CG Midterm

9+2+0=11; Prime!

Common Questions For All Groups:

- Who are the team members? Indicate roles and responsibilities.

Robbie Strauch (100784219)
I also did things (Programming and Report)
Gabe Mercier - (100753262)
I did things (Art Assets/UV Maps/Report)
Dante Arruda - (100709110)
I did things (Programming/Report)

- Which game did you choose? Please provide a brief description clearly outlining playable character(s), enemies or obstacles, conflict to be resolved, and the win/lose condition.

Game: Pac-Man
Playable Characters: Pac-Man
Enemies: 1 Ghost
Obstacles: Walls
Win Condition: Eat Ghost With Powerup Pellet
Lose Condition: Die to Ghost

Basic Functionality: (10%)

- 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: (10%)

- 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
- Texturing must be able to be TOGGLED on/off

Basic Lighting: (10%)

- 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.

Shaders: (10%)

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

Blinn_Phong_Textured is a nice basic shader to use as a base. It provides the Light Contribution for all light sources, Attenuation and takes in all of our Uniform Values. AS we're trying to make a sort of "Horror Pacman" we thought a simple way to convey this would be the proximity to the ghost, reddening it until it touches you. Giving a sort of Dread feeling for the player.

This is our Horror Shader

```
//The Blinn_Phong_Textured shader is the basis for our "Horror Shader"
#version 430

#include "../fragments/fs_common_inputs.glsl"

// We output a single color to the color buffer
layout(location = 0) out vec4 frag_color;

////////////////////////////////////////////////////////////////
/////////////// Instance Level Uniforms ///////////////////////
////////////////////////////////////////////////////////////////

// Represents a collection of attributes that would define a material
// For instance, you can think of this like material settings in
// Unity
struct Material {
      sampler2D Diffuse;
      float     Shininess;
      Float     RednessMultiplier;
};
// Create a uniform for the material
uniform Material u_Material;

////////////////////////////////////////////////////////////////
///////////// Application Level Uniforms ///////////////////////
////////////////////////////////////////////////////////////////

#include "../fragments/multiple_point_lights.glsl"

////////////////////////////////////////////////////////////////
/////////////// Frame Level Uniforms ///////////////////////////
```

```glsl
/////////////////////////////////////////////////////////////////

#include "../fragments/frame_uniforms.glsl"
#include "../fragments/color_correction.glsl"

// https://learnopengl.com/Advanced-Lighting/Advanced-Lighting
void main() {
      // Normalize our input normal
      vec3 normal = normalize(inNormal);

      // Use the lighting calculation that we included from our partial
file
      vec3 lightAccumulation = CalcAllLightContribution(inWorldPos, normal,
u_CamPos.xyz, u_Material.Shininess);

      // Get the albedo from the diffuse / albedo map
      vec4 textureColor = texture(u_Material.Diffuse, inUV);

      vec4 textureColor.x = textureColor.x * RednessMultiplier;

      // combine for the final result
      vec3 result = lightAccumulation  * inColor * textureColor.rgb;

      frag_color = vec4(ColorCorrect(result), textureColor.a);
```

Explanation of Concepts: (25%)

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

The limitations of not using the programmable stages of the graphics pipeline is for one that you would not be able to edit different shader programs to give different effects for your game for example if you wanted to give a metallic effect material to an object you would still need the vertex and fragment shaders.

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

The Phong lighting model uses the sum of three components(ambient, diffuse, specular) as the light reflected off of the object. These three components vary based on the reflectance properties of its surface. By focusing on the specular lighting component of

The Phong lighting model, we can adjust the reflective properties of the object, making it mirror the properties of glass as similarly as possible. By calculating the dot product of the view direction and the reflected direction, then raising it to the power of 32, we can determine the shininess value of the highlight.

- Explain what approach allows you to create a horror feel using shaders.

The approach used for the horror feel with shaders is done by having the orbs that the Otomatome eats emit a small amount of illumination. This illumination will allow for shading on the Otomatome and the ghost, giving them a small amount of light to show the player where the location of the enemy is in the scene.

Explanation of Implementation: (25%)

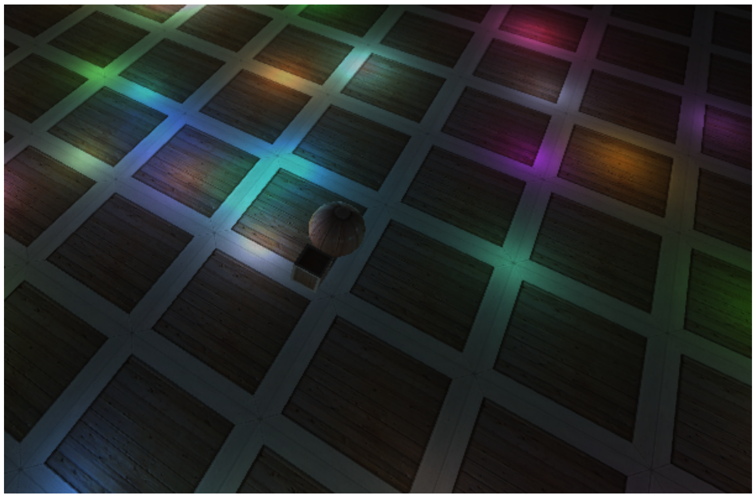
- Making scene elements (e.g., trees, floor, etc.) emit light upon interactions (e.g., a collision). This includes proper light behavior when moving away or closer to objects.

PseudoCode:

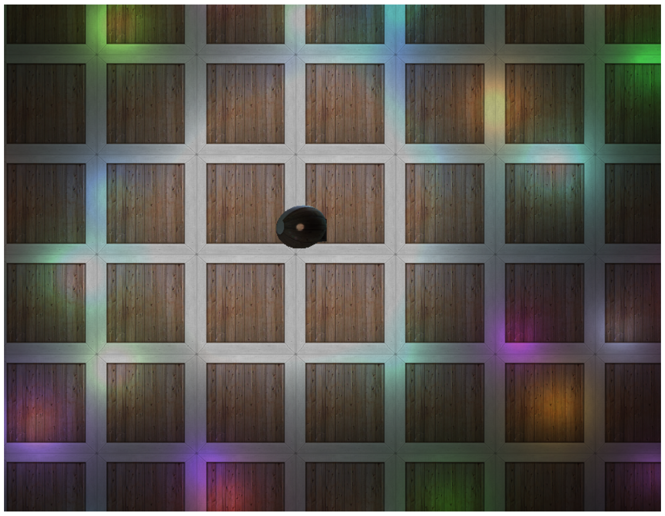
```
//Check For Collisions/Interactions
if (collisionDetected){
      Light->IsEnabled = true;
     //Bind Blinn_Phong_Textured Material to the player.
}
```

We check for collisions with pellets, they emit a light when they're touched. the light illuminates the corridors around you to help you see your surroundings and navigate your way through the Pac Maze. This gives the player a paranoid suffocating feeling because they never know when/where the ghost will strike.

Without Lighting



With Lighting



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

```
//check range from target
     //Change RednessMultiplier based on range
if(PlayerInRange) {
//Material Swap to the "Horror Shader"
     //Bind G-Buffers
}
```

The shader we chose is a simple shader based on the Blinn_Phong_Textured.glsl Shader provided in the tutorials which is modifies to change redness based on some parameter (In this case proximity to our Ghost/Main enemy) it does this by multiplying the TextureColor.a field for our Texture. For Non-Textured Shaders we will simply change the fragcolor.a instead.

Artist Tools & Logic:
Using Blender, as well as photoshop, I created the models for the Otomatome, and the ghost, as well as the UV map for the models, and attached them to the project.
Blender was used for the models, and Photoshop was used to create the pallet for the colors on the models. The logic behind using Blender was the ease of being able to export the models easily as OBJ files, as well as set up the UV maps for each object and export them efficiently.

Acknowledgements:
Codebase comes from Sage Matthews TA for Intermediate Computer Graphics.