CG Midterm Question

Student Numbers- 6+6+6=18 (even questions)

Repository link: https://github.com/zenyahima/CG-midterm-project-2022a
Base framework taken from intermediate CG tutorial 3 by Sage.

Roles and Responsibilities:

Angelina: Worked on the coding, providing components for functionality, and the report
Connor: Worked on the coding, setting up the scene and lighting
Ikponmwosa: Worked on the art portion, and the report

Game Chosen- Mario vs Koopa Boss Fight

(Even questions)

1.Which game did you choose? Please provide a brief description clearly outlining

We wanted to create a boss fight game where Mario and Koopa fight against each other. Mario is the main character that the player will use and Koopa is the enemy we will have to fight. We wanted to make the mechanics as simple as possible. In order to cause damage on the enemy, the player will have to jump on the enemy and in order for the enemy to cause damage to you, the enemy has to follow you and touch you. If the the player successfully takes all of the enemy's health, the player will win. If the enemy takes all the players' health, the enemy wins.

Select the playable character and use it to explain the graphics pipeline stages associated with Vertex shader, Geometry Shader and fragment shader

Playable Character- Mario

- In the first stage of creating the character Mario, Mario goes through vertex specification. In this stage, all the data that has been placed in code for Mario will be organized and setup for rendering.

- In the second stage the Mario mesh goes through the vertex shader. The vertices for Mario will be processed and handled. Also the transformations of the vertices will happen during this Stage and the 3D form for Mario will start to come to shape.

-In the Geometry Shader, the vertices on Mario are taken and manipulated. It processes the primitives of his mesh/uv. All information is then sent to the fragment shader.

-In the fragment shader, every pixel is assigned a color and then outputted to the frame buffer for rendering.

Explain how the Phong lighting model allows you to create a metallic feel for objects within the game.

The phong lighting model is a combination of an ambient, diffuse, and specular to create a phong reflection. The equation is roughly ambient + diffuse + specular = phong reflection. The diffuse component determines how rough an object looks, the specular component determines how shiny it looks, and the ambient component determines how scattered the reflection is on the surface of the object. With these combinations, we can easily recreate the properties of metal. Metal has a hard uniform surface, but is very shiny across that surface. To create a metallic feel you would simply have a low diffuse component, because metal is smooth, high specular, and a uniform ambient component.

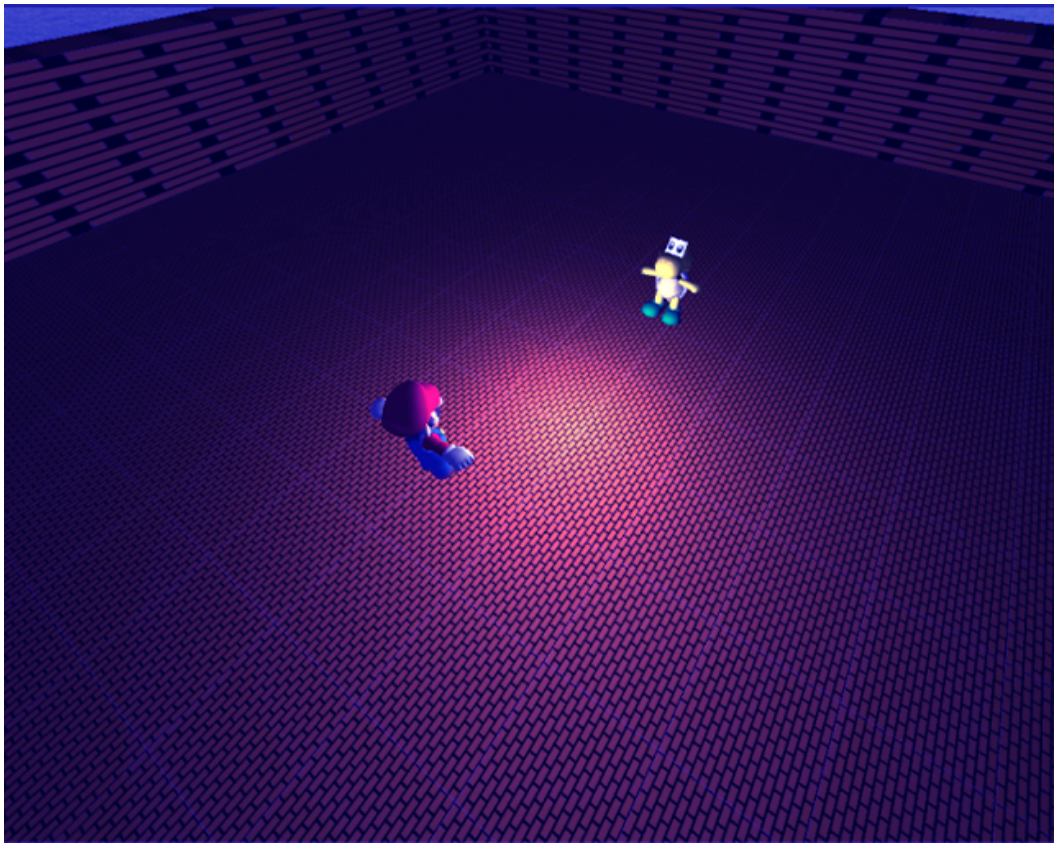
Explain what approach allows you to create a winter feel Using shaders

There are lots of ways to achieve a winter effect. One way is to use color correction to apply a “cool” effect to the scene. Using LUTs, we can color correct our scene. We create a 3D texture by taking a screenshot of our game, going into photoshop and applying our desired color correction, and exporting that as a 3D texture. Then in our game engine, we apply this texture as a LUT to our scene. This can be easily toggled on/off. Then, in the render stage of the pipeline, we retrieve this LUT and bind it to the buffer, applying it on top of everything else.

Using shaders, a winter feel can be created using a fragment shader. The fragment shader determines the final colour of the pixels/input. Normally we determine the color by

getting the material's diffuse/albedo map, so to apply a winter/cooling effect, we can get that color and then correct it using the LUT explained above. The calculation would be something like

"Frag_color = texture(ColorCorrection, inputColor).rgb"
We could also just directly set this ColorCorrection variable ourselves with something like ColorCorrection = vec3(0.0, 0.0, 1.0)
We emphasize only the blue part as blue is what is associated with winter.



A dynamic light that gives the effect of changing the scene (e.g., day passing, seasonal changes, etc.)

This includes proper light behavior when moving away or closer to objects.

The dynamic light that we want implemented into the game is one that allows for a day/night cycle. To do this we need to first create some basic lights and attach them to our scene. We need to make sure that these are sufficient and bright enough for our day portion. We do this by adjusting the color and range (intensity) of these lights.
Then, during the program runtime, we would need a timer to keep track of how much time is

passing within the game. Once we deem that enough time has passed to consider it to be nighttime, we can simply take some of the lights in our scene (but not all of them, otherwise it would be pitch black), and reduce their range/intensity to 0. After another period of time, we will say that it's day again, and once again we will take those same lights we turned off before and set their range/intensity back to their original values.
We can make this effect more fancy as well by making the lights that we're changing move across the screen as we turn them on/off. This could be achieved by simply modifying their positions over time.
We already have some basic light functionality in the game, that changes how objects look depending on how close/far they are, but these lights are not dynamic.

Explain how you implemented the shader for this Midterm and indicate why this choice was made.

Make sure to indicate why this shader was chosen and how it enhances the chosen game

For this game we were thinking about a pixelation shader. We wanted to give this game an old retro look to bring back the feel from the older Mario games. Using this shader will make the game look unique from others and it puts the player into a whole different era and this is why this is what makes this shader so diverse.

The pixelation shader would be implemented as a fragment shader. It would divide the screen into tiny rectangles as determined by a value that we set (the smaller this value is, the greater the pixelation effect, because this would increase the size of the rectangles which would mean a greater division). The shader would then sample the color from the lower left corner of each rectangle and fill the whole rectangle with that color. So we are essentially coloring every fragment using only a portion of the intended texture. The result is the pixelation effect. (Figure 1)

Figure 1

Links (Textures/Models)

https://www.pngfind.com/mpng/ibmhhmh_bricks-wall-8-bit-super-mario-brick-hd/

https://3dwarehouse.sketchup.com/model/495f8aeaf88258369f7d71070c69ca1f/Mario

https://3dwarehouse.sketchup.com/model/a7ec9133556b5eb39f7d71070c69ca1f/Koopa-Verde