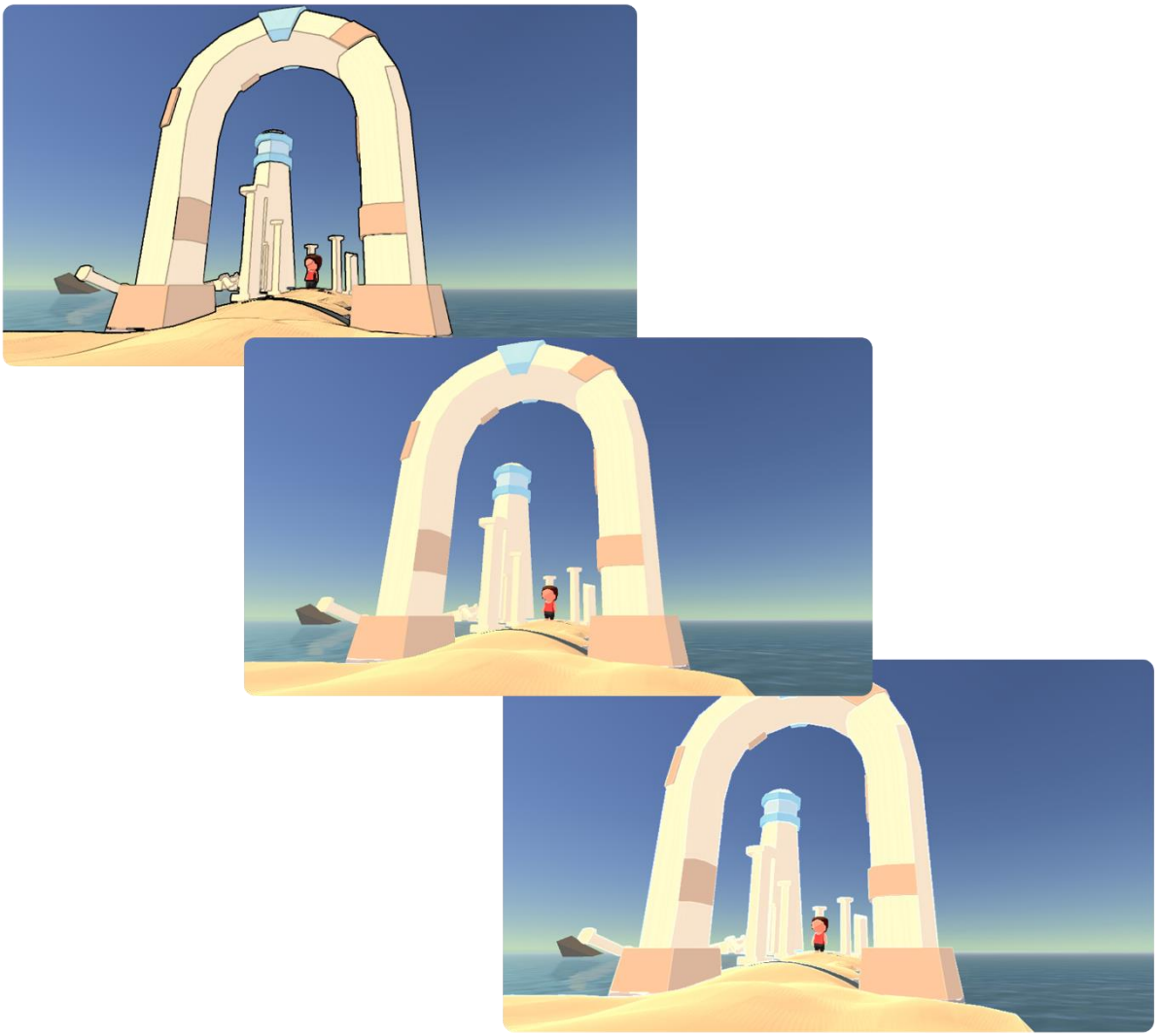
Shaders

Toon & Water



Date: 08.09.2021
Name: Luca Ruiters
Student nr: 500796991
GitHub: ru1t3rl/GPE

Table of Contents

Notes and sketches can be found in the Files chapter

Intro

I really like the effect shaders can have on games. Previous semester, during the VR minor I started experimenting with Shader Graph and enjoyed it a lot. This experience will help with writing normal shaders. Which I have never done before.

Toon Shader

Before I started, I made of list of properties of a toon shader. For example, you need to be able to change the shadow color, the number of shadow "rings" and a custom specular. Eventually, I came across this wiki page which had a very explanation on how to make this effect (Wikibooks, 2020).

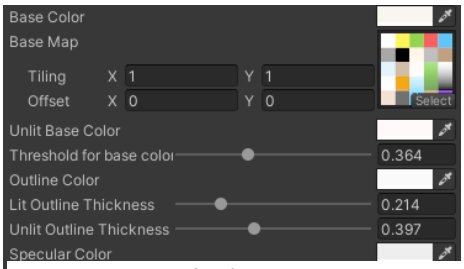


Figure 2 - Toon Shader Inspector

Diffuse

At the beginning of the shader, I first calculate the light direction based on the "Main" directional light's position. The light direction will later be used for specular highlights and shadows.

```
if(_WorldSpaceLightPos0.w == 0.0) {
    // Directional Light
    attenuation = 1.0;
    lightDir = normalize(_WorldSpaceLightPos0.xyz);
    } else {
    // Point or Spot Light
    float3 vertexToLightSource = _WorldSpaceLightPos0.xyz - i.posWorld.xyz;
    float distance = length(vertexToLightSource);
    attenuation = 1.0/ distance;
    lightDir = normalize(vertexToLightSource);
}
```

Figure 3 - Calculate the light direction and attenuation base on the "Main" light's position

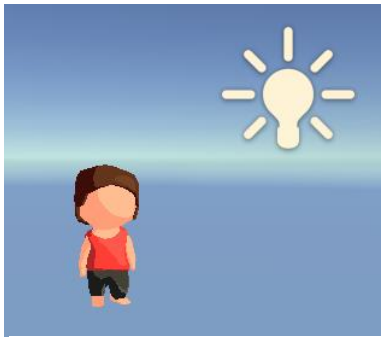


Figure 1 - Toon Shading in action (With support for multiple light source)

To get the correct diffuse color, I multiply the "Main" light's color with the base color.

```
// default: unlit
float3 fragmentColor = _UnlitColor.rgb;

// Low priority: diffuse illumination
if(attenuation * max(0.0, dot(normalDir, lightDir)) >= _DiffuseThreshold) {
    fragmentColor = _LightColor0.rgb * _BaseColor.rgb;      You, seconds ago
}
```

Figure 4 - Calculate the fragments color based on the light color and base color

Specular Highlights

Calculate the specular color base on the light color and out specular tint. The alpha of the specular tint will determine the intensity of the specularity. In a later version of the shader, I have added a variable called "_LightColorInfluence". This variable will change the influence the light color has on all the calculations. The shader also contains an outline option, most of the time I use this as an extra contrast instead of an outline. To support multiple light sources, I transferred the shader code to a .cginc file and include that in both the base and add pass.

```
// highest priority: highlights
// first check if the light source is on the right
// and if it has more than half light intensity
if(dot(normalDir, lightDir) > 0.0 &&
attenuation * pow(max(0.0, dot(reflect(-lightDir, normalDir), viewDir)), _Smoothness) > 0.5) {
    fragmentColor = _SpecColor.a * _LightColor0.rgb * _SpecColor.rgb
    + (1.0 - _SpecColor.a) * fragmentColor;
}
```

Water Shader

The water shader consists out of multiple parts, a color/gradient, reflection, foam, flow and refraction part. I have used a couple of sources to reach the result. I will link them below and throughout the text.

Color

I have chosen to use two separate colors, one for deep water and the other for shallow water. I will lerp between those colors using a depth value. To get this depth value, I use the CameraDepthTexture and use it to calculate the distance between the depth and the current vertex position.

```
// Get depth behind current pixel, linear depth
float depth = tex2Dproj(_CameraDepthTexture, UNITY_PROJ_COORD(i.screenPos)).r;
depth = LinearEyeDepth(depth);

// depthDelta between water surface and pixel behind
float depthDelta = depth - i.screenPos.w;

// Set color based on depth
fixed4 c = lerp(_ShallowWater, _DeepWater, saturate(depthDelta / _Depth));
```

Flow

```
float3 FlowUVW (float2 uv, float2 flowVector, float2 jump,
                float flowOffset, float tiling, float time, bool flowB) {
    float phaseOffset = flowB ? 0.5 : 0;
    float progress = frac(time + phaseOffset);
    float3 uvw;
    uvw.xy = uv - flowVector * (progress + flowOffset);
    uvw.xy *= tiling;
    uvw.xy += phaseOffset;
    uvw.xy += (time - progress) * jump;
    uvw.z = 1 - abs(1 - 2 * progress);
    return uvw;
}
```

Figure 6 - Calculate UV's based on a flow vector and time

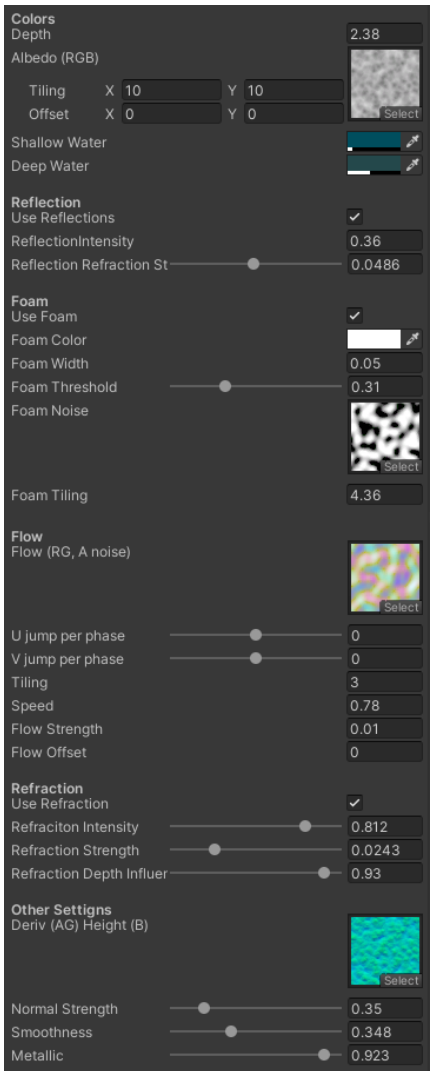


Figure 5 - Water shader inspector

Reflection & Refraction

The reflection and refraction are set using the “finalNormal” offset. Doing so will make sure the refraction moves equal to the ripples/waves.

```
if(_UseReflections == 1) {
    // Set through a script made by Unity
    float4 reflection = tex2D(_WorldReflectionTexture, i.screenPos.xy/i.screenPos.w + (finalNormal.xy * _ReflectionRefract
    reflection = lerp(c, reflection * c.a, c.a);
    c += (reflection * _ReflectionIntensity);
}

if(_UseRefraction == 1)
{
    // Calculate UV offset based on the normal's
    float2 uvOffset = finalNormal.xy * _RefractionStrength;

    float2 uv;
    if(depthDelta/_Depth > 0) {
        uv= i.screenPos.xy / i.screenPos.w + uvOffset;
        } else {
        uv = i.screenPos.xy / i.screenPos.w;
    }
    float4 refraction = tex2D(_GrabTexture, uv).rgba;

    c += (lerp(refraction * (1 - c.a), c, saturate(depthDelta/_Depth) * _RefractionDepthInfluence) * _RefractionIntensity)
}
```

Figure 7 - Reflection and Refraction

```
// Flow vectors
float3 flow = tex2D(_FlowMap, i.uv_MainTex).rgb;
flow.xy = flow.xy * 2 - 1;
flow *= _FlowStrength;
float noise = tex2D(_FlowMap, i.uv_MainTex).a;
float time = _Time.y * _Speed + noise;
float2 jump = float2(_UJump, _VJump);

// Calculate uv's
float3 uvwA = FlowUVW(i.uv_MainTex, flow.xy, jump, _FlowOffset, _Tiling, time, false);
float3 uvwB = FlowUVW(i.uv_MainTex, flow.xy, jump, _FlowOffset, _Tiling, time, true);

// Unpack normal's/height's
float3 dhA =
UnpackDerivativeHeight(tex2D(_DerivHeightMap, uvwA.xy)) * (uvwA.z * _NormalIntensity);
float3 dhB =
UnpackDerivativeHeight(tex2D(_DerivHeightMap, uvwB.xy)) * (uvwB.z * _NormalIntensity);
float3 finalNormal = normalize(float3(-(dhA.xy + dhB.xy), 1));

// Map the main texture
fixed4 texA = tex2D(_MainTex, uvwA.xy) * uvwA.z;
fixed4 texB = tex2D(_MainTex, uvwB.xy) * uvwB.z;
fixed4 finalTex = texA + texB;
```

Figure 8 - Linked the displaced UV to the main texture and normal map

Post Processing Outline

Since I made the toon shader, I figured it would be nice to have some form of an outline effect. To do this I decided to go with an image effect (3x3 kernel). This was partially based on an example Daniël showed us.

To get started, I opened the Wikipedia page about the image kernels and used that in combination with a tutorial to make the image effect (Ilett, 2019).

Using this effect will online show the edges of your scene. I wrote this simple formula to get the color back.

```
float4 col = tex2D(_MainTex, i.uv);
return col + col * fixed4(sobel(i.uv) * _LineColor.rgb, 1.0);
```

```
float3 sobel(float2 uv)
{
    float x = 0;
    float y = 0;

    float2 texelSize = _MainTex_TexelSize;

    x += tex2D(_MainTex, uv + float2(-texelSize.x, -texelSize.y)) * -1.0;
    x += tex2D(_MainTex, uv + float2(-texelSize.x,            0)) * -2.0;
    x += tex2D(_MainTex, uv + float2(-texelSize.x,  texelSize.y)) * -1.0;

    x += tex2D(_MainTex, uv + float2( texelSize.x, -texelSize.y)) *  1.0;
    x += tex2D(_MainTex, uv + float2( texelSize.x,            0)) *  2.0;
    x += tex2D(_MainTex, uv + float2( texelSize.x,  texelSize.y)) *  1.0;

    y += tex2D(_MainTex, uv + float2(-texelSize.x, -texelSize.y)) * -1.0;
    y += tex2D(_MainTex, uv + float2(           0, -texelSize.y)) * -2.0;
    y += tex2D(_MainTex, uv + float2( texelSize.x, -texelSize.y)) * -1.0;

    y += tex2D(_MainTex, uv + float2(-texelSize.x,  texelSize.y)) *  1.0;
    y += tex2D(_MainTex, uv + float2(           0,  texelSize.y)) *  2.0;
    y += tex2D(_MainTex, uv + float2( texelSize.x,  texelSize.y)) *  1.0;

    return sqrt(x * x + y * y) * _LineIntensity;
}
```

Figure 11 - Sobel kernel transformation

Conclusion

I first started the water shader as a surface shader. This mainly had to do with my method for reflections doing weird stuff in a vertex shader. But during development and research I found another method to do reflections which worked better and in a converted version of the shader (surface to vertex).

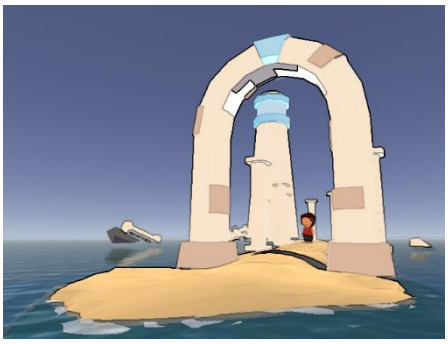


Figure 9 - Black Outline

Figure 10 - White Outline

References

A. (2017, August 14). Unity Refractive Shader. Creative Tinkering. http://tinkering.ee/unity/asset-unity-refractive-shader/

Alisavakis, H. (2020). My take on shaders: Stylized water shader – Harry Alisavakis. H. Alisavakis. https://halisavakis.com/my-take-on-shaders-stylized-water-shader/

Cg Programming/Unity/Toon Shading - Wikibooks, open books for an open world. (2020, April 16). Wikibooks. https://en.wikibooks.org/wiki/Cg_Programming/Unity/Toon_Shading

Flick, J. (2018a, May 30). Texture Distortion. Cat Like Coding. https://catlikecoding.com/unity/tutorials/flow/texture-distortion/

Flick, J. (2018b, August 30). Looking Through Water. Cat Like Coding. https://catlikecoding.com/unity/tutorials/flow/looking-through-water/

How do I calculate UV space from world space in the fragment shader? (2016, August 29). Game Development Stack Exchange. https://gamedev.stackexchange.com/questions/129139/how-do-i-calculate-uv-space-from-world-space-in-the-fragment-shader

How to calculate screen space texcoords for surface shader? - Unity Answers. (2017, September 3). Unity Form. https://answers.unity.com/questions/1402029/how-to-calculate-screen-space-texcoords-for-surfac.html

Ilett, D. (2019, May 11). Image Effects | Part 4 - Edgy Talk. Daniel Ilett: Games | Shaders | Tutorials. https://danielilett.com/2019-05-11-tut1-4-smo-edge-detect/