Owen Perry 21905958

GPU Programming & Shaders-20S2

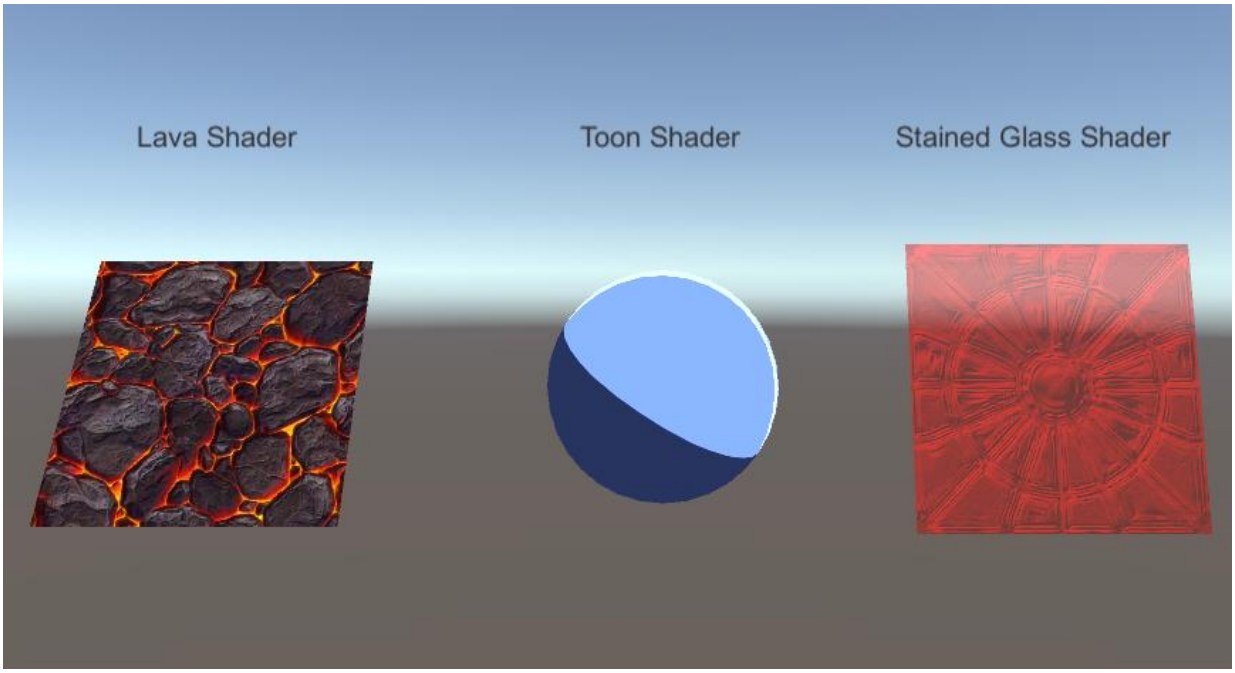
CO569

CW1

Design and implement Shaders to produce a specific effect.

Submission Date: 26/03/2021

Module Leader: Guy Walker

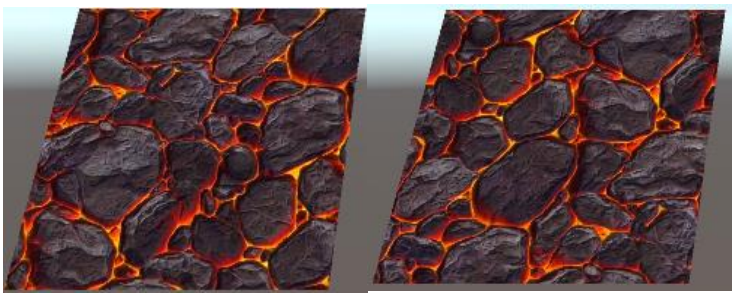


This piece was submitted on 30th March 2021 in accordance with the No Detriment Policy

Shader 1 – Lava Shader

Brief description: A surface shader that displays a lava texture that is repeating which moves over time to simulate the flow of the liquid, additionally it utilises a normal map to achieve more depth.

Image:



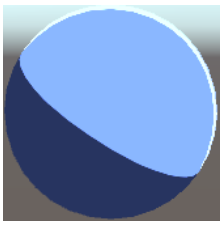
Code:

```
Shader "Custom/LavaShader" {
Properties {
        _Texture ("Base (RGB)", 2D) = "white" {}                        //Defines _Texture property
        _ScrollXSpeed ("Scroll X Speed", Range(0, 10)) = 2              //Defines _ScrollXSpeed
property
        _ScrollYSpeed ("Scroll Y Speed", Range(0, 10)) = 2              //Defines _ScrollYSpeed
property
        _ReversedScroll ("Reversed Scroll", Range(-1, 1)) = 1           //Defines _ReversedScroll
property

        _NormalMap ("Normal Map", 2D) = "bump" {}                       //Defines _NormalMap property
       _NormalMapIntensity ("Normal Map Intensity", Range(0, 2)) = 1    //Defines _NormalMapIntensity
property
    }
    SubShader {
        CGPROGRAM   //Shader code is contained blow this
        #pragma surface surf Lambert //Defines the Lambert lighting model

        sampler2D _Texture;     //Initializes the _Texture variable
        float _ScrollXSpeed;    //Initializes the _ScrollXSpeed variable
        float _ScrollYSpeed;    //Initializes the _ScrollYSpeed variable
        float _ReversedScroll;  //Initializes the _ReversedScroll variable

        sampler2D _NormalMap;       //Initializes the _NormalMap variable
      float _NormalMapIntensity;  //Initializes the _NormalMapIntensity variable

        struct Input {
            float2 uv_Texture;      //Defines the uv coordinates variable for _Texture to use
            float2 uv_NormalMap;    //Defines the uv coordinates variable for _NormalMal to use
        };

        void surf (Input IN, inout SurfaceOutput o) {   //Initializes the surf function
            fixed2 scrolledUV = IN.uv_Texture;  //Assigns the uv coordinates of _Texture to the
scrolledUV variable

            fixed xScrollValue = _ScrollXSpeed * (_Time * _ReversedScroll); //Sets the scroll speed of
the X axis
            fixed yScrollValue = _ScrollYSpeed * (_Time * _ReversedScroll); //Sets the scroll speed of
the Y axis
            scrolledUV += fixed2(xScrollValue, yScrollValue);   // Adds the new scrolled coordinates
to the scrolledUV variable

            half4 c = tex2D (_Texture, scrolledUV); //Assigns the texture with the scrolled
coordinates as a second parameter to the variable "c"
            o.Albedo = c.rgb;   // assigns the .rgb of "c" to the .Albedo of o (the output)

            float3 normalMap = UnpackNormal(tex2D(_NormalMap, scrolledUV)); //Unpacks the normal map
and assigns it to a float3, also takes the scrolled uv as a second argument
         normalMap = float3(normalMap.x * _NormalMapIntensity, normalMap.y * _NormalMapIntensity,
normalMap.z);  //Calculates the float3 of the normal map with its intensity multiplying the x and y
axis and assigning it to the "normalMap" variable
         o.Normal = normalMap.rgb;   //Assigns the .rgb of the "normalMap" variable to the .Normal of
the o (output)

        }
        ENDCG     //Shader Code is contained above this
}        }
```

Shader 2 – Toon Shader

Brief description: A shader that utilizes lots of smoothstep functions to create a “toon” type of aesthetic.

Image:



Code:

```
Shader "Custom/ToonShader 1"
{
    Properties
    {
        _Color("Color", Color) = (0.5, 0.65, 1, 1)                 //Defines the _Color property
        _MainTex("Main Texture", 2D) = "white" {}                  //Defines the _MainTex property
        [HDR]  //Enables HDR
        _AmbientColour("Ambient Color", Color) = (0.4, 0.4, 0.4, 1)       //Defines the _AmbientColour
property
        _RimColor("Rim Color", Color) = (1,1,1,1)                  //Defines the _RimColor property
        _RimAmount("Rim Amount", Range(0, 1)) = 0.716               //Defines the _RimAmount property
        _RimThreshold("Rim Threshold", Range(0, 1)) = 0.1           //Defines the _RimThreshold property
    }
    SubShader
    {
        Pass
        {
            Tags { "LightMode" = "ForwardBase" "PassFlags" = "OnlyDirectional" }  //Requests lighting
data and restricts it to the main directional light only
            CGPROGRAM               //Start of the shader code
            #pragma vertex vert          //Defines the vertex function
            #pragma fragment frag      //Defines the fragment function
            #include "UnityCG.cginc"   //Provide the "TransformTex" function used on line 49

            struct appdata //Defines the appdata structure
            {
                float4 vertex : POSITION;  //Accesses the vertext position
                float4 uv : TEXCOORD0;     //Accesses the first texture coordinate
                float3 normal : NORMAL;        //Accesses the normals within appdata
            };

            struct v2f    //Defines the v2f structure
            {
                float4 pos : SV_POSITION;      //Accesses the position of a vertex after being transformed
into projection space
                float2 uv : TEXCOORD0;        //Accesses the first texture coordinate
                float3 worldNormal : NORMAL;    //Accesses the normals within the v2f structure
                float3 viewDir : TEXCOORD1;        //Accesses the second set of texture coordinates
            };

            sampler2D _MainTex;        //Initializes the _MainTex variable
            float4 _MainTex_ST;        //Initializes the _MainTex_ST variable

            v2f vert (appdata v)    //Vertex function
            {
                v2f o;                                          //Defines the output of the struct
                o.pos = UnityObjectToClipPos(v.vertex);                //Defines the position of the
output
                o.uv = TRANSFORM_TEX(v.uv, _MainTex);              //Defines the uv coordinates of the
output
                o.worldNormal = UnityObjectToWorldNormal(v.normal);        //Defines the world normal of
the output
                o.viewDir = WorldSpaceViewDir(v.vertex);            //Defines the view direction of the
output
                return o;                                       //Returns the output
            }

            float4 _Color;          //Initializes the _Color variable
            float4 _AmbientColour;     //Initializes the _AmbientColour variable
            float4 _RimColor;         //Initializes the _RimColor variable
            float _RimAmount;         //Initializes the _RimAmount variable
            float _RimThreshold;       //Initializes the _RimThreshold variable
```

```
        float4 frag (v2f i) : SV_Target    //Fragment function
        {
            float3 normal = normalize(i.worldNormal);                                   //Normalizes the
worldNormal into a float3
            float NdotL = dot(_WorldSpaceLightPos0, normal);                            //Defines the
NdotL float that is assigned the dot product of the world light position and the normal just defined
            float lightIntensity = smoothstep(0, 0.01, NdotL);                          //Defines the
light intensity and assigns it the smoothstep function of the NdotL float
            float3 viewDir = normalize(i.viewDir);                                     //Defines the viewDir
float3 and assigns it the normalized vector of i.viewDir
            float4 rimDot = 1 - dot(viewDir, normal);                                   //Defines the rimDot
float4 and assigns it 1 - the dot product of the view direction and the normalized worldNormal
            float rimIntensity = rimDot * pow(NdotL, _RimThreshold);;                   //Defines
rimIntensity float calculated by rimDot with the pow function of NdotL and the _RimThreshold var
            rimIntensity = smoothstep(_RimAmount - 0.01, _RimAmount + 0.01, rimIntensity);
//Smoothsteps the rim intensity to create the light outline around the sphere
            float4 rim = rimIntensity * _RimColor;                                     //Defines the rim
float4 that incorporates the desired intensity into the colour calculation
            float4 sample = tex2D(_MainTex, i.uv);                                     //_MainTex is sampled
            return _Color * sample * (_AmbientColour + lightIntensity + rim);           //Returns the
final colour consisting of desired colour multiplied by the sample, multiplied again by the added
values of...
                                                                                      //the ambient colour
(for the lower shadow) the light intensity, and the rim (for the edge light)
        }
        ENDCG     //End of the shader code
    }
  }
}
```

Shader 3 – Stained Glass Shader

Brief description: A shader that utilizes the alpha channel with colours to create a tinted glass effect, along with a normal map to give it the depth and distinctive shapes found on stained glass panes.

Image:

Code:

```
Shader "Custom/StainedGlassShader"
{
    Properties {
        _Colour ("Colour", Color) = (1,1,1,1)                       //Defines the _Colour property
        _Transparency ("Transparency", Range(0, 1)) = 1               //Defines the _Transparency
property
        _NormalMap ("Normal Map", 2D) = "bump" {}                    //Defines the _NormalMap property
        _NormalMapIntensity ("Normal Map Intensity", Range(0, 2)) = 1    //Defines the
NormalMapIntensity property
    }
    SubShader {
        Tags{ "Queue"="Transparent" "RenderType"="Transparent"}       //Sets the appropriate tags to
enable alpha blending
        Blend SrcAlpha OneMinusSrcAlpha                          //Sets the alpha blending method
        ZWrite Off                                      //Disables ZWrite as it is not needed for
transparent materials and can cause compilation issues when active in them

        CGPROGRAM                          //Start of the shader code
        #pragma surface surf Lambert alpha    //Sets the Lambert lighting model and includes "alpha" to
enable the use of the alpha channel

        sampler2D _NormalMap;            //Initializes the _NormalMap variable
        float _NormalMapIntensity;        //Initializes the _NormalMApIntensity variable
        float4 _Colour;                    //Initializes the _Colour variable
        float _Transparency;             //Initializes the _Transparency variable

        struct Input {           //Defines the input structure of the shader
            float2 uv_NormalMap;   //Initializes the uv_NormalMap variable within the input structure of
the shader
        };                    //End of the input structure

        void surf (Input IN, inout SurfaceOutput o) {     //Defines the surf function that provides an
output for the shader
            o.Albedo = _Colour.rgb;        //Sets the Albedo of the output to the rgb value of the _Colour
variable
            o.Alpha = _Transparency;    //Sets the Alpha of the output to the float value of the
_Transparency variable

            float3 normalMap = UnpackNormal(tex2D(_NormalMap, IN.uv_NormalMap)); //Unpacks the normal map
and assignes it to a float3, takes the uv var as a second argument
            normalMap = float3(normalMap.x * _NormalMapIntensity, normalMap.y * _NormalMapIntensity,
normalMap.z);  //Calculates the float3 of the normal map with its intensity multiplying the x and y
axis and assigning it to the "normalMap" variable
            o.Normal = normalMap.rgb;    //Assigns the .rgb of the "normalMap" variable to the .Normal of
the o (output)
        }                                       //End of the surf function
        ENDCG  //Shader code ends here
    }
}
```