Graphics Midterm Report

Kai Joseph - 100783670
Stephen Chin - 100786592
Jaden Hepburn - 100791169

Game: Description

- Our game is space invaders. We'll have 1 player that can move left and right and shoot projectiles at 3 enemies at the far end of the plane. When all 3 enemies die from the projectiles the player wins. If the character gets hit by any of the enemy's projectiles then the player loses.

Common Questions

Team members:

Kai Joseph: Programmer - Game's basic functionality
Stephen Chin: Artist - Models and texturing
Jaden Hepburn: Programmer - Lighting

Space invaders

- Basic Functionality
  - Camera behind the back - Kai
  - 2.5D
  - Projectiles - Jaden
    - Enemies Shoot Back - Lose condition when character dies.
    - Win by killing all the enemies on screen
  - Character movement - Kai
  - Enemy Movement - Kai
  - Models - Stephen
- Texturing
  - Create Model textures - Stephen
  - Plan to get in game image editing as backup
- Shaders
  - Holes in texture effect (keyword) - Jaden
  - Toon shader - for cartoonish effect - Kai
- Lighting
  - Phong Lighting - kai

https://discord.gg/A5q3YYRrQu

What is the result of adding the last digit of each team member's student number?
0 + 2 + 9 = 11 → Prime

Explanation of Concepts

Explain the limitations of not using the programmable stages of the graphics pipeline to enhance your game.

One of the many limitations of not using the programmable stages of the graphics pipeline to enhance the game is that the models, or primitives within the game would appear dull as the users are not able to directly interact with how the pipeline calculates and processes data from the points array. As such, The models in the game would appear to be more polygonal, rather than a refined shape.

Another limitation of not using the optional stages of the graphics pipeline is that the game will demand more processing power, as clipping and face culling would be able to reduce the amount of data the pipeline would have to process. And as such, by not using the programmable stages of the pipeline the user would have a slower gaming experience as the game struggles to render areas of the model where they would never be seen.

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

To create a glass effect through phong lighting by adjusting the diffuse, specular, ambient and shininess of the lighting equation. Bumping up shininess,diffusion's reflectance, specular lighting. Will create a glass feeling effect. To get the glass feel in an object. Combining the ambient, diffusion, specular and shininess into a struct in the fragment shader to make a material then referencing and applying the values how ever you see fit.

Explain what approach allows you to create a horror feel using shaders
A way that you could create a horror feel using shaders in a game would be by using a shader to limit the distance that a player could see around them. For example the player can only see 1-2 meters in front of them. This will make the player scared to progress

since they will not be able to see what is far ahead of them. This could most likely be done through a fragment shader utilizing attenuation.

How to Implement Concepts

Making scene elements (e.g., trees, floor, etc.) emit light upon interactions (e.g., a collision). This includes proper light behavior when moving away or closer to objects.

To make scene elements light up within our game, we are planning to make the character emit light when they are struck by the enemy's projectiles.

To implement this we use one of the individual lights in the game under application.cpp then update its position to move toward the player's position whenever it gets hit by a projectile which will be handled by a bool within charactermover.cpp. This will give us the position of the player that we can use to update the light's position every time a player hits a projectile a light will appear within that spot.

To create this effect, we need both the enemy models and the player projectiles to have a collider, and create an inactive lightsource in each enemy model to make them glow. Next, we will need to add a component that is tied to each enemy entity that would constantly check if a player projectile is colliding with the hitbox of the enemy. Finally, if the component senses that a player projectile is colliding with the hitbox of the enemy model. The inactive light source would be "lit up" and start to emit light from within the enemy model. As the interior of the model does not contain any normals, the light from the now activated light source would pass through the model and into the camera, as such the model would appear to be glowing.

Explain how you implemented the shader for this Midterm and indicate why this choice was made.
We decided to implement a toon shader or cel shader mainly because toon shaders give a cartoon/cel shading effect to objects in a scene which can make them look more cartoonish which would fit well with our casual arcade game Space Invaders.

To implement toon shading we would give materials that use the toon shader a uniform called steps or cellSteps. This uniform would then be passed to the fragment shader that can be used as a parameter for how many cells to include in the calculation. After we're in a fragment shader we would implement blinn-phong lighting however at the end before passing the out color we would take the result of the lighting after it has it's light accumulation, it's color from the VS shader and its texture color we would take that

result then round it using this calculation: result = round(result * cellSteps) / cellSteps this will create a cell shading effect on objects that use this shader.

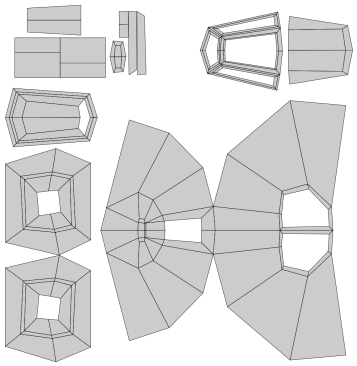
Implementation

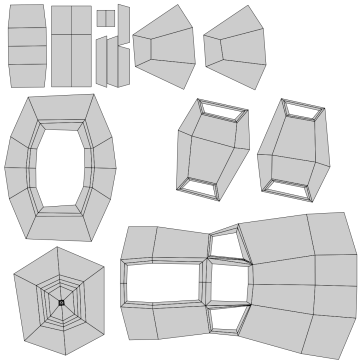
Basic functionality in OpenGL

- Playable character basic movement, obstacle or enemy movement, win and lose conditions.
- Each light term (i.e., Diffuse, Ambient, Specular) must be TOGGLED using either the keys below, or GUI toggles:
    - ‘1’ = No Lighting
    - ‘2’ = ambient lighting only
    - ‘3’ = specular lighting only
    - ‘4’ = Ambient + specular
    - ‘5’ = Toggle Textures on / off

Textures

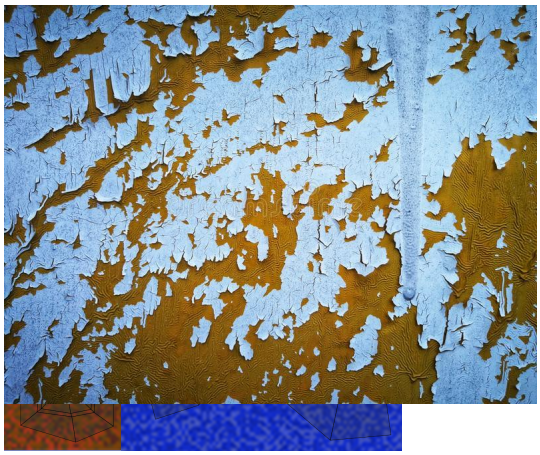
- Your scene must utilize texturing effectively.
- UV-mapping in shaders, texture sampling is required.

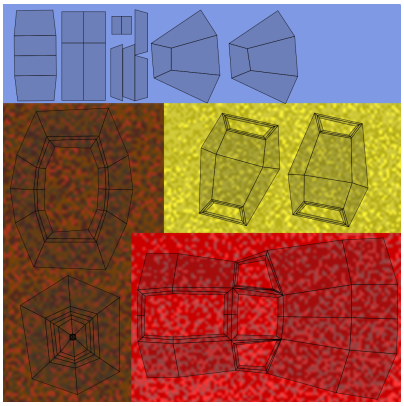




- Your texturing can not be straightforward boring stretched low-resolution textures.

- Please provide references for your textures

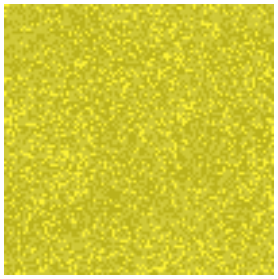




A majority of the textures are pixel art interpretations of paint flakes on a wall.

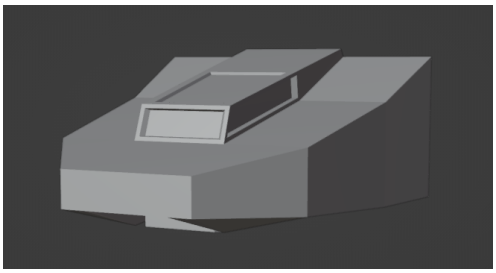
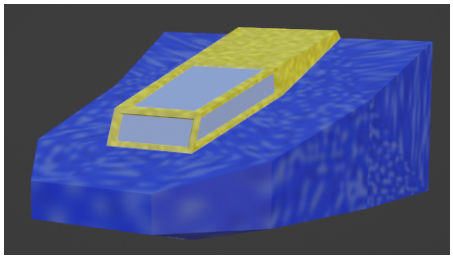
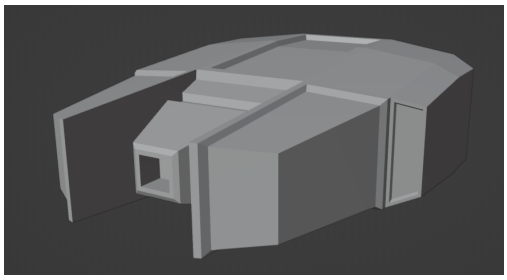
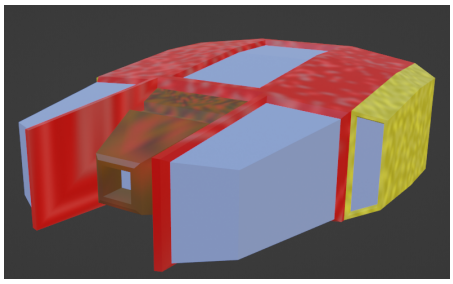
As for the windows and thrusters, they have the same plain texture as to contrast with the paint flake texture.

For reference



Although done to a limited extent, the color difference between the pixels are meant to be the color of the paint slowly disappearing overtime, and show the previous, “rotten” layer of paint that was covered by the newest layer of paint.

- Texturing must be able to be TOGGLED on/off

(Note, this is done in blender)

Basic Lighting

- Making scene elements (e.g., trees, floor, etc.) emit light upon interactions (e.g., a collision). This includes proper light behavior when moving away or closer to objects.

Shader

- Implement a shader of your choice. Make sure to indicate why this shader was chosen and how it enhances the chosen game.