Goldsmiths, University of London

CM3045 – 3D Graphics & Animation

Peer-Graded Assignment: First GPU Shader

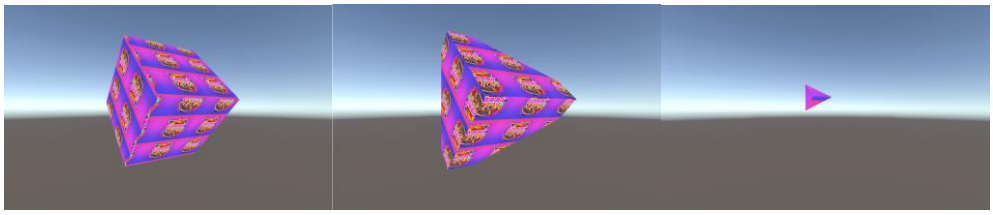
Report by Hristo Stantchev

This report covers my submission for the "First GPU Shader" Peer-Graded Assignment. This submission implements a fragment shader, as well as a vertex shader. Vertex shaders apply each function on each vertex (3D Point) whilst Fragment shaders apply functions on each fragment (not to be confused with pixels, as pixels cover screen space). It Implements 2 tasks which work simultaneously.

Overview

The scene features a cube with an animated tiled texture. Furthermore, the cube deforms on several axes, creating the appearance of constant shrinking and expansion.

Screenshots:



Task 1: Vertex Shading, **Task 2:** Animated Values

This solution deforms the object based on each axis (X, Y, Z). This is done through a vertex function:

```
v2f vert (appdata v)
{
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.vertex.xyz += o.vertex * _AnimateXY.xyz * clamp(sin(_Time.y),0,1);
```

Using a vertex-to-fragment data type in a vertex function, the object vertexes are put to a clipping space position using the in-built Unity function. Then, the vertex xyz positions are taken and added to using a custom made float4's XYZ positions. These define the margins of vertex manipulation. This is then multiplied by using a clamped sine of the elapsed time (Unity-provided attribute).

Clamping avoids negative vertex values and leaves some seconds for the shape to remain at maximum and at minimum size. This code was adapted and modified from Alan Zucconi's [1] Normal Extrusion method tutorial.

Extension Task: Fragment + Vertex Shading; Correlated

As mentioned above, the fragment shader moves tiled images at XY values. This is done partly through the vertex and the fragment functions of the pass:

```
sampler2D _MainTexture;
float4 _MainTexture_ST;
float4 _AnimateXY;
float4 _MinVert;
float4 _MaxVert;

v2f vert (appdata v)
{
    v2f o;
    o.vertex = UnityObjectToClipPos(v.vertex);
    o.vertex.xyz += o.vertex * _AnimateXY.xyz * clamp(sin(_Time.y),0,1);
    o.uv = TRANSFORM_TEX(v.uv, _MainTexture);
    o.uv += frac(_AnimateXY.xy *_MainTexture_ST.xy *_Time.yy);
    return o;
}

fixed4 frag (v2f i) : SV_Target
{
    float2 uvs = i.uv;
    fixed4 textureColor = tex2D(_MainTexture, uvs);
    return textureColor;
}
```

As shown, the output vector-to-fragment takes its UV from the appdata as well as the texture specified in the inspector. This is done through the built-in function “TRANSFORM_TEX”. Then, each fragment of the UV map is being given new XY values from the custom Animate XY. They dictate animation speed on each axis. This is multiplied by the tiling XY values provided by “_MainTexture_ST”. This ensures that the animation runs at the same speed regardless of the tiling count. It is finally multiplied by time once again to create the animation. Finally, that is wrapped in a fraction function to solve texture pixilation during animation. Inside the fragment function, the UVs and the main texture are pushed into a tex2D function that is then returned as the fragment’s color. This code is taken from Benjamin Swee’s Unity tutorial: “Writing Unity Shaders Texture Sample, Tiling and Animation | Shader Fundamentals”. [2]

References:

1. Zucconi, Alan, “Surface shaders in Unity3D”, blog, June, 2015
2. Swee, Benjamin, “Writing Unity Shaders Texture Sample, Tiling and Animation | Shader Fundamentals”, Unity CG/HLSL Tutorial, Apr 8, 2022
3. **Texture – Custom made by me for final project**