Midterm Intermediate Computer Graphics

Evrett Upshall - 100784071 - Explanation Concepts and Implement
Ryan Fieldhouse - 100784589 - Explanation Concepts and Implement
Jordan Peylo - 100790170 - Implementation (code)

Description

The game we chose was frogger, the playable character is the frog that starts at the bottom of the screen. There is only one playable character that the player can control. The obstacles in the game are cars that move right to left or left to right. The cars are treated more as obstacles than enemies since the player has no other interaction with the cars besides the lose condition. The conflict to be resolved is helping the frog cross the road and get to the other side. The win condition is when the playable character reaches the other side of the road. The lose condition is when the playable character gets hit by one of the obstacles.

1 + 9 + 0 = 10 even not prime
Result: Even

Explanation of concepts

- The playable character is the frog and the frog has to move through the graphics pipeline in order to be created and have a model. The vertex shader is a programmable part of the graphics pipeline that applies transformations to the object (translation, rotation, scale, etc), the vertex shader can also be used to perform a per-vertex lighting and also be used to set up effects for shaders in the later stages of the pipeline. The geometry shader is a fixed part of the graphics pipeline that is optional meaning this part can skipped over. The geometry shader is used for taking one primitive and then creating and rendering many images of the same primitive so for example if the primitive is a triangle the geometry shader will create multiple triangles with the same dimensions and shaders. The fragment shader is a fixed part of the graphics pipeline and writes to the framebuffers about colours for primitives and shapes. The fragment shader uses rasterization to produce the colour for the framebuffers.

- With the phong lighting model you have a few different types of lighting techniques. The ones we will use to create the metallic texture are the ambient, diffuse and specular techniques. First we want to go through with a first pass of ambient lighting. We want this lighting to be very low so that it doest illuminate the model but does give an outline. We can then add a diffuse lighting technique so that part of the model is illuminated and some is not, this effect gives the

impression of a directed light. Finally we want to create a specular highlight, this will give the effect that the material on the object is shiny as most metals are. It would also be beneficial to make the base color of the object similar to a metal color that most people would be used to, maybe a gray or blue-gray of sorts.

- To give a winter feeling, the most obvious step would be to make the shaders look like snow. Just some sort of grainy white sheet over the original textures would do well. Just make sure to keep some of the original textures relatively untouched as snow wouldn't typically reside there. Secondly, you could make the lighting that affects the shaders dark, as if it were nighttime. When you think of summer you generally think of bright sunny days so doing the opposite would imply that it is the opposite season, in this case winter. There is also the option to use a bump map on your snow textures, this could make it seem like less of a white sheet and more of a snowy texture and give the scene some depth. You could also give a small blue tint to the shaders. When there is a lot of snow it can actually end up reflecting some of the light from the sky back at you which ends up giving it a blue tint. Finally, you can use particle effects. Simply put, the particle effect could be a blizzard around you. Each one of these particles would be white, grey or tinted blue. They could also have a few specular highlights on maybe 1 in 50 particles.

Explanation of how to implement

- A dynamic light that can gives the effect of changing the scene by time passing would need to move and change the strength of the light over time. The light would be treated as the sun and would move as the sun does in real life.
    - The first part would be having the light move from left to right to simulate the rise of the sun in the east and then set in the west. Moving the light would be done in the main loop so that the light can interpolate while time advances. The light would move based on the program seconds so for every second the light would progressively move more to the left side of the screen. The amount of time between the rise and set would be determined by the programmer but once the light is set on the left it would then be turned off and reset position on the right side of the screen. The light would be turned back on when the programmer sets the seconds that would need to pass in order for the light to turn back on. This cycle can repeat forever as it uses the program seconds counter.
    - The second part would be having the strength of the light slowly diminish as the light moves closer to the left side of the screen and eventually sets. Reducing the strength of the light can be done by changing the intensity of

the light in the main loop. The intensity could be directly tied to the position of the light so if the light is directly in the middle it would be the most intense and as the light rises or sets it would be at the lowest and then while the light is being reset to simulate night it would have set awake as false or have an intensity of 0. Using the intensity of the light to show the passage of time would be done faster compared to the changing position of the light since the light will always be moving the intensity would change once the light hits a position. If the light is moving from east to middle it would be getting brighter and if the light is moving from middle to west it would be getting dimmer.

- Shader
    - We implemented the shader by applying the shader when the objects are made, using this format would allow us to use different shaders if wanted but all the objects use the same shader. For the shader, we wanted to implement, we chose to go with the toon shader. Since we are recreating frogger we wanted to go with a very retro and cheery vibe. Since toon shaders are associated with cartoons and fun, we thought this would be a great addition to our game. The toon shader makes the objects brighter and smoother to the eye resembling that of a cartoon. This shader was picked over other options because it felt more tied to arcade games and gave a closer frogger feel compared to the others.

Notes

- Unfortunately our week 6 code was broken and everything would move together for some reason that we couldn't figure out, there was no parenting or attachments or anything. We had to resort to using the week 3 framework which gave us effectively no control over lights other than through the GUI. We knew how to implement the lights we just simply couldn’t because the framework baked the lights to the scene rather than making them game objects like week 6 did. This made it so that we couldn’t implement any of the lighting mechanics (dynamic and togglable lighting) our plan was to do a dynamic light by creating a day-night cycle using an ambient light. We would have had it change as a function of delta time and once it gets below a certain threshold we would have begun to make it get brighter until it was day time in which case we would have decreased it again. Togglable lights would have been simple boolean values, if on, display the lighting layer, if off, don't. This code would have been placed in the Application.cpp file inside of the game object loop. We also planned to attach

spotlights to the obstacles and the player to bring focus to them and enhance the game experience.