IMAT2908 CW: Lighting

|  |  |
| --- | --- |
| **Full Name: Robert Quick** | **P\_Number: P2658455** |
| **GitHub Username: rbtq** | **Word Count: <2200** |

[Introduction 3](#_Toc130533185)

[What are shaders? 3](#_Toc130533186)

[Brief History 3](#_Toc130533187)

[1971: Gouraud Shading 3](#_Toc130533188)

[1975: Phong Shading 3](#_Toc130533189)

[1990s: 4](#_Toc130533190)

[Real world examples – Phong Shading 5](#_Toc130533191)

[Half Life 2 5](#_Toc130533192)

[Star Citizen 5](#_Toc130533193)

[Real world examples – Gouroud Shading 6](#_Toc130533194)

[Methodology 6](#_Toc130533195)

[Introduction 6](#_Toc130533196)

[shaders.vert 6](#_Toc130533197)

[InputData 6](#_Toc130533198)

[ProcessData 7](#_Toc130533199)

[LightData 7](#_Toc130533200)

[Functions 7](#_Toc130533201)

[shaders.frag 8](#_Toc130533202)

[LightData 9](#_Toc130533203)

[MaterialDefinition 9](#_Toc130533204)

[LightDefinition 9](#_Toc130533205)

[CameraDefinition 9](#_Toc130533206)

[Functions 10](#_Toc130533207)

[TeapotAD.cpp 11](#_Toc130533208)

[Functions 11](#_Toc130533209)

[SceneDiffuse(.h and .cpp) 12](#_Toc130533210)

[Enums 12](#_Toc130533211)

[Classes 13](#_Toc130533212)

[Structures 13](#_Toc130533213)

[Methods 14](#_Toc130533214)

[Results 15](#_Toc130533215)

[Default Values 15](#_Toc130533216)

[Phong Shader 16](#_Toc130533217)

[Diffuse 16](#_Toc130533218)

[Ambient 16](#_Toc130533219)

[Specular 17](#_Toc130533220)

[Final Result 17](#_Toc130533221)

[Toon Shader 18](#_Toc130533222)

[Diffuse 18](#_Toc130533223)

[Ambient 18](#_Toc130533224)

[Specular 19](#_Toc130533225)

[Final Result 19](#_Toc130533226)

[Controls 19](#_Toc130533227)

[Results 20](#_Toc130533228)

[Introduction 20](#_Toc130533229)

[Default Values & Result 20](#_Toc130533230)

[Phong shader 21](#_Toc130533231)

[Toon Shader 23](#_Toc130533232)

[Conclusion 23](#_Toc130533233)

[References 23](#_Toc130533234)

## Introduction

A statement of the reports purpose

The purpose of this report is to show the different effects of attenuation factors, shininess, light intensity and material types on a teapot and a surface plane as well as provide a brief description of the shaders used to create the result and the implementation process in which both toon and Phong shading was implemented.

## What are shaders?

### Brief History

#### 1971: Gouraud Shading

Gouraud shading was one of the first shading techniques published in 1971 by Henri Gouraud in which ‘a surface represented by a patch is approximated by polygonal planar facets. Gouraud computes information about the curvature of the surface at each vertex of each of these facets’.(Phong, 1975)

‘Gouraud computes information about the curvature of the surface at each vertex of each of these facets. From the curvature, a shade intensity is computed and retained’. (Phong, 1975)

A shade intensity can be calculated for each vertex on a surface and then linearly interpolated between each point on a facet using the equation below.



Figure 1 (Phong,1975)

#### 1975: Phong Shading

Phong shading was an improvement to Gouraud shading published in 1975 by Bui Phong. The major building blocks of the Phong lighting model consist of 3 components: ambient, diffuse and specular lighting’ (OpenGL, unkn)

Text

Description automatically generated

Figure 2 (Phong,1975) – how to calculate specular lighting

Chart

Description automatically generated with medium confidence

Figure 3 (unknown, unkn) – How to calculate diffuse lighting



Figure 4 (unknown, unkn) - How to calculate ambient lighting

To get the overall light intensity using the Phong model, simply add the 3 values you have calculated for ambient, diffuse and specular together

#### 1990s:

Before the late 1990’s programming a shader was difficult and typically done in assembly code. This was changed when ‘PixelFlow and RealTim Shading language offered software support for real-time shaders’(unkn, unkn). However, it was not until the early 2000s that consumer grade graphics cards supported shaders.

### Real world examples – Phong Shading

#### Half Life 2



Specular Lighting

Ambient

Diffuse Lighting

Figure 5 (misterhaan, 2011) – Half life 2

#### Star Citizen



Specular Lighting

Diffuse Lighting

Ambient Lighting

Figure 6 (meatley, 2019) - Star Citizen

### Real world examples – Gouroud Shading

## Methodology

### Introduction

In this section is an explanation of all the structures and functions I have added to the program or were modified significantly, split into categories based on the file they are implemented in. At the end is a guide on the controls for the program should you wish to play around with the values yourself.

### shaders.vert

Shaders.vert calculates the model’s position on-screen using the values stores in the InputData structure below.

#### InputData

Text

Description automatically generated

This structure contains all the values that need be set in the main program for the vertex shader to work.

#### ProcessData

Text

Description automatically generated

This structure caches all the values that would be calculated more than once when the vertex shader is set up to reduce the number of operations.

#### LightData

Text

Description automatically generated

This structure contains all the data calculated from the input data in the vertex shader to be used for the fragment shader.

#### Functions

Text

Description automatically generated

What each function does:  
**calculateVM** – calculates the result of the model being translated by the camera’s view

**convertToHomogenous** – converts a vec3 to homogenous cords for matrix multiplication

**calculateVertexPosition** – calculates the vertex’s position on screen.

**calculateLightPosition** – calculates the light sources position on screen.

**calculateVertexNormal** – calculates the unit direction from the point to the light source

**setModelPositionOnScreen** – sets the model’s position on screen

Text

Description automatically generated

The main function puts all the functions together to put the objects in the scene on screen.

### shaders.frag

Shaders.frag contains code for both toon and Phong shading

When Phong shading is active, the shader calculates the total intensity of light using ambient, diffuse and specular light and the materials properties and produces a brightness value.

When toon shading is active, the shader calculates the total intensity of light using the materials ambient properties and the light direction and rounds the intensity value to the nearest available shader value.

#### LightData

Text

Description automatically generated

This structure contains all of the data in the LightData structure in phong.vert

#### MaterialDefinition

Text

Description automatically generated

This structure contains data on a material with 3 different reflectivity values and a value for shininess.

#### LightDefinition

Text

Description automatically generated

This structure contains data about the light source.

#### CameraDefinition

Text

Description automatically generated

This structure contains data about the camera.

#### Functions

Text

Description automatically generated

What each function does:

**calculateLightVector** – calculates the unit vector from the vertex to the source

**calculateAmbient** – calculates ambient light intensity based on the intensity of ambient light produced by the source and the ambient reflectiveness of the material.

**calculateDiffusePhong** – calculates diffuse light intensity based on the intensity of diffuse light produced by the source, the diffuse reflectiveness of the material and a positive angle between the material’s normal and the light source. This value is clamped between 0 and 1.

**calculateSpecularPhong** – calculates specular light intensity based on the intensity of specular light produced by the source multiplied by the specular reflectiveness of the material further multiplied by (The angle between the camera angle and light reflected off the surface of the material) to the power of the material shininess value.  
**calculateDiffuseToon** & **calculateSpecularPhone** – These functions do what their respective Phong functions do but they do not multiply by the reflectiveness of the material and the intensity of the source as the value needs to be rounded to a discrete set.

Text

Description automatically generated

The main function puts all the above functions together and simply adds the intensity of each light type together (and then rounds if toon shader is active) to produce a final value which is used when rendering.

### TeapotAD.cpp

#### Functions

Text

Description automatically generated

This function prints the change in value that just occurred after +, -, \* or / was pressed on the keypad into console.

Graphical user interface, text

Description automatically generated

This function prints the value that would currently be changed if +, -, \* or / was pressed on the keypad into console. The value selected can be changed using 8 and 2 on the keypad. The switch statement will output a string based on what value was passed in.

Text

Description automatically generated

This function handles keypresses, I modified it so it enabled me to change values for all objects in the scene and view those values in console. Useful for testing purposes.

### SceneDiffuse(.h and .cpp)

#### Enums

Text

Description automatically generated

This Enum contains everything that can be changed about the scene using the keyboard

#### Classes

Text

Description automatically generated

The above class keeps track of what value is selected to be changed. The operator functions enable the value being selected to be changed. 22 is the last value in the Enum.

#### Structures

A screenshot of a computer

Description automatically generated with medium confidence

This structure can be used to define a light source’s intensity and position and enables the modification of these values.

Text

Description automatically generated

This structure can be used to define a materials reflectiveness and shininess factor and enables the modification of these values.

The default values are set up in the scene diffuse constructor function.

#### Methods

Text

Description automatically generated

The above method displays an object in the scene, used to display the plane and teapot and reduces code re-use.

A screenshot of a computer

Description automatically generated with medium confidence

The above method changes the value currently selected using the keypad by a specified amount.

Text

Description automatically generated

The above method gets the value currently selected using the keypad.

Text

Description automatically generated

The above method prints all the values for everything in the scene into console.

### Results

#### Default Values

Text

Description automatically generated with medium confidence

Figure 7 shows the default values for both the Phong and Toon shader implemented in this program

#### Phong Shader

##### Diffuse

Scene with only diffuse lighting


Figure 8 Teapot with only diffuse lighting

Diffuse lighting was already implemented into the program, resulting in a teapot that looked like figure 1. Somewhat realistic, until you look to the horizon or at the dark side of the teapot.

##### Ambient

A picture containing text

Description automatically generated

Figure 9 Teapot with only ambient lighting

Ambient light alone resulted in a teapot that looked like figure 2. Dull and a single color.

##### Specular

A picture containing text

Description automatically generated

Figure 10 A teapot with only specular lighting

Specular light alone resulted in a teapot that looked like figure 3, very dark, but a little shiny.

#### Final Result

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

Figure 11b No shadow casted by the teapot on to the plane

Figure 11a A teapot with full Phong shading

Figure 4 shows the final result of my Phong shader, it looks realistic, apart from the fact there is no shadow casted by the teapot on to the plane.

#### Toon Shader

##### Diffuse

Graphical user interface

Description automatically generated

Figure 12 Teapot with cell shading with only diffuse lighting

##### Ambient

A picture containing graphical user interface

Description automatically generated

Figure 13 Teapot with cell shading but only ambient lighting

##### Specular

Graphical user interface

Description automatically generated

Figure 14 Teapot with cell shading with only specular lighting

#### Final Result

A teapot on a green surface

Description automatically generated with medium confidence

Figure 15 The final result of my toon shader

Long story short, my toon shader looks very similar to my Phong shader but with one exception, its not at smooth as the Phong shader and is noticeably blockier.

### Controls

I added the ability to change the values used for the shaders when the program is running. These are all **keypad** keys. Here is a list of keys and what they do:

8 & 2 – cycle through the different values that can be changed.

+ and – increase/decrease the currently selected value by a set amount.

\* & / - Increase/decrease the currently selected value by a set percentage.

Enter – Clears the console and prints all of the current values for the scene into console.

0 – Switches between Phong Shader and Toon shader.

When a key is pressed it will output the change into console

## Results

### Introduction

In this section, I will go through the results of changing the values in my scene for both the Phong shader and toon shader I have created using the functions I showcased earlier. I have chosen to take screenshots for 5 multiplier values (0.1x,0.5x,2x,10x and 1x which is the default value and there is a screenshot in the section below of what both shaders look like with the default values) as I felt this would provide a good range to showcase the effects of increasing/decreasing the values in the scene without over-doing it.

### Default Values & Result

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

Figure 16 The default toon shader scene (right)

Figure 17 The default phong shader scene (left)

### Phong shader

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Changed | 0.1x | 0.5x | 2x | 10x |
| Ambient intensity | Graphical user interface  Description automatically generated with medium confidence | A teapot on a green surface  Description automatically generated with medium confidence | Graphical user interface  Description automatically generated | Graphical user interface  Description automatically generated with medium confidence |
| Diffuse intensity | A picture containing text  Description automatically generated | Rectangle  Description automatically generated | Graphical user interface  Description automatically generated | Graphical user interface  Description automatically generated |
| Specular intensity | Graphical user interface  Description automatically generated with low confidence | Graphical user interface  Description automatically generated | Graphical user interface, shape  Description automatically generated with medium confidence | A teapot on a green surface  Description automatically generated with medium confidence |
| Shininess factor (Plane + Teapot) | A teapot on a green surface  Description automatically generated with medium confidence | Graphical user interface  Description automatically generated with medium confidence | Graphical user interface  Description automatically generated | Graphical user interface  Description automatically generated with medium confidence |
| Teapot Ambient Reflectivity Red |  |  |  |  |
| Teapot Diffuse Reflectivity  Green |  |  |  |  |
| Light Source Distance from Origin |  |  |  |  |

Reducing ambient light intensity darkened the scene, particularly as you look towards the horizon, making it look more like the teapot was in a night-time setting with a spotlight. As the ambient intensity increased, it began to look more like a daytime scene before the ambient intensity became so high that it hid all of the other lighting effects in the scene.

Reducing diffuse light intensity makes the scene feel more like it was in a twilight setting. As the diffuse light intensity increased, the scene started to look more like it was in a room being illuminated by a light source. When the diffuse intensity became very high, the scene looked as if it was directly next to a light source.

Reducing the specular intensity to a low value made the teapot look very dull, like it was made of clay. However, if the specular intensity was set to a high value, the teapot would look a lot more metallic. This didn’t really affect the appearance of the plane, likely as the part of the plane that would be effected by the change in intensity was offscreen.

Reducing the shininess factor to a low value made the teapot very reflective, like a mirror. As the shininess factor was increased, the scope of the reflection and glassiness of the teapot decreased.

As the red value for the teapot’s ambient reflectivity increased from 0.046 to 0.92, the teapot’s color went from having a blue tint, to brown, to a sort of orangey color. As the value increased beyond 0.92, it became increasingly red and bright and lost its 3D effect.

As the green value for the teapot’s diffuse reflectivity increased from 0.029 to 0.58, the teapot turned from a near solid red to brown to a sort of camo green before turning increasingly green until it reached 1 and started becoming brighter as well.

As the light source distance from the origin was moved away, it began to illuminate more of the scene and more consistently, which can be most easily noticed by looking at the plane. For small values under 2, the light source moves inside the teapot, causing this weird lighting effect.

### Toon Shader

In regards to the values changed in the table in the Phong shader section above, my toon shader produced the same results to my Phong shader above (with the only difference being the fact it wasn’t as smooth).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Changed | 0.1x | 0.5x | 2x | 10x |
| Number of shades |  |  |  |  |

As the number of shades is increased, each shade becomes less noticeable, to the point in which in the 2x image (100 shades), each shade is barely noticeable, only visible in the bottom left on the plane. In the 10x image, it is basically impossible to see an individual shade. I also noticed that smaller objects require less shades before each shade is hard to notice, just look at the 0.5x image or the default image as an example.

## Conclusion

In conclusion, I think the programming of the shaders and controls went well and I am pleased with the results of both the toon and Phong shader. I think I could improve my program by implementing more types of shaders and potentially adding shadows to my shaders as the teapot doesn’t cast a shadow on to the plane (See Fig 11a and 11b for more info).

## References

B. T. Phong, “Illumination for computer generated pictures,” *Communications of the ACM*, 01-Jun-1975. [Online]. Available: https://dl.acm.org/doi/10.1145/360825.360839. [Accessed: 24-Mar-2023].

O. G. L. OpenGL, “Basic lighting,” *LearnOpenGL*, n.d.. [Online]. Available: https://learnopengl.com/Lighting/Basic-Lighting. [Accessed: 24-Mar-2023].

C. unknown, “WorkShop 2 - Diffuse Lighting,” *Shader Workshops*, n.d.. [Online]. Available: http://www.conitec.net/shaders/shader\_work2.htm. [Accessed: 24-Mar-2023].

unkn unkn, “History of shaders - knowww.eu,” *History of Shaders*, n.d.. [Online]. Available: https://knowww.eu/nodes/59b8e93cd54a862e9d7e40e7. [Accessed: 24-Mar-2023].

meatley meatley, “Star citizen screenshots, images and pictures,” *Giant Bomb*, 03-Aug-2019. [Online]. Available: https://www.giantbomb.com/star-citizen/3030-40011/images/. [Accessed: 24-Mar-2023].