HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

---------***----------

MAJOR ASSIGNMENT REPORT

MODULE: CALCULUS 2 (EXERCISE)

CODE: MT1006

Students: **TRẦN NGUYỄN GIA PHÁT** **LÊ TRỌNG KIÊN** **NGUYỄN CHÁNH TÍN** **TRƯƠNG ANH QUÂN** **PHAN LÊ TIẾN THUẬN** **Cohort: CC04 – Group 10** **Semester: 213** **Academic year: 2021-2022** **Lecturer: PHAN THÀNH AN** **Submission date: 20th August 2022**	FOR EXAMINERS ONLY **Grade (in number):** **………………..** **Grade (in words):** **………………..** **Examiner 1** (Signature & Full name) **………………..** **Examiner 2** (Signature & Full name) **………………..**

Ho Chi Minh City, August 2022

STATEMENT OF AUTHORSHIP

Except where reference is made in the text of the report, this assignment contains no material published elsewhere or extracted in a whole or in part from an assignment which we have submitted or qualified for or been awarded another degree or diploma.

No other person's work has been used without the acknowledgments in the report.

This report has not been submitted for the evaluation of any other models or the award of any degree or diploma in other tertiary institutions.

Ho Chi Minh City, August 2022

[illegible] ZR [illegible]

Le Trong Kien Tran Nguyen Gia Phat Nguyen Chanh Tin

[illegible] [illegible]

Truong Anh Quan Phan Le Tien Thuan

ACKNOWLEDGEMENT

We would like to express our gratitude towards Mr Phan Thanh An, for helping us to coordinate in writing this report. We have explored meaningful findings during this work.

We are also very grateful to article writers on the internet for providing priceless knowledge about the field we are doing. Without their contributions, this report may not be done completely.

DISCLAIMER

For concise explanation, only the snippets that contain necessary calculations are shown. Methods/commands used in the process which are out of this topic's scope are not explained in the central part of the report. Comments are minimized for readability.

For examination, readers may download and run the project from the link provided in the *Appendices*. Readers may look up essential installation, required libraries and guidelines in the same section.

Please only run *main.cpp* file from the project and strictly follow the instructions in the report if one needs to modify some elements in order to avoid unexpected errors.

MEMBER CONTRIBUTION STATEMENT

-	Trần Nguyễn Gia Phát	**2153681**	**(CSE)**	(code writer, report writer)
-	Lê Trọng Kiên	**2153494**	**(CSE)**	(part I, part VI)
-	Nguyễn Chánh Tín	**2153043**	**(CSE)**	(part II)
-	Trương Anh Quân	**2153751**	**(CSE)**	(code reviewer, part III)
-	Phan Lê Tiến Thuận	**2153013**	**(CSE)**	(part IV, part V)

CSE: Faculty of Computer Science and Engineering

INTRODUCTION

Computer graphics, more commonly known as CG, is the process of creating images on a computer screen that resemble those seen in traditional art or photography. In the early days of computing, CG was used mainly for creating images for games and movies. However, over the years, CG has been used for a variety of other purposes such as creating 3D models and illustrations, designing websites and logos, creating scientific visualizations, and even teaching computer programming.

One of the most important aspects in CG is shading, a process that defines how light interacts with an object and defines its colors. Shading can be applied to different parts of an object by using different mathematical equations. There are several main types of shading, including: Flat Shading, Gouraud Shading, Phong Shading, and Ambient Occlusion (AQ).

TABLE OF CONTENT

In this report, the writers will give explanations of mathematical concepts from the Phong Shading (Phong illumination/reflection model) as well as provide implementation code written in C++/OpenGL. The readers will be able to understand the principles behind shading and develop skills in creating realistic images on their own.

I. OVERVIEW

1.1. Brief history

Phong shading was created by Bui Tuong Phong, a pioneer in Vietnamese computer graphics, and was published in his 1973 PhD dissertation at the University of Utah.

Due to its high computational complexity, it didn't become widely used until the middle of the 2000s.

It has since surpassed Gouraud shading (which had previously surpassed Flat shading) as the most popular shading technique in modern 3D video games.

1.2. Equation of the Phong Illumination Model

For each 3D surface point of the examining object, the Phong reflection model provides a formula to compute the illumination:

$$\boldsymbol{f(\vec{p})} = \underbrace{\boldsymbol{I_a K_a}}_{\boldsymbol{ambient}} + \underbrace{\boldsymbol{K_d} \sum_{\boldsymbol{i \in lights}} \left(\boldsymbol{\widehat{N}.\widehat{L_\imath}}\right)\boldsymbol{I_i}}_{\boldsymbol{diffuse}} + \underbrace{\boldsymbol{K_s} \sum_{\boldsymbol{i \in lights}} \boldsymbol{f_{spec}}\left(\boldsymbol{\vec{l}_i(\vec{p}), \vec{v}(\vec{p})}\right) \boldsymbol{I_i}}_{\boldsymbol{specular}}$$

The function $f: R^3 \to R$ takes the coordinates of a point P(x,y,z) and returns a real number, epresenting the light intensity at the point one would like to compute.

Note that this function is usually evaluated three times for red, blue and green light intensities (RGB colour model), and the hats indicate that vectors are normalized.

Despite being an extreme simplification of how lights behave, the Phong illumination model is incredibly effective for computation.

II. IN-DEPTH THEORY ANALYSIS

Since the function is the sum of three intensity components: ambient, diffuse and specular, we can split up and analyse the equation part by part. Although the formula is the standard way of presenting the Phong reflection model, each term should only be included if the term's dot product is non-negative.

$$\boldsymbol{I_p = I_{ambient} + I_{diffuse} + I_{specular}}$$

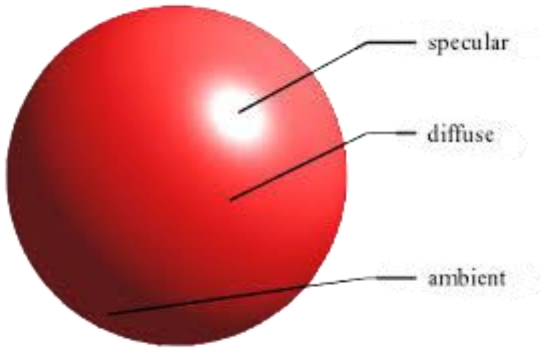


Figure 2.1: Light components of an object

Source code for this report is available at:

https://github.com/Zaphat/Phong_Shading_Demo

2.1. Ambient illumination

Ambient light refers to a general level of illumination that does not come directly from a light source. It consists of light that has been reflected and re-reflected so many times that it is no longer coming from any particular direction. Ambient light is why shadows are not absolutely black. In fact, ambient light is only a crude approximation for the reality of multiply reflected light, but it is better than ignoring multiple reflections entirely.[1]

In Phong lighting, we do not simulate indirect light bouncing, but if we did not consider this phenomenon, parts of the object not directly exposed to the source light would stay black. To fix this, the ambient component is arbitrarily assigned a fixed intensity. This intensity only depends on the material of the object (user-assigned)

$$\boldsymbol{I_{ambient} = I_a K_a}$$

Where,

$\boldsymbol{I_a}$: The ambient reflection constant, the ratio of reflection of the ambient term present in all points in the scene rendered (real number, user input)

$\boldsymbol{K_a}$: The surface ambient reflectivity, ranging from 0 to 1 (real number, user input)

[1] *Introduction to Computer Graphics*, David J. Eck (2021), section 4.1, *Introduction to Lighting.*

2.2. Diffuse reflection

Most objects we see around us do not emit their own light. Instead, they absorb daylight or artificial light and reflect a portion of it.

When a surface is illuminated with white light, it reflects coloured light caused by diffuse reflection. A surface that is a perfect diffuser scatters light equally in all directions. This indicates that the amount of reflected light observed by the viewer is independent of the viewer's position.

Diffuse reflection occurs on rough or grainy surfaces. The brightness of a point in this reflection depends on the angle formed by the light source and the surface. Lambert's Law describes the intensity of diffuse light as follows:

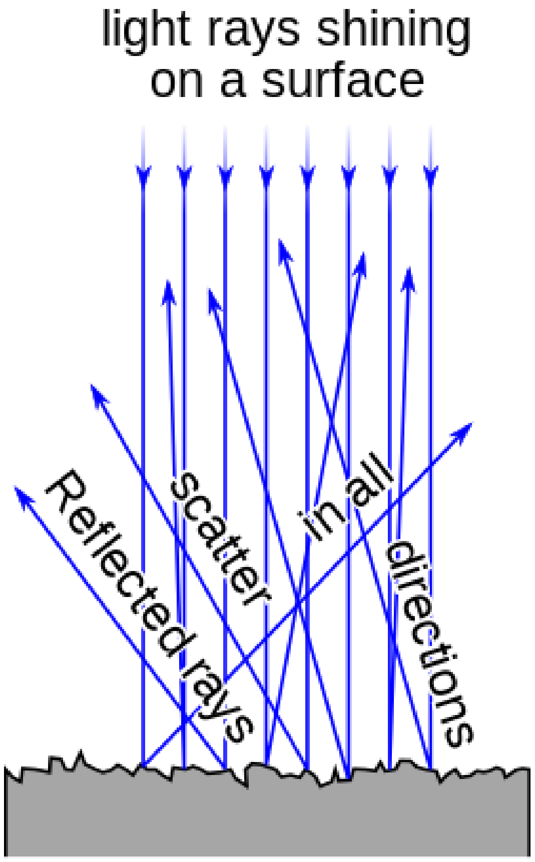


Figure 2.2: Diffuse reflection from an irregular surface (Wikipedia)

$$I_{diffuse} = I_i K_d(\hat{N}.\hat{L}) = I_i K_d \cos(\theta)$$

Where,

I_i : The intensity of the light source of the ith light (real number, user defined)

K_d : The surface diffuse reflectivity, ranging from 0 to 1 (real number, user defined)

$\hat{N}$: The surface normal (Normalized, 3D vector)

$\hat{L}$: The direction from the light source to the evaluating point (Normalized, 3D vector)

θ : The angle between light direction and surface normal.

If there is more than one light source then:

$$\boldsymbol{I_{diffuse} = K_d \sum_{i \in lights} \left(\widehat{N}.\widehat{L_i}\right) I_i}$$

Note that the dot product term must be greater than zero to be valid for computing, or else the value will be zero.

Therefore,

$$\boldsymbol{I_{diffuse} = K_d \sum_{i \in lights} max\left(0, \left(\widehat{N}.\widehat{L_i}\right)\right) I_i}$$

2.3. Specular reflection

A type of reflectivity called specular reflection is commonly described as the surface's mirror-like reflection of light. When there is a specular reflection, the incident light only reflects in one alternative manner.

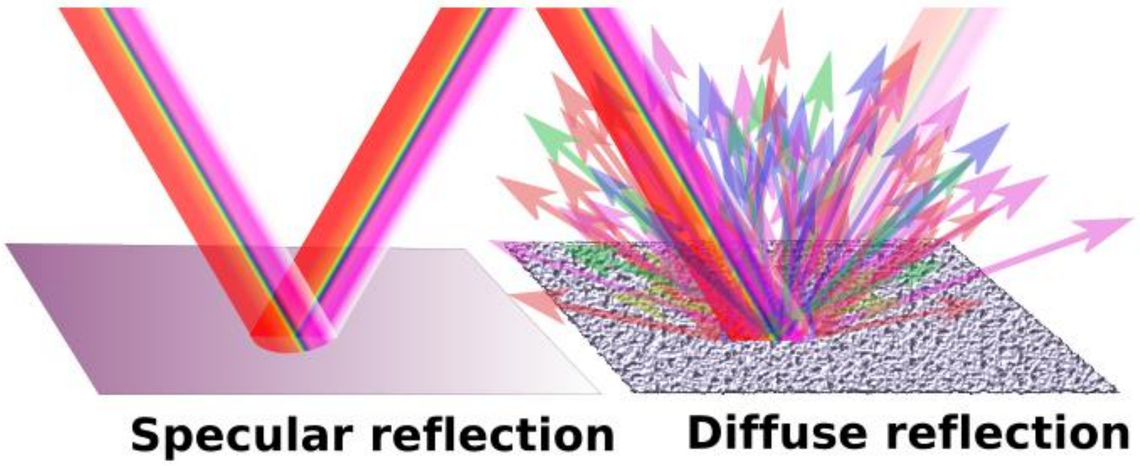


Figure 2.3: Difference between specular and diffuse reflection

Phong's model provides us with the formula to calculate the specular reflected intensity:

$$\boldsymbol{I_{specular}} = \sum_{\boldsymbol{i \in lights}} \boldsymbol{f_{spec}}\left(\vec{\boldsymbol{l}}_{\boldsymbol{i}}(\vec{\boldsymbol{p}}), \vec{\boldsymbol{v}}(\vec{\boldsymbol{p}})\right) \boldsymbol{K_s I_i}$$

Where,

$\boldsymbol{K_s}$: specular intensity of the material, ranging from 0 to 1 (real number, user input)

$\boldsymbol{I_i}$: The intensity of the light source of the ith light (real number, user defined)

$\boldsymbol{f_{spec}}\left(\vec{\boldsymbol{l}}_{\boldsymbol{i}}(\vec{\boldsymbol{p}}), \vec{\boldsymbol{v}}(\vec{\boldsymbol{p}})\right)$: The specular term

There are two ways to compute the specular term: The Phong's original method and the Jim Blinn's improvement.

2.3.1. Phong's specular term

$$\begin{cases} f_{spec} = (\boldsymbol{r}.\boldsymbol{v})^{c} \; if \; \boldsymbol{r}.\boldsymbol{v} > 0 \\ f_{spec} = \boldsymbol{0} \; otherwise \end{cases}$$

Where,

$\boldsymbol{c}$: size of specular reflection coefficient (real number, user input)

$\boldsymbol{v}$: viewer direction (3D vector)

$\boldsymbol{r}$: direction of reflection of light vector $\widehat{\boldsymbol{L_i}}$ characterized by the surface normal $\widehat{\boldsymbol{N}}$ using:

$$\boldsymbol{r} = \boldsymbol{2}(\widehat{\boldsymbol{N}}.\widehat{\boldsymbol{L_i}})\widehat{\boldsymbol{N}} - \widehat{\boldsymbol{L_i}}$$

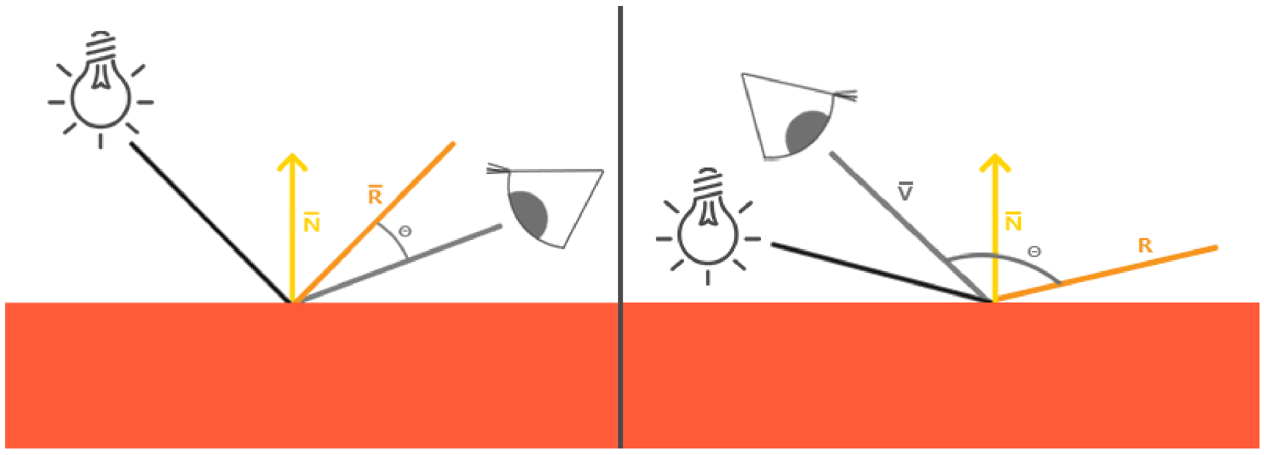


Figure 2.4: Visualization of elements in Phong's specular term (LearnOpenGL)

2.3.2. Blinn's specular term

James F. Blinn developed the Blinn-Phong shading model as an addition to the Phong shading in 1977. The Blinn-Phong model is mainly similar but approaches the specular model differently. Instead of relying on a reflection vector $\boldsymbol{r}$, we use a halfway vector that is a unit vector exactly halfway between the view and light directions. The closer this halfway vector aligns with the surface's normal vector, the higher the specular contribution.

$$\boldsymbol{f}_{\boldsymbol{spec}} = (\boldsymbol{n}.\boldsymbol{h})^{\boldsymbol{c}}$$

Where $\boldsymbol{h}$ is the bisector between $\widehat{\boldsymbol{L}_i}$ and $\boldsymbol{v}$:

$$\boldsymbol{h} = \frac{\widehat{\boldsymbol{L}_\iota} + \boldsymbol{v}}{|\widehat{\boldsymbol{L}_\iota} + \boldsymbol{v}|}$$

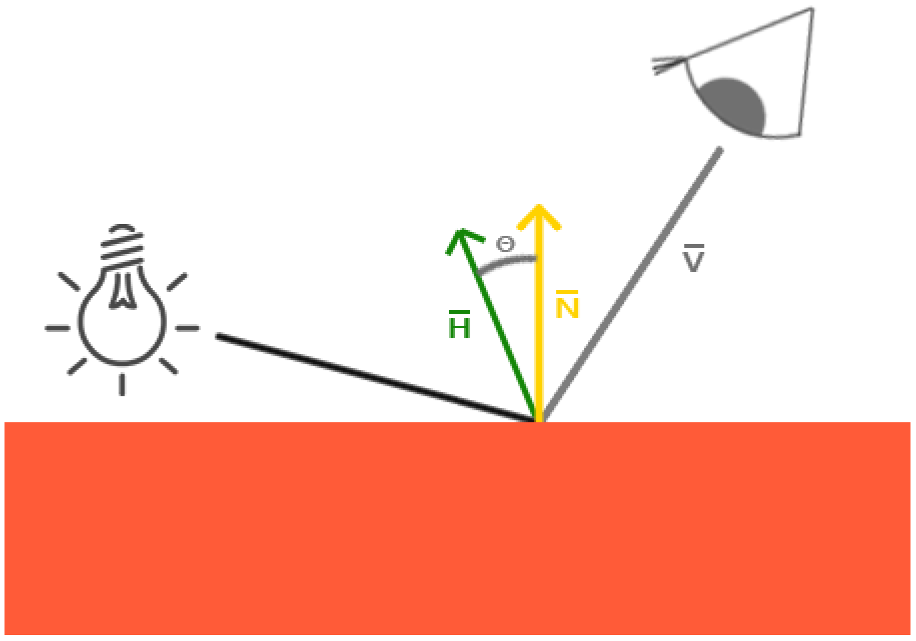


Figure 2.5: Visualization of elements in Blinn's specular term (LearnOpenGL)

Source code for this report is available at:

https://github.com/Zaphat/Phong_Shading_Demo

III. IMPLEMENTATION

Central part of the Phong shading algorithm:

There are two types of shaders in OpenGL that process different aspects of the input model: Fragment shader and Vertex shader.

"A fragment shader is the same as pixel shader.

One main difference is that a vertex shader can manipulate the attributes of vertices. which are the corner points of your polygons.

The fragment shader on the other hand takes care of how the pixels between the vertices look. They are interpolated between the defined vertices following specific rules."[2]

The Phong Shading algorithm evaluates each pixel of the object and produces the corresponding intensity. Since the modern OpenGL fragment shader has done the interpolation work for us, we do not need to rebuild the Phong interpolation for each pixel like the original formula. All we need is to implement the components in the file mentioned above.

In *assets/object.fs*:

```
struct Material {

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininess;
};
struct Light {
    vec3 position;
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
void main() {
```

[2] Stack Overflow, *Vertex shader vs Fragment Shader*, from https://stackoverflow.com/questions/4421261

```
    // ambient component
    vec3 ambient = light.ambient * material.ambient;
    // diffuse component
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(light.position - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = light.diffuse * (diff * material.diffuse);
    // specular component
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess * 
512.0);
    vec3 specular = light.specular * (spec * material.specular);
    //Combined
    FragColor = vec4(vec3(ambient + diffuse + specular), 1.0);
}
```

Furthermore, should the above calculations were done in vertex shader file (assets/object.vs), we will obtain the Gouraud Shading instead.

Switch between the Phong and Blinn-Phong specular component:

In *assets/object.fs*, find the following comments

```
//Phong specular
        //float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.shininess * 512.0);
    //Blinn specular
        //vec3 halfWay = (lightDir+viewDir)/length(lightDir+viewDir);
        //float spec = pow(max(dot(norm,halfWay),0.0),material.shininess * 512.0);
```

- The program will display light shading using the Blinn-Phong method by default.
- To see the difference between two methods, uncomment one and do the opposite with the other.

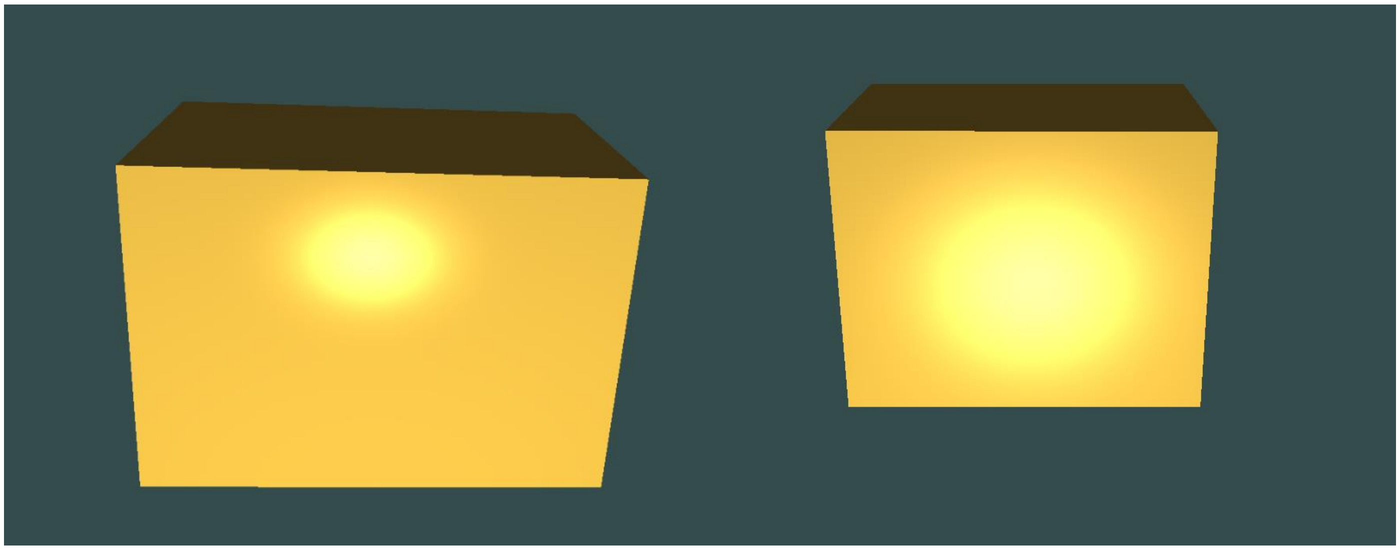


Figure 3.1: Difference between specular terms on gold surface (Phong – left, Blinn-right)

Change the material type:

In *src/main.cpp*, find *//MODELS*

```
// MODELS
    Cube cube(Material::mix(Material::emerald,Material::jade),
glm::vec3(1.0f, 2.0f, -1.0f), glm::vec3(0.75f));
    Cube cube(Material::gold, glm::vec3(3.5f, 4.5f, -1.0f),
glm::vec3(0.75f));
```

1. To change material type, simply change "xxxx" from "Material::xxx"
2. To make a material type that is the mix value of 2 types, change the x and y values from "Material::mix(Material::xxxx,Material::yyyy)"

Notes:

- There are three initial material objects in the source code: the cyan plastic cube, the gold pyramid and the obsidian sphere. Users can remove or create more at will.
- List of available materials can be found in *src/graphics/material.h*

Final result:

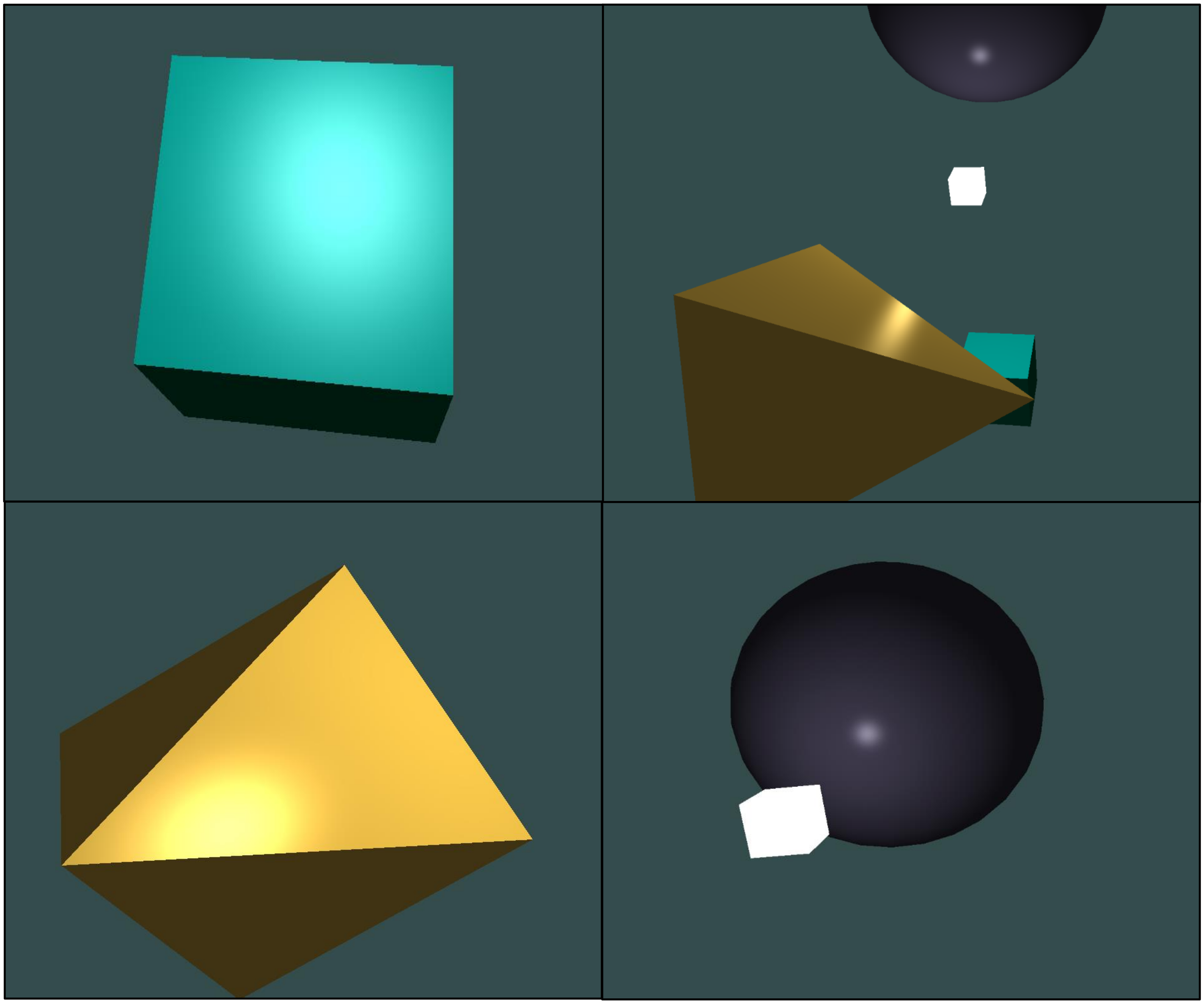


Figure 3.2: Different camera angles of the demonstration

IV. CONCLUSION

As technology improves, so does the field of computer graphics. Over the years, computer graphics has evolved from simple 2D images to complex 3D scenes and environments. Therefore, computer scientists need ways to accurately depict natural objects and surfaces as computers become more sophisticated. This allows them to build better virtual simulations and computer games closer to representing the real world.

Phong Shading is one of the most effective techniques for rendering natural life behaviours of lights and materials in computer graphics. The final results will take us to limitless possibilities combined with OpenGL, a powerful graphic library.

Due to capability and time constraints, the writers of this report have just written a simple demonstration to give readers an overlook of how the algorithm works. There are some drawbacks of the source code, such as the imperfect illumination models and single light source or lack of interactive ways to modify object elements (which can only be done by directly rewriting the code before running each time). Readers can use the source template to help implement their programs.

V. REFERENCES

Online articles

1. Eck , D. J. (2021). *Introduction to Lighting* in Introduction to Computer Graphics (version 1.3.1). Retrieved from https://math.hws.edu/graphicsbook/index.html.
2. Vaillant, R. (n.d.). *Phong illumination model*. Retrieved August 15, 2022, from http://rodolphe-vaillant.fr/entry/85/phong-illumination-model-cheat-sheet
3. *Advanced lighting*. LearnOpenGL. (n.d.). Retrieved August 15, 2022, from https://learnopengl.com/Advanced-Lighting/Advanced-Lighting
4. Guo, X. J. (n.d.). *Phong Shading and Gouraud Shading*. Retrieved August 15, 2022, from https://people.ece.cornell.edu/land/OldStudentProjects/cs490-95to96/GUO/report.html
5. J, T. (n.d.). *Phong Lighting and Specular Highlights*. Retrieved August 15, 2022, from https://www.robots.ox.ac.uk/~att/index.html
6. Luks , R. F. (n.d.). *Sphere in opengl* . Retrieved August 15, 2022, from https://romanluks.eu/blog/sphere-in-opengl/
7. *OpenGL/VRML materials*. (n.d.). Retrieved August 15, 2022, from http://devernay.free.fr/cours/opengl/materials.html
8. *Phong Shading*. Javapoint. (n.d.). Retrieved August 15, 2022, from https://www.javatpoint.com/computer-graphics-phong-shading
9. *Vertex shader vs fragment shader*. Stack Overflow. Retrieved August 15, 2022, from https://stackoverflow.com/questions/4421261/vertex-shader-vs-fragment-shader
10. Wikimedia Foundation. (2021, November 15). *Diffuse reflection*. Wikipedia. Retrieved August 15, 2022, from https://en.wikipedia.org/wiki/Diffuse_reflection
11. Wikimedia Foundation. (2022, April 9). *Phong reflection model*. Wikipedia. Retrieved August 15, 2022, from https://en.wikipedia.org/wiki/Phong_reflection_model

Books

1. Gordon, V. S., &amp; Clevenger, J. L. (2021). Computer Graphics Programming in opengl with C++. Mercury Learning &amp; Information.

Videos

1. 2etime. (2019, July 20). *Phong shading math concepts*. YouTube. Retrieved August 15, 2022, from https://www.youtube.com/watch?v=zUOvyORpgsI
2. Evans, B. (2019, November 12). *Basic shading concepts: Lambert and Phong models.* YouTube. Retrieved August 15, 2022, from https://www.youtube.com/watch?v=z0-MsGvKOhg
3. Grieco M. (n.d.). *C++ OpenGL Tutorial*. Retrieved August 15, 2022, from https://www.youtube.com/playlist?list=PLysLvOneEETPlOI_PI4mJnocqIpr2cSHS
4. Will, B. (2019, September 9). *OpenGL - lighting with the Phong reflection model (part 1 of 2)*. YouTube. Retrieved August 15, 2022, from https://www.youtube.com/watch?v=lH61rdpJ5v8
5. Will, B. (2019, September 9). *OpenGL - lighting with the Phong reflection model (part 2 of 2)*. YouTube. Retrieved August 15, 2022, from https://www.youtube.com/watch?v=KdDdljGtfeg

VI. APPENDICES

DOWNLOAD THE SOURCE CODE

1. Go to https://github.com/Zaphat/Phong_Shading_Demo
2. Select Code, click on Download ZIP and unzip the downloaded file

SETTING UP THE ENVIRONMENT

For Visual Studio 2022 users:

1. Go to the extracted folder, open *Phong Shading.sln* file and run the project. Background configurations have been done since the project was written in this version.

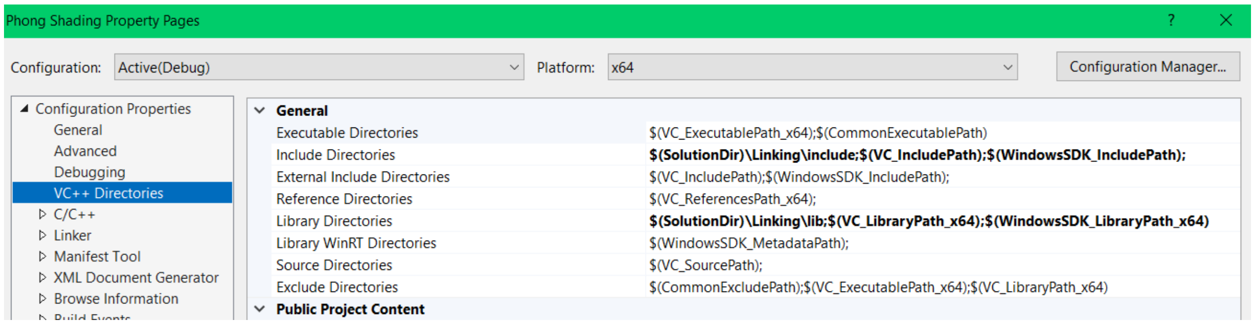
For other Visual Studio versions:

NOTES:

- Make sure everything you configure is set for ***x64*** in your IDE.
- *$(SolutionDir)* is the directory containing the *.sln* file for the Visual Studio Solution
- *$(ProjectDir)* is the directory containing the *.vcxproj* file for the Visual Studio Project

PREREQUISITES:

1. Open "Phong Shading.sln" file, in the top navigation bar, go to **View→Solution Explorer**
2. From the Solution Explorer tab, right click **Phong Shading→ Properties**
3. Make sure the configurations are the same as following bold lines:


Phong Shading Property Pages

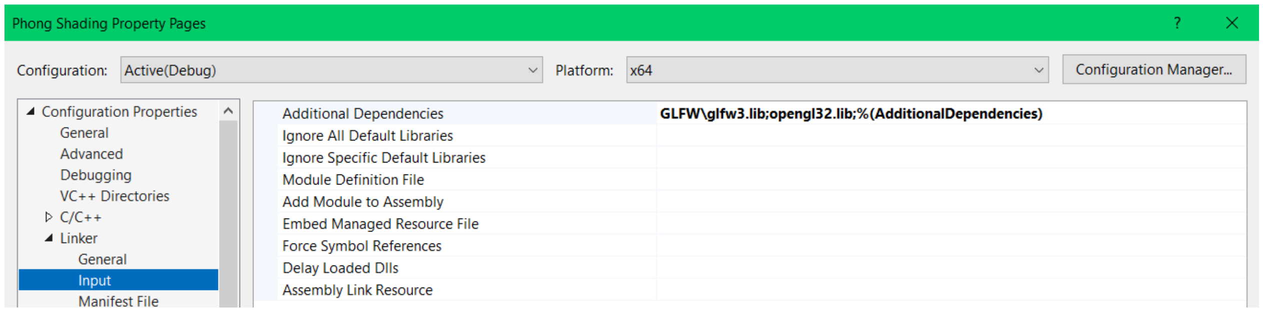
Configuration: Active(Debug) Platform: x64 Configuration Manager...

Configuration Properties
- General
- Advanced
- Debugging
- VC++ Directories
- C/C++
- Linker
- Manifest Tool
- XML Document Generator
- Browse Information
- Build Events

General

Executable Directories	$(VC_ExecutablePath_x64);$(CommonExecutablePath)
Include Directories	**$(SolutionDir)\Linking\include;$(VC_IncludePath);$(WindowsSDK_IncludePath);**
External Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x64);
Library Directories	**$(SolutionDir)\Linking\lib;$(VC_LibraryPath_x64);$(WindowsSDK_LibraryPath_x64)**
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(CommonExcludePath);$(VC_ExecutablePath_x64);$(VC_LibraryPath_x64)

Public Project Content


Phong Shading Property Pages

Configuration: Active(Debug) Platform: x64 Configuration Manager...

Configuration Properties
- General
- Advanced
- Debugging
- VC++ Directories
- C/C++
- Linker
  - General
  - Input
  - Manifest File

Additional Dependencies	**GLFW\glfw3.lib;opengl32.lib;%(AdditionalDependencies)**
Ignore All Default Libraries	
Ignore Specific Default Libraries	
Module Definition File	
Add Module to Assembly	
Embed Managed Resource File	
Force Symbol References	
Delay Loaded Dlls	
Assembly Link Resource	

GLFW

1. Download pre-compiled binaries from GLFW
2. Unzip download, copy and replace all *.lib* files from corresponding VS folder to *Linking\lib\GLFW*
3. Copy and replace *glfw3.dll* from corresponding VS folder to *$(ProjectDir)*

From this point, users can run the project normally.

CAMERA NAVIGATION

W = CAMERAS GO UP (OBJECTS GO DOWN)

S = CAMERAS GO DOWN (OBJECTS GO UP)

A = CAMERAS GO RIGHT(OBJECTS GO LEFT)

D = CAMERAS GO LEFT (OBJECTS GO RIGHT)

SPACEBAR/LEFT SHIFT = ZOOM IN/OUT

SHOW PARAMETER HINT OF A FUNCTION

For Windows and Linux users:

Ctrl + Shift + Space

For MacOS:

Shift ⇧ + Cmd ⌘ + Space

DOCUMENTATIONS FOR FURTHER READING

1. Basic understanding of linear algebra for computer graphics:
   https://youtube.com/playlist?list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab
2. OpenGL tutorials: http://www.opengl-tutorial.org/

   https://learnopengl.com/
3. OpenGL API Documentation: https://docs.gl/