Design Document

Assignment 4 – Final Project Documentation

INFT575 Preparation of Final Business Case

Chris Murphy (CIT182880)

25 June 2019

Table of Contents

Overview

The Retro 3D VFX Pack for Unity will allow the user to imitate the various visual quirks which create the distinct visual style of early 3D games, such as those on the original PlayStation. Each included effect will be modular and scalable, as well as being able to be effectively utilised by both programmers and non-programmers alike. The intended audience is indie developers without the experience, time, or resources to be able to achieve this visual style on their own.

Functionality

Assets

Lighting types:

The user will be able to select which of these four lighting models is applied on a per-mesh basis to allow for greater development flexibility.

- **Flat shading** – the oldest technique of the included lighting models in which each pixel of every face of a polygon uses the same colour. This gives a sharp, flat look to a polygon in which each face is clearly distinct.
- **Gouraud shading** – the colour value of each vertex is calculated using a normal, then the colours of each vertex that make up a face are interpolated across the face to give it it's colour. Looks slightly more realistic than flat shading while still being distinctly retro.
- **Phong shading** – the most modern of the three custom lighting models. Uses a combination of ambient, diffuse, and specular reflection values assigned to a material in order to more accurately simulate how real-world materials reflect light.
- **Unity shading** – the package will also support the in-built lighting system included in the Unity engine by default.

Shaders:

In the standard Unity fashion these shader effects will be used by selecting them to render a material which is then applied to the appropriate mesh or sprite.

- **Vertex jitter** – this is accomplished by quantizing the world-space position of each vertex to imitate the same effect which would occur in older 3D rendered models. The effect was caused because the available precision of the position value for each vertex was much lower than nowadays.
- **Affine texture mapping** – cancels perspective correction for texture mapping so that as the camera moves closer to an angled surface, the texture will warp. This method of texture mapping was used on the original PlayStation before hardware became powerful enough to use mapping techniques that included perspective correction as part of the process.
- **Transparency** – the shaders included in the package will have support for transparent materials & vertex colours.
- **Fog** – the shaders included in the package will have support for Unity's inbuilt fog system.
- **Animated textures** – animated materials are a useful tool for visually implying that a mesh is moving without having to transform the mesh at all. As such, including support for animated textures in the shaders of the package will prevent the likely occurrence of the user having to dig into the shaders and implement this feature themselves.
- **Sprite shader** – a shader for 2D sprites that also includes support for all the above

effects (e.g. vertex jitter).

Post-processing shaders:

Rather than being applied to a material which is used to render a mesh, 'post-processing' or 'screen' shaders are used to alter the screen image that has been rendered by the camera.

- **Pixelation** – reduces the number of texels that make up the rendered screen image texture. This is especially useful for helping to build a retro aesthetic, because the 'visual' resolution of the game can be lowered while keeping the 'rendered' resolution the same – the size of the game window itself doesn't have to be changed.
- **Posterization/colour banding** – quantizes the colour values of each texel in the rendered screen image. This creates a distinctive look where instead of colours fading together smoothly, they separate into distinct 'bands' of solid colour. Similar to the vertex jitter effect described above, this appeared in early 3D games because the values used to represent colours on older hardware had much less precision available and thus a much smaller range of colours.
- **Dithering** – an effect that was specifically used to counter the existence of colour banding in older 3D games, it helps to blend colours when a limited range of colour values are available. Additional colour values are approximated by juxtaposing the constituent colours together to trick the human eye, similar to how RGB screens work. Depending on the type of dithering used, this often creates a distinctive 'dotted' pattern.
- **Chromatic aberration** – in this effect, the texel position of each colour value (e.g. the red, green, and blue) shift position in different directions to create 'fringes' of solid colour to simulate a lens which can't focus all the colours to the same point. This isn't strictly a 'retro' effect but is often used as a post-processing effect in games, especially to simulate looking through a camera or video tape.
- **Vignetting** – a reduction of the brightness of an image the closer the texel is to the edge, which creates a circular black border that fades towards the sides of the image. Often used to draw attention to the centre of the screen.
- **Blur** – in this case, Gaussian blur is the most likely technique that will be used for this effect.

Prefabs:

- **Low-res camera** – a prefab which is comprised of two cameras, a script, and a low-resolution render texture and used to adjust the resolution of the screen image displayed to the viewer.

User documentation

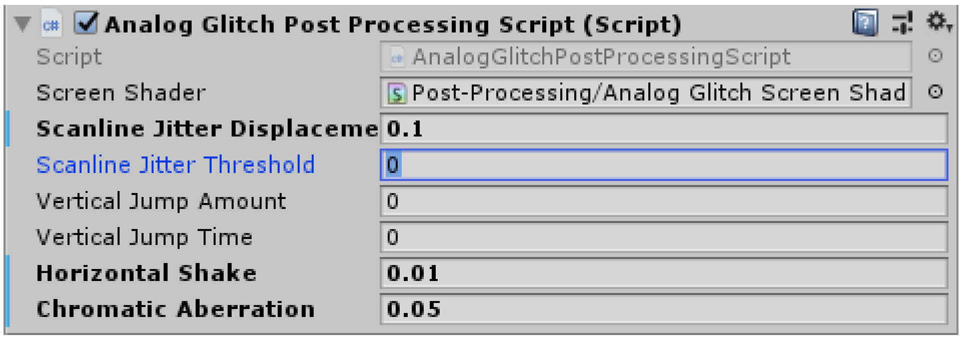
In addition to extensive in-code commenting to provide support for programmers, the package will include external documentation files that explain how each asset works as well as how it can be effectively used.

Example scene

Alongside the user documentation, the example scene will be developed for a practical demonstration of how each of the package components can be implemented within a Unity scene. Similar to the free demo scene provided on the store page for Christian Kosman's *PSXEffects* pack, the player will be able to fly through the scene and view visual examples of each included effect in action. It will also include a UI menu that allows the player to modify the effects in real time using sliders and checkboxes.

User Interface

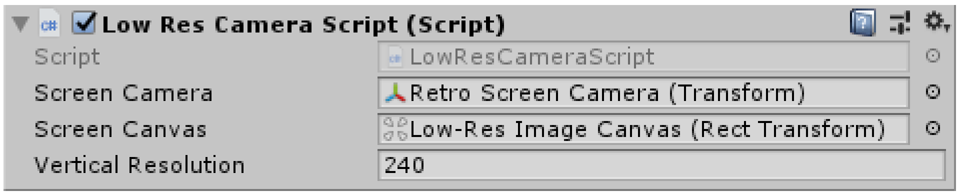
At this point, it seem that the easiest and most effective means of implementing the user interfaces of the package will be to use the default method of exposing variables for adjustment in the Unity Inspector window. As outlined in the Technical Design Document, the two groupings of 'retro shaders' and 'retro post-processing shaders' will likely be grouped into two large shader files – this way, the user can simply assign the appropriate shader to a mesh or camera and use sliders/other controls in the Inspector panel to adjust the effects to their liking. For example, a recent personal project that I've been working on contains an 'analog glitch' post-processing shader which implements 5 different effects in the same file.


Analog Glitch Post Processing Script (Script)

Script	AnalogGlitchPostProcessingScript
Screen Shader	Post-Processing/Analog Glitch Screen Shad
Scanline Jitter Displaceme	**0.1**
Scanline Jitter Threshold	0
Vertical Jump Amount	0
Vertical Jump Time	0
Horizontal Shake	**0.01**
Chromatic Aberration	**0.05**

The above image shows how the variables controlling each of these effects are exposed for easy modification in the Unity Inspector window belonging to their attached camera – they can even be modified by the developer during runtime so that the effects can be seen in action. For the other 'retro shader', it would simply be attached to a mesh rather than a camera. Using a similar implementation for the *Retro 3D VFX Pack* seems like it will be the best course of action, both from a development and a user standpoint. Using Unity's native means of exposing variables could also help protect the package against issues arising from future Unity updates, as opposed to creating a custom window in the editor which could be more likely to run into issues as new versions are released.

This approach of controlling the components within the package through exposed variables in the Unity Inspector window will also extend to any included prefabs, such as the low-res camera prefab.


Low Res Camera Script (Script)

Script	LowResCameraScript
Screen Camera	Retro Screen Camera (Transform)
Screen Canvas	Low-Res Image Canvas (Rect Transform)
Vertical Resolution	240

Milestones

Project documentation: 1 March – 25 June

This is the date by which all project documentation must be completed in order for the development phase of the project to begin. These documents should include:

- Vision & Scope document
- Project Timeframe document
- Learning Plans & Research Journal
- Project Management document
- Design document
- Technical Design document
- Risk Management document
- Test Plan document
- Financial Outline document

Preparing this documentation will ensure that the focus of the project is clearly outlined and that it will proceed as smoothly and efficiently as possible.

Asset development: 26 June – 23 September

By this point all of the assets which make up the package itself will have been built. These are divided into three major groups being the shaders, post-processing shaders, and additional prefabs.

- **Shader development: 26 June – 14 August**
- **Post-processing shader development: 15 August – 13 September**
- **Prefabs: 14 September – 23 September**

This is the section of development with the highest likelihood of falling behind schedule due to the difficulty in accurately estimating the time required for software development tasks. As such, it's important to regularly ensure that the project remains on schedule and if it isn't, to potentially consider removing one or two of the planned assets from the final package.

User documentation: 24 September – 13 October

This segment of development includes both reviewing and improving the existing in-code documentation as well as the creation of the external tutorial documentation for the package.

Example scene: 14 October – 23 October

This phase covers the development of the example/demo scene to be included as part of the package.

Safety/buffer time: 24 October – 10 November

This is a dedicated spare period of safety/buffer time to provide some insurance in case some part of the project development slips past its due date. It will also be used to clean and finalise the package to ensure it's as ready as possible for release onto the Unity Asset Store.

See the 'Project Timeline' image in the documentation folder for a more detailed breakdown of the due dates of specific work tasks.

References

- *Dithering*, 2004, viewed 03/06/19, Web Style Guide, https://webstyleguide.com/wsg2/graphics/dither.html
- *Gouraud shading*, (n.d.), viewed 02/06/19, https://graphics.fandom.com/wiki/Gouraud_shading
- *The Phong Model, Introduction to the Concepts of Shader, Reflection Models and BDRF*, (n.d.), viewed 02/06/19, https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF