COMP220: Graphics & Simulation

8: Post-Processing

Learning outcomes

By the end of this week, you should be able to:

- **Explain** what the framebuffer is and how it can be used to generate 2D effects.
- **Implement** post-processing effects in your application.

Agenda

- Lecture (async):
  - **Introduce** the framebuffer and its uses.
  - **Describe** a variety of 2D post-processing effects.
- Workshop (sync):
  - **Extend** the use of OpenGL textures to store rendered results.
  - **Apply** a selection of post-processing techniques to the rendered texture in a shader.

Post-processing

What is Post-Processing?

- A series of techniques that are carried out **after** a scene has been rendered
- Typically mimic effects caused by camera (or eye) hardware, e.g. lens flare, bloom, motion blur
- Traditionally this could only be done as an offline process
- With the advent of faster GPUs and programmable shaders, we can implement some of the effects in real time

Post-Processing Stages

- The key to the effect is to switch from drawing to the back buffer to a **texture**
- This texture will contain the current view of the scene
- We then use a shader to implement a post-processing effect
- We then map the processed texture onto a full screen quad, which is rendered to the backbuffer

The Frame Buffer

- We use several types of **screen buffers** when rendering with OpenGL:
  - The **colour buffer** stores the pixel output from the fragment shaders.
  - The **depth buffer** (or *z*-buffer) stores information for depth-testing.
  - A **stencil buffer** can be used to mask out certain fragments.
- A **framebuffer** is just a combination of different buffers.
- A default framebuffer is created for you; you can create additional ones to render to instead.

COMP120 Refresher

Image Processing Techniques

- Some of the algorithms you learned in COMP120 can be applied in GLSL:
  - Colour Replacement
  - Luminance calculation
  - Colour Correction
  - Black & White and Sepia Tone
  - Edge Detection
- Key difference: we access the data via **texture coordinates**, rather than array indices.

Modifying Textures

- A Texture Lookup uses the texture coordinates passed to the fragment shader
- Additionally, the fragment shader only processes a single fragment at a time
- If we want to access a pixel adjacent to the current one, we can offset the current texture coordinates - this is especially useful for edge detection
- GLSL has lots of inbuilt functions (e.g distance) which can aid in creating post-processing effects

Other Post Processing Techniques

Colour Correction

- Process of taking an image, convert each colour in the image to a some other colour
- Mimic a specific film stock, provide coherent look or provide a mood

Original Shot

Day-for-Night Color Corrected shot

Colour Correction Again

- Depending on the technique this could involve manipulating the colours using a Filter Kernel
- Or we could use a colour palette (see Colour Grading)

Blur

- This often used as a basis for other techniques such as Depth of Field
- To achieve blurring we apply a filter to the texture lookup of the texture render target
- We can apply many different filters, one of the simplest is a Box Filter
- With a box filter we sample 4 points around the point we are interested in and take an average

Motion Blur

- If an object moves very fast through you Field of view it can appear that the object leaves a slight ghost of itself
- There are several ways of implementing this, we can use the actual speed of the object to determine the amount of blur

Motion Blur

- Or we can simply take the results of the last render update and blend them with current render results and blend based on a lerp

Depth of Field

- DOF attempts to simulate the effect where object that are too close or too far away appear out of focus
- We first have to consider in our scene what range of depth values will be considered in focus. We then need to blur everything that is outside this range

Depth of Field

- Once we have determined this we render a blurred scene to a render texture

Bloom

- This attempts to simulate the overloading of the optic nerve if extremely bright areas of a surface are visible
- First we render to a smaller render target, usually 1/4 size

Bloom

- We then apply a filter (usually Gauss) to this reduced render target, if we apply this filter in multiple passes then we achieve a smoother blur
- We then blend the blurred image onto the screen with the original scene

Post Processing Stack

- In theory we can chain post-processing effects together
- Essentially we keep rendering to a texturing using different fragment shaders each time
- These could be stored in a data structure such as a stack
- Once we have completed the last effect on the stack, we then switch to normal rendering and display the result

Next steps

- **Review** the additional asynchronous material for more background on the framebuffer and various post-processing (or image processing) effects.
- **Attend** the workshop to learn how to save a rendered scene to a texture in OpenGL and apply effects to it in a shader.