Technical Design Document

Assignment 4 – Final Project Documentation

INFT575 Preparation of Final Business Case

Chris Murphy (CIT182880)

25 June 2019

Table of Contents

Systems & Component Listing

See Design Document for explanation of the functionality for each component

A note on shader pseudocode:

An important aspect of this project to note is that when developing shaders, finding the correct algorithm for the desired effect is most of the overall work required. In the case of developing shaders for Unity, they are developed in a modular fashion and attached to a material – the 'packaged' nature of this product means that most components function independently and as such the majority of development for this project will be focused on creating these effects rather than building any systems around them.

The low-level nature of shader code means that shader pseudocode cannot be as abstract as most other forms of pseudocode, and that to write accurate shader pseudocode requires most of the effort of creating the shader itself. Since building shaders makes up the majority of the development phase of the project, the only shader pseudocode that has been included in the listing below is that which had already been developed prior to this project starting. To include pseudocode for every shader effect would essentially require 50% of the development phase to already have been completed by this point.

Shaders

- **Vertex jitter**

  input geometric resolution value – the lower the value, the more intense the vertex jitter effect

  in the vertex function

    transform the vertex into view space

    multiply the vertex position by the geometric resolution value

    round each position value to the nearest integer

    divide the vertex position by the geometric resolution value

    transform the vertex into screen space

- **Affine texture mapping**

  input texture tiling and offset parameters

  disable the default texture perspective correction setting

  store vertex texture coordinates as a 3D rather than 2D vector

  in the vertex function

    transform the vertex position into screen space

    create 2D vector to store transformed vertex texture coordinates according to tiling and offset parameters

    set vertex texture coordinate u as transformed texture coordinate u multiplied by screen space vertex position w

    set vertex texture coordinate v as screen space vertex position w

    set vertex texture coordinate z/third value as screen space vertex position w

in the fragment function

set final texture coordinate of fragment as input fragment texture coordinate u & v divided by input fragment coordinate z

get fragment texture colour using final texture coordinate

- **Transparency**
- **Fog**
- **Animated textures**
- ***Sprite shader***

Lighting types

- **Flat shading**
- **Gouraud shading**
- **Phong shading**
- **Unity shading**

Post-processing shaders

- **Pixelation**
- **Posterization/colour banding**

input colour resolution value – the lower the value, the more intense the posterization effect

in the fragment function

multiply each colour value by the colour resolution value

round each colour value to the nearest integer

divide each colour value by the colour resolution value

output the colour of the fragment

- **Dithering**
- **Chromatic aberration**

input chromatic aberration intensity value – the higher the value, the more intense the chromatic aberration effect

in the fragment function

set red colour value as red colour value of rendered screen texture moved the distance of the chromatic aberration intensity to the left

set blue colour value as blue colour value of rendered screen texture

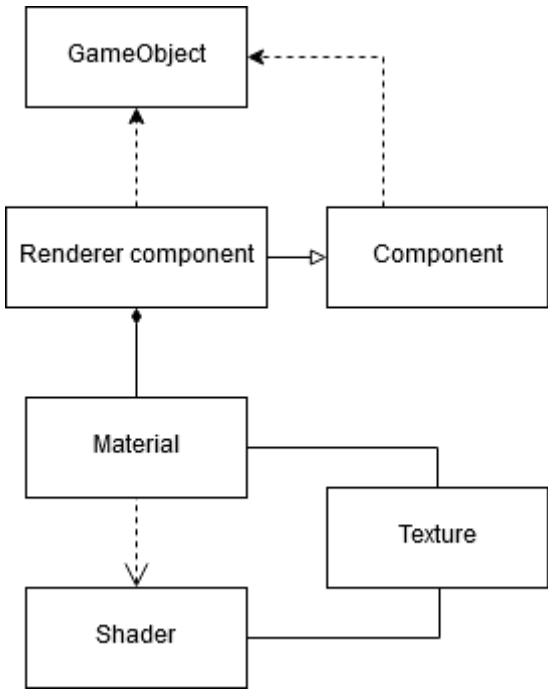
set green colour value as green colour value of rendered screen texture moved the distance of the chromatic aberration intensity to the right

output the combined colour of the fragment

- **Vignetting**
- **Blur**

Shader implementation

To render a GameObject in Unity with a visible mesh the engine makes use of several classes. First is the Renderer class, which is derived from the Component class and tells the engine that the gameobject is to be rendered by the scene camera. In order to be visible, the Renderer component must have a Material assigned, with the exposed variables in the inspector window being defined by a Shader that is core component of the material. The material can also have one or more Textures assigned, which can also be accessed by the Shader for rendering.



A basic high-level depiction of the relationships between the classes used to render a mesh in Unity

At this point, the most efficient and effective method of implementation for the shader effects in the package will likely be to include all of them in two large Unity shader files (retro shader and post-processing retro shader), as opposed to splitting each effect into its own separate shader file. This will make them as easy as possible to use – for example, if the user wants a mesh to have flat shading, vertex jitter, and affine texture mapping applied than they can simply apply the retro shader to the mesh material and use the in-built Unity inspector panel to adjust the exposed shader variables so that these are the only three effects that are active. This is the method that is also used by both *PSXEffects* and *Retro Look Pro*.

Barring unforeseen consequences, this also prevents the need to develop a custom Unity editor extension, which leaves more development time available to work on the effects themselves.

Low-res camera prefab

A non-shader asset to be included as part of the product is a low-res camera prefab is comprised of two cameras, a script, and a screen canvas and used to adjust the resolution of the screen image displayed to the viewer. It functions as follows:

input screen canvas vertical resolution

when the script is initialised or the screen canvas vertical resolution is changed

a render texture is created with the vertical resolution specified and applied to the screen canvas

every frame

the 'world' camera renders the scene as normal and outputs the rendered image to the render texture being displayed on the screen canvas

the 'screen' camera can only see the screen canvas which is positioned to fill the entire camera view – it renders the low-res render texture and displays this image on the game window to the player

References

- Kosman, C, 2019, *PSXEffects*, Unity Asset Store, viewed 04/05/19, *https://assetstore.unity.com/packages/vfx/shaders/psxeffects-132368*
- Limitless Unity Development, 2019, *Retro Look Pro*, viewed 04/05/19, https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/retro-look-pro-133193