**Lab Report**

**Lab 10 - Shaders Lab 02**

**Craig Broskow**

**November 11, 2015**

Introduction:

This lab continued our exploration of Shader coding by creating more complex shaders. Specifically, we created a specular shader in which light reflected from a light source is highlighted in the image. We also moved the majority of the shader code from the vertex program into the fragment program in order to improve the quality of the lighting and the performance of the shader.

Methods:

Shader coding for this lab continued with the same approach we used for our last shader lab: a combination of CG (C for Graphics) code along with a "wrapper" of Unity's Shaderlab code to allow the shader to interact well with the Unity game engine. However, this lab added a couple new properties to the shader: the specular color of the shader ("\_SpecColour") and the shininess of the object, which **must** be called "\_Shininess" to be recognized as such by other shader functions (presumably within the Unity context). Of course, these properties must have the same names within the CG code and the Shaderlab code in order for them to interface properly with the Unity Inspector.

Since we are dealing with light rays bouncing around in 3D space, we must perform a number of vector calculations to figure out what the player (i.e., the camera) sees. Initially, these calculations are performed within the vertex program, but most of them are transferred over to the fragment program in a later refinement of this lab's original shader software.

In determining the color value to be returned from the vertex program, we start off by determining three vectors: 1. the direction of the light (obtained from the Unity variable "\_WorldSpaceLightPos0"); 2. the normal vector from the surface of the object (obtained from the normal of the input and converted from world space to object space); and finally 3. the difference between the viewer vector and the reflection vector (obtained by subtracting the input vertex position from the camera position, in world space).

There are two types of relected light that will factor into the color returned by the vertex program: diffuse reflection and specular reflection. Diffuse reflection is, of course, no different for specular lighting than it is for Lambert lighting; it is calculated by taking the dot product of the normal vector and the light vector, multiplied by the primary light color (Unity variable "\_LightColor0") and whatever attenuation is present (none for this lab). The calculation of specular reflection is much more complicated (and of course the primary purpose of the lab). In essence, the value of specular reflection is returned by the following snippet of code:

specularReflection = max(0.0, dot(normalDirection, lightDirection)) \*

pow(max(0.0, dot(reflect(-lightDirection, normalDirection), viewDirection)), \_Shininess);

Note that this code uses many of the factors we have either calculated or incorporated as properties in the shader: lightDirection, normalDirection, viewDirection and \_Shininess.

The color returned by the vertex program is a combination of the reflected light (diffuse and specular), ambient light ("UNITY\_LIGHTMODEL\_AMBIENT") and the color tint property ("\_Color"). In the first "On Your Own" exercise, the specular color property ("\_SpecColour") and attentuation also factor into the color returned by the vertex program. In the second "On Your Own" exercise (parts b and c), processing of the returned color is moved from the vertex program to the fragment program to improve the quality of the lighting and the performance of the shader. The vertex program is left to its primary purpose of calculating vertex position, and also calculates the world space positions of the vertex and its normal vector.

Conclusions:

Having little to no experience with computer graphics, I still can't say that I completely understand the potential uses of custom shader programming. This lab certainly introduced an application that made a lot more sense to me: the highlighting of light source reflections. While there are probably many situations where custom shaders are important to game programming, I don't know how many shaders are available "off the shelf" or how often they must be customized to a particular game environment or situation.

I also suspect that custom shaders may have a fairly important reliance on the specific graphics hardware which the gamer is playing the game on. Consoles are probably pretty standardized in terms of graphics hardware, while PC and mobile games probably have to deal with a much greater variation in graphics hardware capabilities. My ignorance of the extent to which custom shaders rely on graphics hardware further complicates (for me) the desirability of investing programming resources (time, money, etc.) in developing custom shaders.

In summary, I'm feeling a little more comfortable in working with shader code, but I have very little idea of where and when it would be advisable to apply these techniques in games that I might work on (ignoring for the moment the supervisor who says "Do it!").

Postlab Questions:

1. How does max(0.0, specularReflection) soften the colour bleed onto the backside of the object when using a specular shader?

In a shader, values less than 0 indicate that the light should display on the back of the object. By taking the maximum of 0 and the specularReflection, we reduce the chance that the the color will show up on the back of the object.

2. How does max(0.0, dot(normalDirection, lightDirection)) remove the specular highlight from appearing on the back of the object?

The dot product of the normalDirection vector and the lightDirection vector gives the "fade factor" to apply to the highlight. However, as in the answer to question # 1, values less than zero will show up on the back of the object. Using the max function to limit values to zero and above lessen the chance that the specular highlight will appear on the back of the object.

On Your Own Problems:

The shader program for the main portion of this lab is named "SpecularShader". The shader program for the first "On Your Own" problem (a) is named "SpecularShader1". For questions (b) and (c) of the "On Your Own" section, the shader is named "SpecularShader2". In the Main Scene, these shaders can be applied to the "SpecularCapsule" game object to demonstrate their effects.

Code:

// SpecularShader.shader

Shader "Custom/SpecularShader" { // Basic shader for Lab 10

Properties {

\_Color ("Color Tint", Color) = (1,1,1,1)

\_SpecColour("Specular Color", Color) = (1,1,1,1)

\_Shininess("Shininess", float) = 10

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform float4 \_SpecColour;

uniform float \_Shininess;

//unity defined variables

uniform float4 \_LightColor0;

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float3 vertexNormal : NORMAL;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 pixelCol : COLOR;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

float3 lightDirection;

float attenuation = 1.0;

lightDirection = normalize(\_WorldSpaceLightPos0.xyz);

float3 normalDirection = normalize(mul(float4(input.vertexNormal, 0.0), \_Object2World).xyz);

float3 viewDirection = normalize(float3(float4(\_WorldSpaceCameraPos.xyz, 1.0) -

mul(\_Object2World, input.vertexPos).xyz));

float3 diffuseReflection = attenuation \* \_LightColor0.xyz \* max(0.0, dot(normalDirection, lightDirection));

float3 specularReflection = reflect(-lightDirection, normalDirection);

specularReflection = dot(specularReflection, viewDirection);

specularReflection = max(0.0, specularReflection);

specularReflection = max(0.0, dot(normalDirection, lightDirection)) \* specularReflection;

specularReflection = pow(max(0.0, specularReflection), \_Shininess);

specularReflection = max(0.0, dot(normalDirection, lightDirection)) \* specularReflection;

float3 finalLight = specularReflection + diffuseReflection + UNITY\_LIGHTMODEL\_AMBIENT;

toReturn.pixelCol = float4(finalLight \* \_Color, 1.0);

toReturn.pixelPos = mul(UNITY\_MATRIX\_MVP, input.vertexPos);

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

return input.pixelCol;

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}

// SpecularShader1.shader

Shader "Custom/SpecularShader1" { // Lab 10 Shader for On Your Own #1 (a)

Properties {

\_Color ("Color Tint", Color) = (1,1,1,1)

\_SpecColour("Specular Color", Color) = (1,1,1,1)

\_Shininess("Shininess", float) = 10

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform float4 \_SpecColour;

uniform float \_Shininess;

//unity defined variables

uniform float4 \_LightColor0;

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float3 vertexNormal : NORMAL;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 pixelCol : COLOR;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

float3 lightDirection;

float attenuation = 1.0;

lightDirection = normalize(\_WorldSpaceLightPos0.xyz);

float3 normalDirection = normalize(mul(float4(input.vertexNormal, 0.0), \_Object2World).xyz);

float3 viewDirection = normalize(float3(float4(\_WorldSpaceCameraPos.xyz, 1.0) -

mul(\_Object2World, input.vertexPos).xyz));

float3 diffuseReflection = attenuation \* \_LightColor0.xyz \* max(0.0, dot(normalDirection, lightDirection));

float3 specularReflection = reflect(-lightDirection, normalDirection);

specularReflection = dot(specularReflection, viewDirection);

specularReflection = max(0.0, specularReflection);

specularReflection = max(0.0, dot(normalDirection, lightDirection)) \* specularReflection;

specularReflection = pow(max(0.0, specularReflection), \_Shininess);

specularReflection = max(0.0, dot(normalDirection, lightDirection)) \* specularReflection;

float3 finalLight = specularReflection + diffuseReflection + UNITY\_LIGHTMODEL\_AMBIENT;

toReturn.pixelCol = float4(finalLight \* \_Color.rgb \* attenuation \* \_SpecColour.rgb, 1.0);

toReturn.pixelPos = mul(UNITY\_MATRIX\_MVP, input.vertexPos);

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

return input.pixelCol;

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}

// SpecularShader2.shader

Shader "Custom/SpecularShader2" { // Lab 10 Shader for On Your Own #2 (b and c)

Properties {

\_Color ("Color Tint", Color) = (1,1,1,1)

\_SpecColour("Specular Color", Color) = (1,1,1,1)

\_Shininess("Shininess", float) = 10

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform float4 \_SpecColour;

uniform float \_Shininess;

//unity defined variables

uniform float4 \_LightColor0;

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float3 vertexNormal : NORMAL;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float3 normalDirection : TEXCOORD0;

float4 pixelWorldPos : TEXCOORD1;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

toReturn.normalDirection = normalize(mul(float4(input.vertexNormal, 0.0), \_Object2World).xyz);

toReturn.pixelWorldPos = mul(\_Object2World, input.vertexPos);

toReturn.pixelPos = mul(UNITY\_MATRIX\_MVP, input.vertexPos);

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

float3 lightDirection;

float attenuation = 1.0;

lightDirection = normalize(\_WorldSpaceLightPos0.xyz);

float3 viewDirection = normalize(float3(float4(\_WorldSpaceCameraPos.xyz, 1.0) - input.pixelWorldPos.xyz));

float3 diffuseReflection = attenuation \* \_LightColor0.xyz \* max(0.0, dot(input.normalDirection, lightDirection));

float3 specularReflection = reflect(-lightDirection, input.normalDirection);

specularReflection = dot(specularReflection, viewDirection);

specularReflection = max(0.0, specularReflection);

specularReflection = max(0.0, dot(input.normalDirection, lightDirection)) \* specularReflection;

specularReflection = pow(max(0.0, specularReflection), \_Shininess);

specularReflection = max(0.0, dot(input.normalDirection, lightDirection)) \* specularReflection;

float3 finalLight = specularReflection + diffuseReflection + UNITY\_LIGHTMODEL\_AMBIENT;

return float4(finalLight \* \_Color.rgb \* attenuation \* \_SpecColour.rgb, 1.0);

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}