CO3303

Assignment Implementation Report

* I have used the Post-processing Area project as a base project for this assignment.

**Post-Processes within a graphics application**

Post-processing is the process of applying effects and filters to the rendered scene before sending presenting the final scene/image to the front-buffer, this is usually done through rendering into intermediate textures. Post-processing can quickly improve a game’s visuals with a variety of customizable filters. Some applications include adding realism to an outdoor environment with depth of field, or providing players feedback by flashing a screen red when they’re damaged (Admin, 2022). Post-processes can add many different effects to the game such as blurs, blooms and field-of-views that enhance the end-product of the game. These effects are done to help immerse the player as they replicate real-life phenomena such as the depth-of-field of a camera lens. Some post-process can sometimes only affect a small area of the screen/scene such as smoke around a fire, it is far less computationally taxing to render an effect to a small area of the screen than to render thousands of smoke particles every frame.

**Basic Requirements:**

* Vertical Colour Gradient post-process.
  + A vertical colour gradient was done through adding the linear interpolation between the two chosen colours with the current UV’s y component.
* Full-screen blur post-process.
  + The first Blur post-processing effect that was implemented was done through down sampling the scene to a texture that was a ¼ the size of the original texture. This new texture was then rendered again to a new texture that was the original size. This final texture was rendered to the back buffer. This blur effect was updated to the two-pass Gaussian that will be explained later in the report.
* Full-screen Underwater post-process
  + The Underwater effect was done by distorting the scene texture with an offset value and then adding the underwater colour tint to this new texture. The offset value was calculated using a sin wave along the X axis multiplied by a strength value entered by the user and the size of the effect. The sin wave is calculated through the intrinsic sin function of the current UV’s x coordinate multiplied by the

**Additional tasks**

* HSL Colour space and gradual changing of the colour in the vertical gradient
  + To update the Gradient post-process effect to the HSL colour space, I first performed the Linear interpolation between the two colours. With this colour value I then converted it to the equivalent HSL value. With this new HSL value I added a timer divided by 10 to get a gradual changing effect. The updated HSL value is then converted back to RGB (as shaders output in the range 0-1) to get the final colour of the pixel.
* Two pass Gaussian Blur
  + For the two pass Gaussian blur I removed the rendering of the scene to a down sampled texture and back up again as I found the results from the Gaussian blur to be more than enough by itself. The scene is rendered to a texture using a horizontal blur and then rendered to another texture with a vertical blur added. This final texture is then rendered to the back buffer to be present. Both blurs are the same with the only difference being is the axis along which the UVs are updated being changed appropriately. I will explain how I implemented the Horizontal pass of the blur. The scene Texture is sampled and the multiplied by a certain blur weight. After this a loop is done through an array of predetermined kernel offsets found in the optimized version of the algorithm. The normalized offset of the current kernel value is found by dividing it by the width of the viewport. The final colour is then updated to be a sample from the scene with a neighbouring pixel from either side.

**Polygon Effects in the wall openings**

* Saturation post-process
  + The saturation post-process was done by getting the luminance value of the current pixel using the dot product of the current pixels colour and a set of user defined weights. A new colour value is calculated by performing a linear interpolation between these two values along a value that can be chosen by the user. The alpha value of the current pixel is then calculated by finding the average of the individual colour components in the output.
* Vignette post-process
  + The vignette post-process effect was done by calculating the distance of the current pixel from the center of the screen and performing a smooth blend between the size of the vignette and the range of the vignette using this distance variable. A linear interpolation was then done between this value and 1 by a user-defined strength. The final colour was then clamped to a value between 0 and 1. This effect created circle in the centre of the screen where it smoothly fades to black around the edges. The user can fully control this effect through Sliders
* Fisheye post-process
  + The position of the pixel is calculated in the range -1 to 1 and then the length of the pixel is calculated to be able to distort the image. To distort the image, the angle between the x and y coordinate is calculated through the inverse tangent. This angle is then used to calculate the radius of the circle. The radius is then raised to a chosen power (this value affects the strength of the fisheye). Once this radius is calculated, the position coordinates are updated to the radius multiplied by the sin or cos of the already calculated angle. This new value is used to sample the scene Texture to get the appropriate colour. If the length of the pixel is greater that 1 the colour is set to black to give the appearance of looking down a telescope.
* Distort post-process
  + The colour from the distort map is used to calculate the direction that would be needed to distort the current UV coordinates. This vector is then converted to the range -0.5 – 0.5 where fake diffuse lighting is calculated based on this. The final colour is calculated by sampling the scene and multiplying it by distort variables before adding it to the fake diffuse light.
* Grey Noise post-process
  + The grey noise post-process was already in the post-processing area lab

**Advanced post-processes**

* Retro Gameboy effect
  + For the Pixellation part, the current pixel’s UV coordinates are scaled up by the required width and height and a floor operation is then performed to set the value to the nearest whole number from the current value. This value is then scaled back down to the original range to get a new value to sample from the scene texture. This is done to ensure that all pixels within each whole number thresholds are given the same colour producing an effect where the pixels appear blockier and larger.
  + For the reduced colour set, I went with the original Gameboy 4 shades of green and calculated the luminance of the scaled-up pixel, by performing a dot product of the pixels colour against a predetermined luminance weight, this luminance value was then used to get the correct shade of green for the pixel.

**Improvements or extensions that I could make:**

* One improvement that I could make would be the optimization of rendering to textures, this could be done through removing unnecessary rendering of the scene to a different texture between the post-processing stages of rendering.
* Another improvement that I could have made was implementing the depth-of-field post-processing effect as I already have the 2-pass Gaussian blur working, and I would only need to implement the focal length
* The third improvement that I would like to have done was fixing the rendering of the polygon post-process effects. During the development of this assignment the polygons began rendering over every other object in the scene, and I tried to fix it within the time period while working on the other post-processes. In the end I decided to leave it be to instead focus on the post-process effects themselves.

Improvements to the Code to add flexibility.

* + The First improvement I implemented was to restructure the file and code to be run through a system class that would control everything, this system class would then create a post-Processing scene through a DirectX window. To help with the restructuring of the code, I implemented a premake file to allow for more flexibility to the creation of the visual studio solution
  + The second improvement I implemented was to create a class for the 2D textures that the rendering pipeline could render the scene to. This hugely increased code readability and allowed for the setting of the render target to be done through the class.
  + The third improvement I made in the code was to add additional functions in the model class to setup all the information the graphics card needs to render a model, such as the pixel or vertex shader, the different states to use and the shader resources.

# References

Admin. (2022, January 4). *What Is Post Processing In Games*. Retrieved from QuestionFun: https://questionfun.com/what-is-post-processing-in-games/