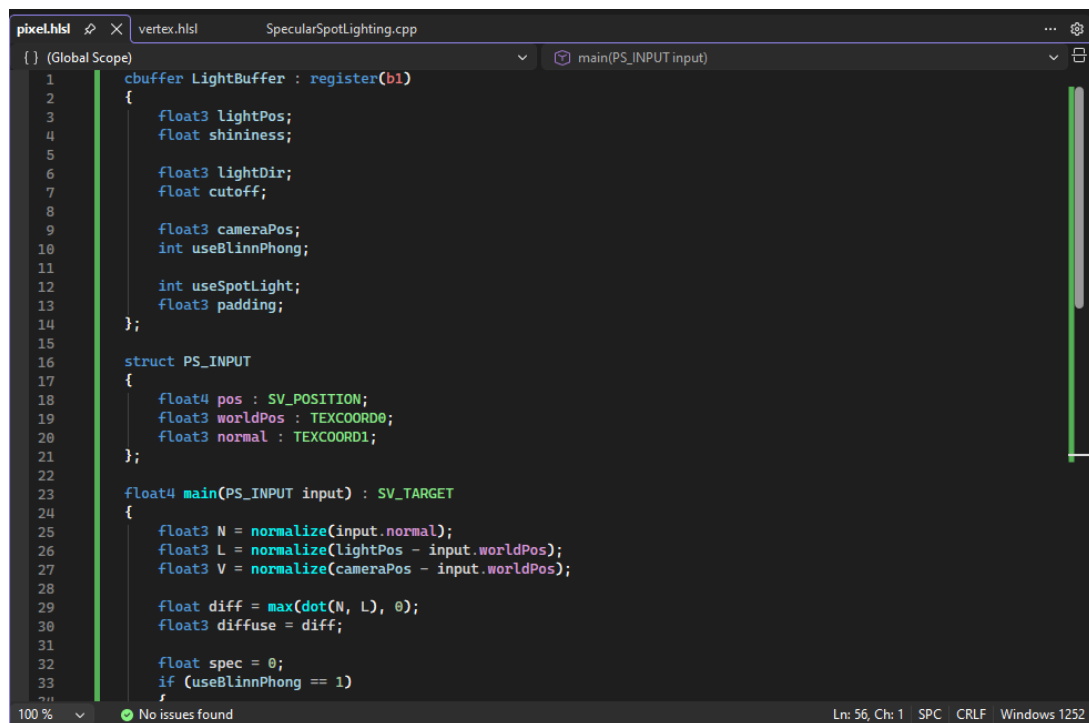
Practical 4: Implementation of Specular and Spot Lighting Using Programmable Shaders

Shader Source Files:



The screenshot shows the Visual Studio IDE with the 'SpecularSpotLighting' project open. The 'vertex.hlsli' file is selected in the Solution Explorer. The code defines a vertex shader with the following structure:

```
1 struct VS_INPUT
2 {
3     float3 pos : POSITION;
4     float3 normal : NORMAL;
5 };
6
7 struct VS_OUTPUT
8 {
9     float4 pos : SV_POSITION;
10    float3 worldPos : TEXCOORD0;
11    float3 normal : TEXCOORD1;
12 };
13
14 VS_OUTPUT main(VS_INPUT input)
15 {
16     VS_OUTPUT o;
17     o.pos = float4(input.pos, 1.0);
18     o.worldPos = input.pos;
19     o.normal = input.normal;
20     return o;
21 }
22
```



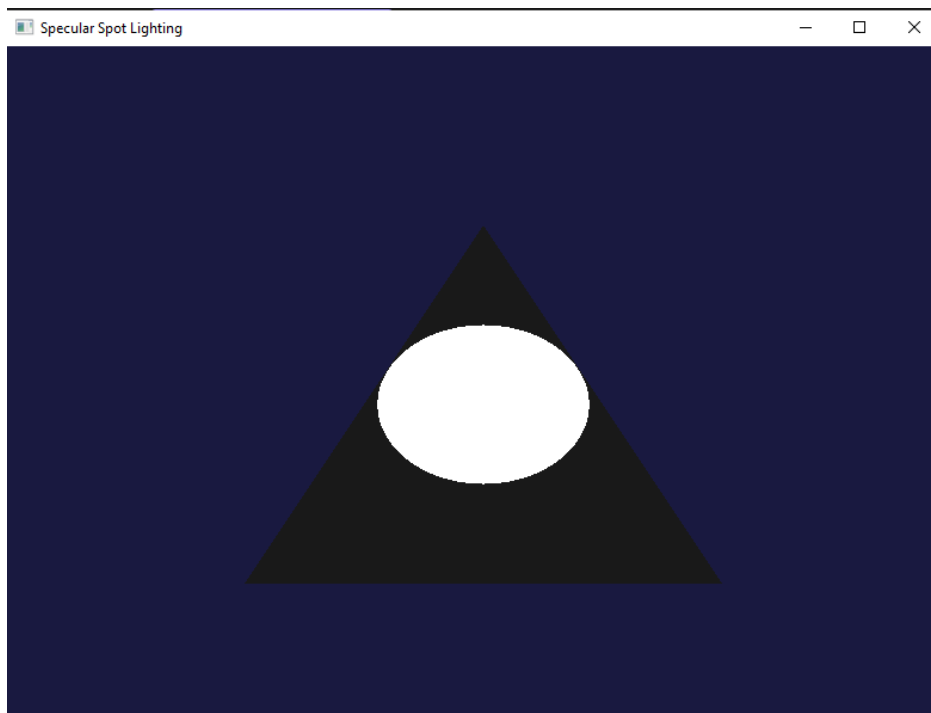
The screenshot shows the Visual Studio IDE with the 'SpecularSpotLighting' project open. The 'pixel.hlsli' file is selected in the Solution Explorer. The code defines a pixel shader with the following structure:

```
1 cbuffer LightBuffer : register(b1)
2 {
3     float3 lightPos;
4     float shininess;
5
6     float3 lightDir;
7     float cutoff;
8
9     float3 cameraPos;
10    int useBlinnPhong;
11
12    int useSpotLight;
13    float3 padding;
14 };
15
16 struct PS_INPUT
17 {
18     float4 pos : SV_POSITION;
19     float3 worldPos : TEXCOORD0;
20     float3 normal : TEXCOORD1;
21 };
22
23 float4 main(PS_INPUT input) : SV_TARGET
24 {
25     float3 N = normalize(input.normal);
26     float3 L = normalize(lightPos - input.worldPos);
27     float3 V = normalize(cameraPos - input.worldPos);
28
29     float diff = max(dot(N, L), 0);
30     float3 diffuse = diff;
31
32     float spec = 0;
33     if (useBlinnPhong == 1)
34     {
35         float3 R = normalize(lightDir - input.worldPos);
36         float3 H = normalize(R + N);
37         float3 reflR = pow(diff, shininess);
38         float3 reflD = 1 - reflR;
39         float3 reflS = 0;
40         float3 refl = reflD * diffuse + reflR * (diff * diff * spec);
41         float3 color = input.worldPos * 0.5 + refl;
42         return float4(color, 1.0);
43     }
44     return float4(diffuse, 1.0);
45 }
```

```
pixel.hlsl  vertex.hlsl  SpecularSpotLighting.cpp
{ } (Global Scope)  main(PS_INPUT input)
25     float3 N = normalize(input.normal);
26     float3 L = normalize(lightPos - input.worldPos);
27     float3 V = normalize(cameraPos - input.worldPos);
28
29     float diff = max(dot(N, L), 0);
30     float3 diffuse = diff;
31
32     float spec = 0;
33     if (useBlinnPhong == 1)
34     {
35         float3 H = normalize(L + V);
36         spec = pow(max(dot(N, H), 0), shininess);
37     }
38     else
39     {
40         float3 R = reflect(-L, N);
41         spec = pow(max(dot(R, V), 0), shininess);
42     }
43
44     float3 specular = spec;
45     float3 ambient = 0.1;
46
47     if (useSpotLight == 1)
48     {
49         float theta = dot(normalize(lightDir), normalize(-L));
50         if (theta < cutoff)
51             diffuse = specular = 0;
52     }
53
54     return float4(ambient + diffuse + specular, 1);
55 }
56
```

100 % No issues found Ln: 56, Ch: 1 SPC CRLF Windows 1252

Output:



Report:

This experiment demonstrates the implementation of **specular spot lighting** using **Direct3D 11** and **HLSL shaders**. The objective is to render a basic geometric object and apply realistic lighting using the **Blinn-Phong illumination model** combined with a **spotlight effect**. The program utilizes the Direct3D graphics pipeline to perform lighting calculations on the GPU.

Implementation Overview

The application is developed in C++ using Direct3D 11. A window is created and the graphics device, swap chain, render target view, and shaders are initialized. A single triangle is rendered using a vertex buffer and input layout.

Lighting parameters such as light position, direction, shininess, and camera position are stored in a **constant buffer** and passed to the pixel shader each frame.

Lighting Model

The **Blinn-Phong lighting model** is used to compute illumination. It consists of:

- **Ambient lighting** to provide base illumination
- **Diffuse lighting**, calculated using the dot product between the surface normal and light direction
- **Specular lighting**, calculated using the half-vector method and a shininess exponent

The shininess value controls the intensity and sharpness of the specular highlight.

Spotlight Effect

A spotlight is implemented by restricting light contribution to a cone defined by a cutoff angle. The angle between the light direction and the fragment direction is computed, and lighting is applied only when it falls within the cutoff. This creates a focused circular illumination similar to a real-world spotlight.

Results

On execution, a triangle is rendered at the center of the screen with visible specular highlights. The spotlight produces a bright circular region on the triangle surface, and changes in shininess affect the size and intensity of the highlight. Keyboard controls allow toggling between lighting modes and adjusting shininess in real time.

Conclusion

The experiment successfully demonstrates specular spot lighting using Direct3D 11. The correct use of shaders, constant buffers, and lighting equations results in realistic illumination. The output verifies proper functioning of the Direct3D rendering pipeline and fulfills the objectives of the assignment.