CO3303

Assignment Implementation Report

* I have used the Postprocessing Area project as a base project for this assignment.

**Post Processes within a graphics application**

Graphics shaders are used to tell the computer how to render each pixel on the screen (Shehata, 2015). Vertex shaders get the vertices of the image and transform them from 3D to 2D, and Pixel shaders process all 2D pixels within 2D polygons that it receives from the vertex shader. A reference/pointer to a shader is needed in C++ to be able to start using shaders in an application. Once a reference has been made to the shader, DirectX11 has to know to use that shader and the textures are sent over as well to the graphics side to be used in the shader (if it is a pixel shader). Once DirectX11 knows what shaders to use, the model can be rendered in the scene. A basic vertex shader will calculate the positions of each vertex in the scene and transform them into 2D coordinates to be passed onto the pixel shader, UV coordinates will also be passed straight through to the pixel shader. A basic Pixel shader will go through every pixel in the scene and calculate the lighting that affects that pixel and outputs the final colour of that pixel to the GPU.

**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
  + a
* Two pass Gaussian Blur
  + a

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
  + a
* Fisheye Post Process
  + Asd
* Distort Post Process

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.
* Improvement of Code to add flexibility.
  + The
  + I have made improvements to the code in two ways, the first was through adding additional classes, for the lights and textures that were being used.
    - For the Lights, I added a new CLight class that uses the Model class as a base, where all the user needs to add when implementing the new light is the Mesh, strength, Colour, Position and Scale of the new light. I have also added a SetLightStates Function to the lights where you are able to set the Blend state, Depth state and rasterizer state to be used by the graphics card for the light.
    - For the textures, I added a new CTexture class, that contains a Resource pointer and a shaderResourceView pointer for the texture, you are then able to use the LoadTextureFromHelper function in the class to add a texture to the texture object.
  + The second 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.
  + The third and final improvement I made was adding a Light structure in the Common.hlsli and Common.h to be used for the lights in the PerFrameConstants struct whenever a new light was needed to be added to the scene. This Light structure contains the information needed about the light to perform the needed calculations, such as the lights direction, colour and position.

**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 of made was improving the polygon effects around the wall openings to

# References

Shehata, O., 2015. *A Beginner's Guide to Coding Graphics Shaders.* [Online]   
Available at: https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313