322COM  
Advanced Graphics Programming  
(Deferral)  
  
Task  
1.  
Written Report

Student ID

9032499

Coventry University

Author’s Note:

GitHub repository

< <https://github.coventry.ac.uk/lattb/9032499_322COM_Task1_Defferal> >

**Table of Contents**

Table of Contents…………………………………………………………………………………1.

Table of Figures…………………………………………………………………………………..1.

Introduction…...………………………………………………………………………...………..2.

Geometric primitives……………………………………………………………………………..2.

Background customization…………………………………………………………………….…3.

Sphere shading…………………………………………………………………………………...5.

Application display……………………………………………………………………………....7.

Performance and Optimization…………………………………………………………………..8.

Application final output………………………………………………………………………….9.

Reflection…………………………………………………………………………………….…..9.

List of References……………………………………………………………………………….10.

**Table of Figures**

Figure Background Math Diagram……………………………………………………………………..3.

Figure Background Calculations……………………………………………………………………….4.

Figure Background Calculated Image………………………………………………………………….5.

Figure Diagram of Phong Shading……………………………………………………………………..6.

Figure Phong Shading Example of Primitives………………………………………………………….6.

Figure Phong Shading calculations…………………………………………………………………….7.

Figure Performance profiling…………………………………………………………………………..8.

Figure Final Application Image………………………………………………………………………..9.

**Introduction**

The report focuses on the analysis of the challenges encountered during the development of ray-tracing applications using c-plus-plus and SDL-Library. The following explores areas of demonstrating both the successful and unsuccessful implementations used for the graphics engine. The principles used are aimed to simulate an engine that produces photorealistic imagery as a result of the defined mathematical algorithms and functions. By the use of casting rays, show the intersection along the objects and lighting functionalities based on a pixel-by-ray cast intersection which defines elements to the screen. *Chicio. (n.d.).*

**Geometric primitives**

The application has a list of three primitive objects that can be displayed on the screen, using the ray-tracing intersection technique that looks at each pixel and fills it with an appropriate colour. As an example, the three primitive objects that can be displayed are the following: a triangle, a three-dimensional sphere and a plane used as the ground object. These areas were not difficult to add and were clearly stated within the modules examples as well as scratch pixel’s website used to understand the mathematics behind displaying the listed objects. For the modularity and flexibility of the application it as well introduces a virtual method for each object giving it ease of use and simplifying de-bugging of the code where appropriate.

**Background Customization**

Customizing the background in the engine gives it a significant viewpoint in displaying the capabilities of shading making the scenery closer to photorealism. Implementation of the formula was hard to understand and implement, which required the use of dismantling the concepts by trial and error to achieve the desired results. The following subject was used *Shirley, P. (2016)* in the implementation of the background formula. In the script *figure 2*, we define the starting area of the screen as “Upper\_left\_corner”, giving values of the screen dimensions as “NormalizedPixelx and NormalizedPixely”. Multiplied by the length and height, providing the dimensions of the gradient present within the scene and the distance it uses to display it, provided with an example in the *figure 1*.

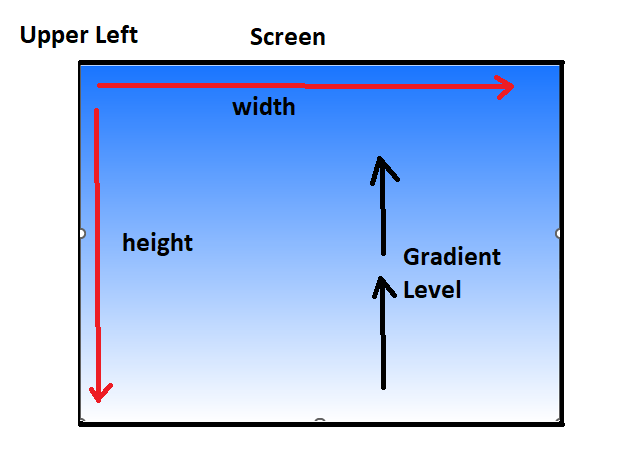


Figure 1 Background Math Diagram

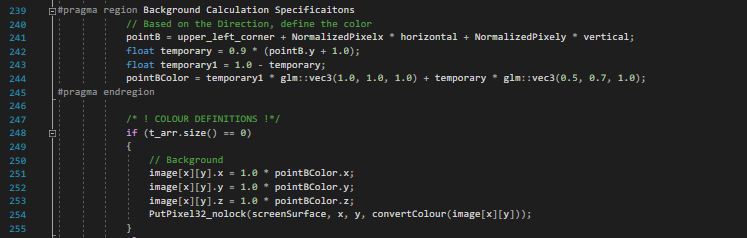


Figure 2 Background Calculations

The code can produce a multitude of areas in shading different types of colors as assigned, which are passed to the background colours in addition to the white background shown in *figure 3*. The primary functionality issue is that when the function is moved to a separate instance outside of the main class, it fails to apply the appropriate shading. Instead, it produced either a full blue or white screen. Debugging showed that functions produced the correct values, but failed to display the expected outputs. Multiplying the issue by the fact that leaving each function inside a main script can make it difficult to identify and troubleshoot future issues complicating its development.

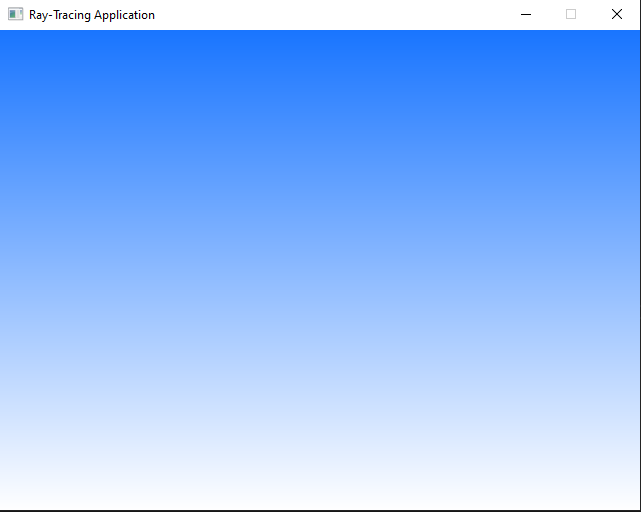


Figure 3 Background Calculated Image

**Sphere Shading**

The sphere shading calculation, issue persists with the modularity of the calculation that is set in the main function, applying shading for each element of the scene. Where the core issue is that the lighting is calculated based on a singular sphere’s position in correlation to the location of the light that is intended to be applied separately for each element such as the triangle, instances of spheres and the plane based on the view seen in *figure 5*. Shading is based on the principles of Phong-Shading, uses tracing rays to calculate the solution of lighting *figure 4*, which is demanding though can provide a realistic lighting solution to the objects it represents.

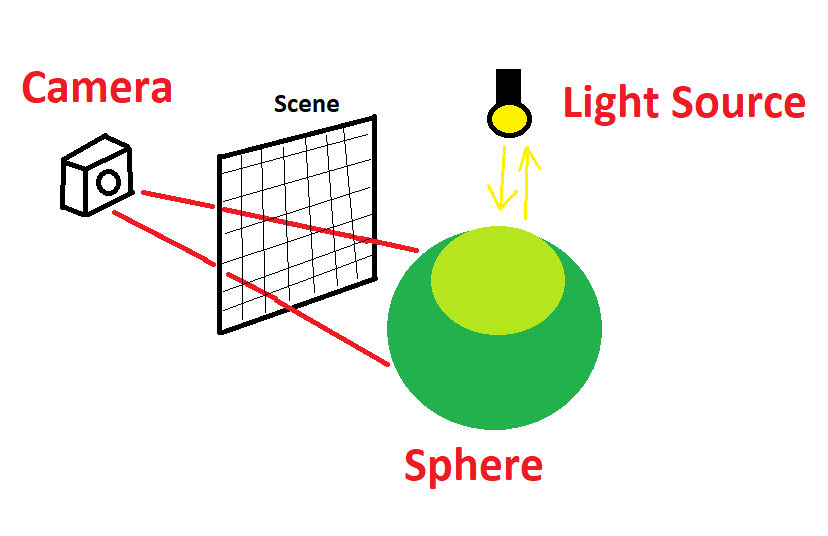


Figure 4 Diagram of Phong Shading

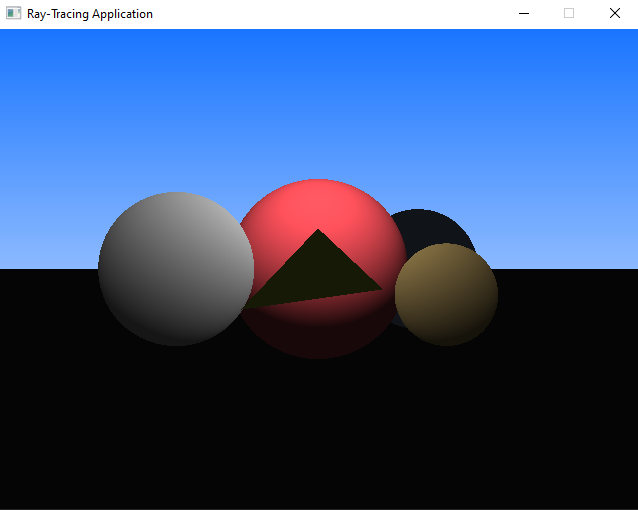


Figure 5 Phong Shading Example of Primitives

Phong-shading calculations used, in *figure 6* takes into consideration a singular sphere position to calculate the lighting of other primitives in the scene, which produces a miss-representation of other elements, defining them with no application of lighting leaving them shaded as black such as the triangle, plane and one of the spheres on the background seen in *figure 5.*

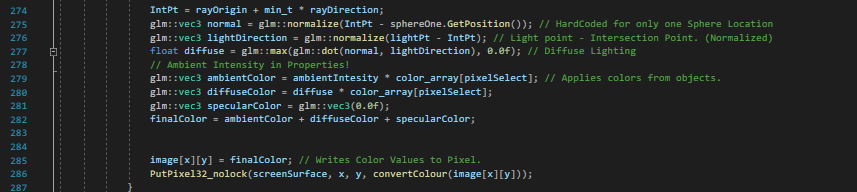


Figure 6 Phong Shading calculations

The problem of implementation was representing it modularly with its calculation as each element requires a position and distance of the light's direction. To incorporate the feature into seperated classes resulted in limitation of shading and colors of individual objects. *Codelight EU. (n.d.)., Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2013).,* Pellacini, F. (2006)., Scratchapixel. (n.d.)., *Stanford University. (2010).,* Stanford University. (2010).

**Application display**

For the display, SDL-Library is used to present the use of the window application on which the application's primitives are showcased and displayed to the user. The implementation of SDL-Library did not produce a significant area of difficulty within the development cycle as the documentation and examples provided by the module were stated professionally. After the implementation, it became clear that following the tutorials and path provided became further difficult as the rest of the examples such as shading, multi-threading and other important factors became a lot more difficult to understand and implement.

**Performance and Optimization**

Run-time of the application is not too significant seen on *figure 7*. Though the use of multi-threading could have improved the rendering time, by a factor of few milliseconds, depending on the hardware. Applications slower performance is mainly affected by the DLL file initialization, which consume significantly more time than calculations. As shading an no complex animations are produced, simple calculations won’t affect the program as significantly, nonetheless the use of modularity and splitting the areas of the task would improve it for future development.

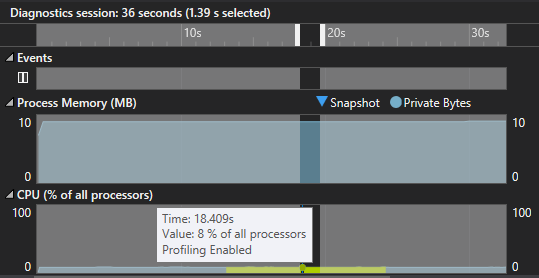


Figure 7 Performance profiling

**Application Final output**

Combining the SDL library, core primitives, phong-shading of the spheres and the customized background produces the following *figure 8.*

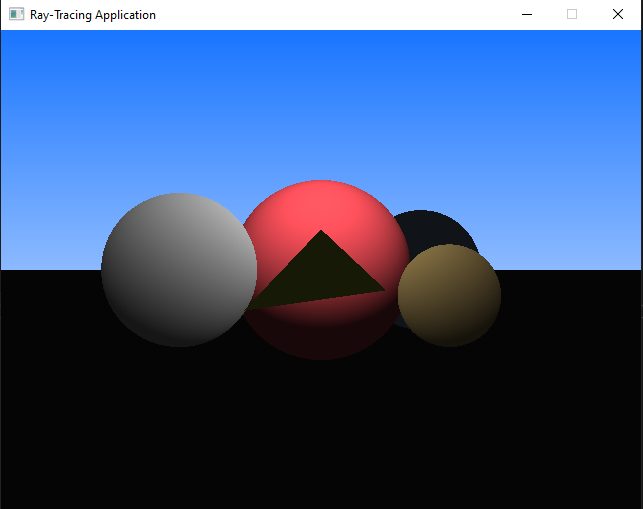


Figure 8 Final Application Image

**Reflection**

During the development, I encountered difficulties in implementing modularity in the code as well as introducing appropriate optimization. In the future, I would approach these challenges by breaking down the issues and making them more manageable prioritizing sections with provided help and direction, which aids in resolving the lack of expertise in programming.

|**List of References**

1. *Chicio. (n.d.). Ray Tracing.* <https://github.com/chicio/Ray-tracing>
2. *Codelight EU. (n.d.). Shading and Lighting. CGLearn.* <https://cglearn.codelight.eu/pub/computer-graphics/shading-and-lighting>
3. *Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (2013). Fundamentals of Computer Graphics (4th ed.)* <http://repo.darmajaya.ac.id/4242/1/Fundamentals%20of%20Computer%20Graphics%2C%20Fourth%20Edition%20%28%20PDFDrive%20%29.pdf>
4. *Pellacini, F. (2006). Lecture 3: Ray Tracing* <http://pellacini.di.uniroma1.it/teaching/graphics06/lectures/03_RayTracing_Web.pdf>
5. *Scratchapixel. (n.d.). Definition of a Ray. Scratchapixel.* <http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/definition-ray.html>
6. *Scratchapixel. (n.d.). Generating Camera Rays. Scratchapixel.* <http://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays.html>
7. *Shirley, P. (2016). Ray Tracing in One Weekend.* <http://raytracing.github.io/books/RayTracingInOneWeekend.html#addingasphere/creatingourfirstraytracedimage>
8. *Stanford University. (2010). CS148 Assignment 3: Mathematics of Shading.* <http://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/mathematics-of-shading/mathematics-of-shading.html>
9. *Stanford University. (2010). CS148 Assignment 3: Ray Tracing*. <http://graphics.stanford.edu/courses/cs148-10-summer/as3/instructions/as3.pdf>