**Lab Report**

**Lab 9 - Shaders Lab 01**

**Craig Broskow**

**November 4, 2015**

Introduction:

This lab started teaching us about Shader coding by adding a texture map to a Shader. We also experimented with adjusting certain aspects about how the texture was rendered.

Methods:

Shaders in this course are 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. As such, the shader(s) for this Lab were typical for Vertex/Fragment shaders. They started off with a Shaderlab Properties section that allows the Unity interface to control parameters within the CG code "core" of the shader program. The primary properties for this Lab were for the actual texture to be added to the shader (called "\_MainTex") and the color tint to be applied to the shader (called "\_Color"). Those properties were duplicated within the CG code (with the **same names**) and datatypes "sampler2D" for \_MainTex and "float4" for \_Color.

Following the Properties section of the shader is the SubShader section, which is essentially a function written in CG code. The SubShader section itself contains a Pass section which layers effects on top of each other. Finally, we get to the CG code itself, which starts with the "CGPROGRAM" declaration and ends with the "ENDCG" declaration.

Within the CGPROGRAM CG code, there are multiple sections: variable declarations (some of which match up with the Unity parameters); input and output Structs (which hold information between the CG functions); a Vertex function (which works on a per-vertex level and allows for vertex deformation within the shader); and finally a Fragment function (which works on a per-pixel level, accepting input from the output Struct generated by the Vertex function, performs most of the work of the shader, and returns a color).

For this particular Lab, the CG code included a float4 property called "\_MainTex\_ST" which provides the Unity user the opportunity to change the tiling and offset of the texture from with the Unity interface. The input and output Structs of the Vertex function not only include the usual vertex position information, but also a semantic called "TEXCOORD0" which is used for mapping a position on the 2D texture to a position on the 3D model.

Within the Vertex function, the primary processing performed is to matrix-multiply the input vertex position by the model\*view\*projection matrix ("UNITY\_MATRIX\_MVP") to obtain the actual pixel position. This pixel position, along with the TEXCOORD0 semantic, is stored in the output Struct and passed into the Fragment function.

Within the Fragment function, the "tex2D" method is used to map texture 2D pixel positions into model 3D pixel positions, and calculations are performed to adjust for scaling and offsets stored within the \_MainTex\_ST property (available for the user to change within the Unity interface). Finally, the Fragment function returns the color of the calculated pixel. In the "On Your Own" portions of this Lab, the color returned by the Fragment function is further modified based on user-defined color tint or lambert lighting.

Conclusions:

Since I don't have any experience with 3D Modeling (or computer graphics in general), I can't really say that I fully understand all of the potential uses of custom shader programming. Obviously, custom shaders can be used to produce some interesting and sometimes fascinating computer graphics effects. They are also probably useful in producing special effects based on the light sources present within a Unity scene, and the desired "mood" desired by a game designer. However, at least to me, these are fairly advanced techniques that will take a while to master, and will most likely be necessary in a relatively small number of situations that can't be handled either through shaders provided directly by Unity, or from third-party suppliers.

Postlab Questions:

1. Draw what a UV map might look like for the following object.

Please see the attached diagram.

2. What meaning does a UV have in relation to a texture and/or a model?

A texture is a 2D graphic image, while a model is a 3D mesh based on vertices. A UV map is essentially a function which identifies which positions on the 3D mesh are associated with which pixel locations on the 2D texture. This allows a 2D graphic image to "wrap around" a 3D mesh and give the impression that the 3D model is actually covered with a 3D graphic image, or "skin". In fact, of course, the 3D model is actually broken down into 2D triangles or polygons, which are themselves covered with 2D graphics. By using a sufficiently large number of 2D surfaces, the illusion is given that the observer is viewing a 3D image.

3. What does it mean that you will sample a texture?

"Sampling a texture" refers to retrieving the data from a texture.

4. What does it mean when you say that you are unwrapping a texture?

"Unwrapping a texture" refers to mapping a pixel on the texture to a position on the 3D model.

5. If I had a sampler2D named \_RawrTex, what would I name the next variable to capture the tiling and offset of that sampler2D?

The tiling and offset variable of a sampler2D named "\_RawrTex" should be named "\_RawrTex\_ST".

On Your Own Problems:

The shader program for the main portion of this lab is named "TextureShader". The shader programs for "On Your Own" problems #1, #2 and #3 are named "TextureShader1", "TextureShader2" and "TextureShader3", respectively. Similarly, in the Main Scene there are capsules with these shaders that are named "TextureCapsule", "TextureCapsule1", "TextureCapsule2" and "TextureCapsule3" to demonstrate their effects.

Code:

// TextureShader.shader

Shader "Custom/TextureShader" // Basic shader for Lab 9

{

Properties {

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

\_MainTex("Diffuse Texture", 2D) = "white" {}

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform sampler2D \_MainTex;

uniform float4 \_MainTex\_ST;

//unity defined variables

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float4 textureCoord : TEXCOORD0;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 tex: TEXCOORD0;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

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

toReturn.tex = input.textureCoord;

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

float4 tex = tex2D(\_MainTex, \_MainTex\_ST.xy \* input.tex.xy + \_MainTex\_ST.zw);

return float4(tex.rgb, 1.0);

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}

// TextureShader1.shader

Shader "Custom/TextureShader1" // Lab 9 Shader for On Your Own #1

{

Properties {

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

\_MainTex("Diffuse Texture", 2D) = "white" {}

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform sampler2D \_MainTex;

uniform float4 \_MainTex\_ST;

//unity defined variables

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float4 textureCoord : TEXCOORD0;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 tex: TEXCOORD0;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

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

toReturn.tex = input.textureCoord;

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

float4 tex = tex2D(\_MainTex, \_MainTex\_ST.xy \* input.tex.xy + \_MainTex\_ST.zw);

return float4(tex.rgb \* \_Color.rgb, 1.0);

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}

// TextureShader2.shader

Shader "Custom/TextureShader2" // Lab 9 Shader for On Your Own #2

{

Properties {

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

\_MainTex("Diffuse Texture", 2D) = "white" {}

}

SubShader {

Pass{

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform sampler2D \_MainTex;

uniform float4 \_MainTex\_ST;

//unity defined variables

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float4 textureCoord : TEXCOORD0;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 tex: TEXCOORD0;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

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

toReturn.tex = input.textureCoord;

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

float4 tex = tex2D(\_MainTex, \_MainTex\_ST.zw \* input.tex.xy + \_MainTex\_ST.xy);

// float4 tex = tex2D(\_MainTex, \_MainTex\_ST.xy \* input.tex.xy + \_MainTex\_ST.zw);

return float4(tex.rgb, 1.0);

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}

// TextureShader3.shader

Shader "Custom/TextureShader3" // Lab 9 Shader for On Your Own #3

{

Properties {

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

\_MainTex("Diffuse Texture", 2D) = "white" {}

}

SubShader {

Pass{

Tags{"LightMode" = "ForwardBase"}

CGPROGRAM

#pragma vertex vertexFunction

#pragma fragment fragmentFunction

//user defined variables

uniform float4 \_Color;

uniform sampler2D \_MainTex;

uniform float4 \_MainTex\_ST;

//unity defined variables

uniform float3 \_LightColor0;

//input struct

struct inputStruct

{

float4 vertexPos : POSITION;

float3 vertexNormal : NORMAL;

float4 textureCoord : TEXCOORD0;

};

//output struct

struct outputStruct

{

float4 pixelPos: SV\_POSITION;

float4 colour : COLOR;

float4 tex: TEXCOORD0;

};

//vertex program

outputStruct vertexFunction(inputStruct input)

{

outputStruct toReturn;

float3 lightDirection;

//Pulling the ambient Light from UNITY

float3 ambientLight = UNITY\_LIGHTMODEL\_AMBIENT.rgb;

//Get the direction of the light from unity, and normalize it!

lightDirection = normalize(\_WorldSpaceLightPos0.xyz);

//Grab the normal from the input

float3 tempNorm = input.vertexNormal;

//Convert normal from World to Object space

float4 objNorm = mul(float4(tempNorm, 1.0), \_World2Object);

//Normalize the normal

float3 normalizedNormal = normalize(objNorm).xyz;

// light color \* color of object \* dot product between light and direction of normal

float3 diffuseReflection = \_LightColor0.xyz \* \_Color.rgb \* max(0.0, dot(normalizedNormal, lightDirection));

//Calc final light

float3 finalLight = diffuseReflection + ambientLight;

//Output the Colour

toReturn.colour = float4(finalLight, 1.0);

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

toReturn.tex = input.textureCoord;

return toReturn;

}

//fragment program

float4 fragmentFunction(outputStruct input) : COLOR

{

float4 tex = tex2D(\_MainTex, \_MainTex\_ST.xy \* input.tex.xy + \_MainTex\_ST.zw);

return float4(tex.rgb \* input.colour.rgb, 1.0);

}

ENDCG

}

}

//Fallback

//FallBack "Diffuse"

}