Advanced Graphics– BRANDON STAFFERTON

Contents

[Literature Review 3](#_Toc121584092)

[Camera 3](#_Toc121584093)

[Controls 3](#_Toc121584094)

[Light 3](#_Toc121584095)

[Light Movement 3](#_Toc121584096)

[Directional Light 3](#_Toc121584097)

[Point Light 3](#_Toc121584098)

[Spot Light 3](#_Toc121584099)

[Normal Mapping 3](#_Toc121584100)

[Tangent Space Lighting 3](#_Toc121584101)

[Normal Map 3](#_Toc121584102)

[Parallax Mapping 3](#_Toc121584103)

[Simple Parallax 3](#_Toc121584104)

[Steep Parallax 3](#_Toc121584105)

[Relief Parallax 3](#_Toc121584106)

[Parallax Occlusion 3](#_Toc121584107)

[Parallax Self-Shadowing 3](#_Toc121584108)

[Special Effects Pipeline 3](#_Toc121584109)

[Render To Texture 3](#_Toc121584110)

[Screen Space Tint Effect 3](#_Toc121584111)

[Advanced Techniques 3](#_Toc121584112)

[Gaussian Blur 3](#_Toc121584113)

[Bloom 3](#_Toc121584114)

[Deferred Rendering 3](#_Toc121584115)

[Shadow Mapping 4](#_Toc121584116)

[Bibliography 6](#_Toc121584117)

# Literature Review

## Camera

### Controls

## Light

### Light Movement

### Directional Light

### Point Light

### Spot Light

## Normal Mapping

### Tangent Space Lighting

### Normal Map

## Parallax Mapping

### Simple Parallax

### Steep Parallax

### Relief Parallax

### Parallax Occlusion

### Parallax Self-Shadowing

## Special Effects Pipeline

### Render To Texture

### Screen Space Tint Effect

## Advanced Techniques

### Gaussian Blur

### Bloom

### Deferred Rendering

Deferred rendering is a screen space technique to remove forward rendering (learnopengl, LearnOpenGl, 2020). Using deferred rendering splits, the lighting calculations, and materials properties into two separate shaders. The material properties shader creates a GBuffer which contains the normals, diffuse, specular, position ambient and emissive textures and pixels. The lighting shader then uses the GBuffer, and lighting calculations combined to generate a rendering pass and results in the lighting only computing for the pixels that are visible and gives a better scalability to the number of light sources.

I have implemented this technique by creating six render target views, shader resources, textures and then a single lighting target view, shader resource and texture to pass through a screen quad. Now that six textures have been created, we can clear the render target views before doing the first render pass of the GBuffer. Once the render target views of the six textures have been cleared, we set the render target to the GBuffer textures, the first render pass then renders all the game objects by using the material shader which only does material calculations for normal mapping and height mapping. I then do a second render pass by setting the render target to the lighting view. This second render creates a blend state which combines the material shader and the lighting. Following the blend state, I then set the lighting vertex and pixel shader and set the shader resource to the GBuffer shader resource view and draw this to the quad which creates six textures as mentioned above is shown in figure X. Lastly, I do a final pass that passes the lighting resource view to the quad by setting the render target to the scene and then setting the vertex and pixel shader for the quad. This results in the use of forward rendering being removed and now our pixels have been split up into six different textures so we can determine what pixels the light hits.

Text

Description automatically generated

Diagram

Description automatically generated

### Shadow Mapping

Shadow mapping is the use of a directional light that uses an orthographic projection and the use of a matrix that uses a lights position and direction to create a depth map. By using this depth map we then can create shadows based on the scenes objects positions (learnopengl, LearnOpenGl, 2014).

I have implemented this by extending my deferred rendering by creating another stencil view for shadows which will be used to create a depth texture; therefore, I made another shader resource for shadows as well. Shadow mapping will require four different passes, before the first I cleared all the render targets and set the depth stencil to the shadow stencil. The first pass is creating the shadow/depth texture. Firstly, I created a light matrix that stores the position and direction of the light and then create an orthographic projection and then multiply these together to get a projection matrix based on the light which is then passed through the directional light properties buffer as shown in figure X. I then render all the game objects using this light matrix to create a depth texture.

Next, for the next three passes, I then did the same as my deferred rendering as previously shown by setting up the GBuffer, lighting pass and quad pass, however, within the lighting pass I change the vertex and pixel shader to use a different shader for shadows which is like the previous one but now includes shadow calculations and sets another shader resource to the shadows shader resource as shown in figure X.

The shadow calculations are done in the shader by firstly creating a light space matrix which multiplies the position of the pixels in world space and the light matrix previously made. I then sample the depth texture and use this as a current depth to compare to the closest depth to determine which depth to be drawn on the objects as shown in figure X. Finally, I multiple the shadow calculation by the final light calculations to get the results shown in figure X.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedA cube on a black and white checkered surface

Description automatically generated with medium confidence

# Bibliography

learnopengl. (2014). *LearnOpenGl*. Retrieved from LearnOpenGl - Shadow Mapping: https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping

learnopengl. (2020). *LearnOpenGl*. Retrieved from LearnOpenGl - Deferred Shading: https://learnopengl.com/Advanced-Lighting/Deferred-Shading