Specular Lighting, Objects, Illumination and Shaders

Benjamin Carter and Josh Canode

Grand Canyon University

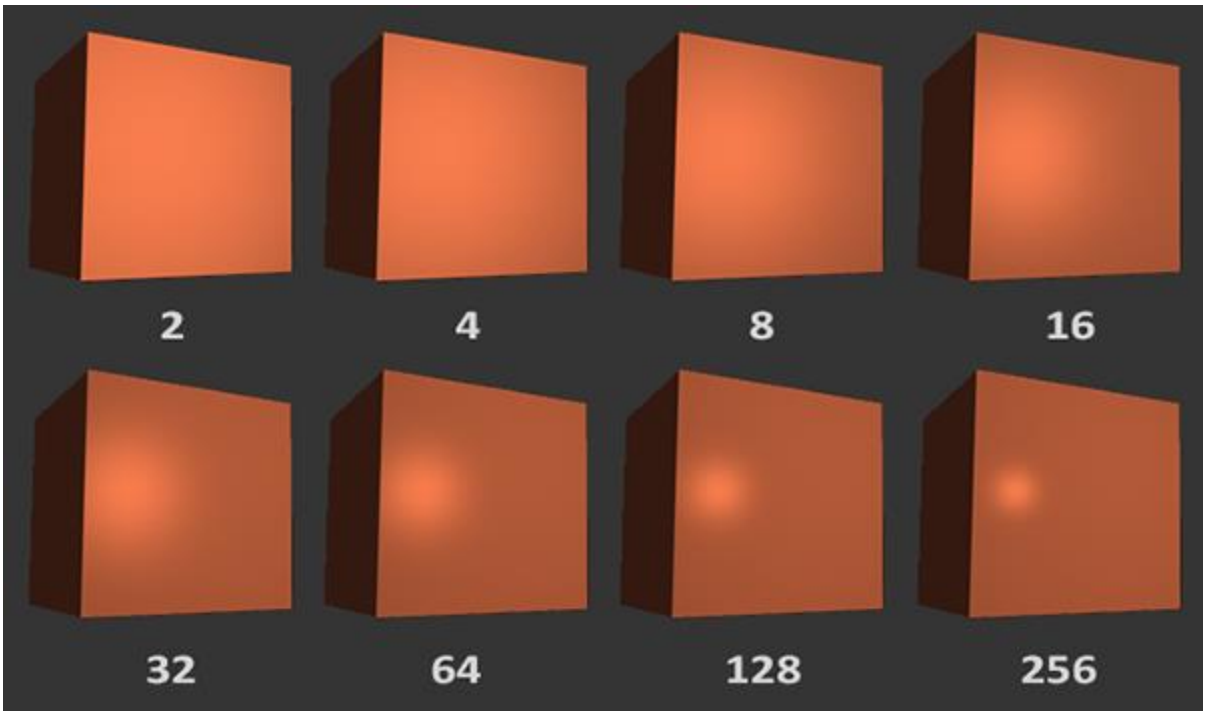
CST-305: Computer Graphics

Professor Ricardo Citro

October 28th, 2023

Project Description

The goal of this assignment is to use various lighting models in OpenGL to replicate this image:



Theoretical background

Lighting in OpenGL simplifies complex real-world illumination through ambient, diffuse, and specular components. Ambient lighting gives uniform illumination to all objects on the scene. For example, in the middle of an open field at night with no moon, there will still be a little bit of light to make out objects in the field. This is ambient lighting. Diffuse lighting illuminates objects the closer it is to the light source, determined by the angle between surface normals and light rays. Specular lighting simulates bright highlights on reflective objects, with shininess influencing their size and intensity. Normal vectors are vital for these calculations. OpenGL only approximates real-world lighting physics, to make realistic scenes computationally efficient.

Definition of All Shininess

Specular lighting determines the shininess of objects. This is a fundamental concept responsible for rendering the bright, focused, highlights on surfaces when they interact with light. In specular lighting, the light has an “exponential spread” number. The larger it is, the more focused the light. This program increases the number by powers of two and places the objects side by side for comparison. In essence, higher shininess values result in a more pronounced and refined shine, while lower shininess values create broader and less intense

highlights, representing less reflective or rougher materials. These changes can clearly be seen by our rendering.

Mathematical concepts -- Shaders

We are using Vertex and Fragment shaders. Vertex shaders are responsible for transforming the 3D coordinates of objects from their local space to clip space, applying operations like translation and rotation. Fragment shaders determine the color for each pixel after vertex processing. We use a combination of both to manage lighting and fine tune the rendering.

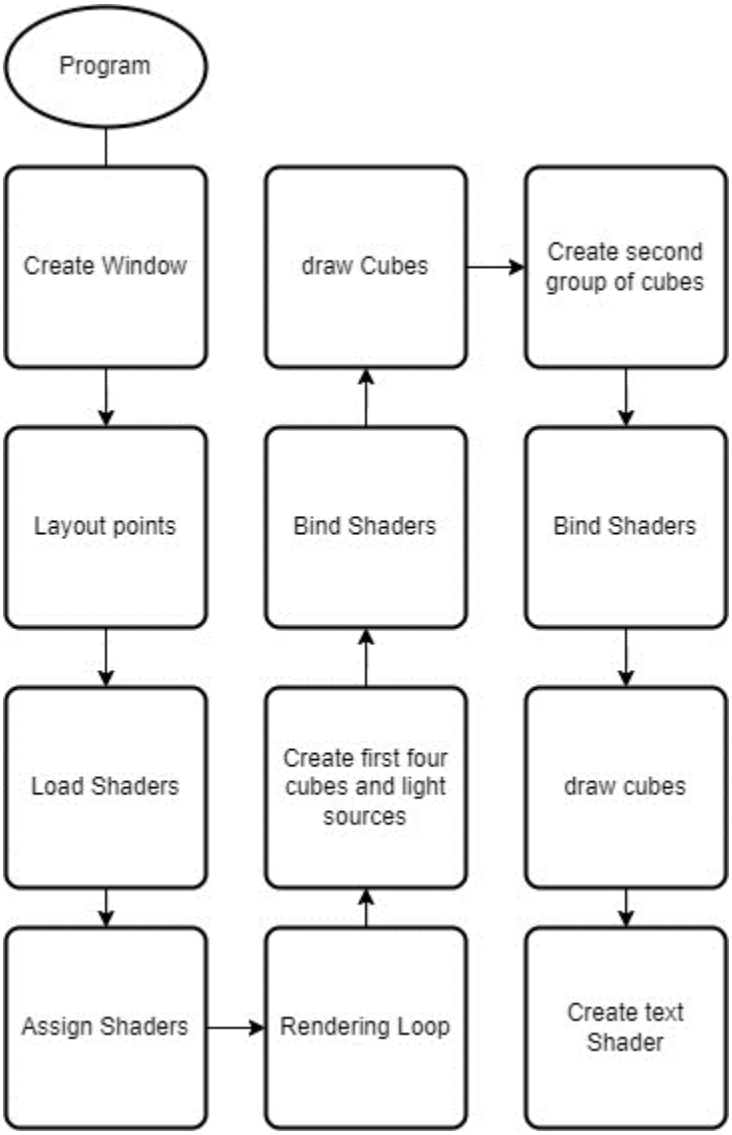
We are using the following shaders:

- 3DColor.vs/.frag
- 3DLightSource.vs/.frag
- textObject.vs/.frag

For 3D Vertex Shaders, they take in three different matrices. The Model Matrix, the View Matrix, and the Projection Matrix. The Model Matrix converts the local space vectors to world space vectors. The View Matrix represents the camera transformations. The Projection Matrix represents the camera space to flat screen space.

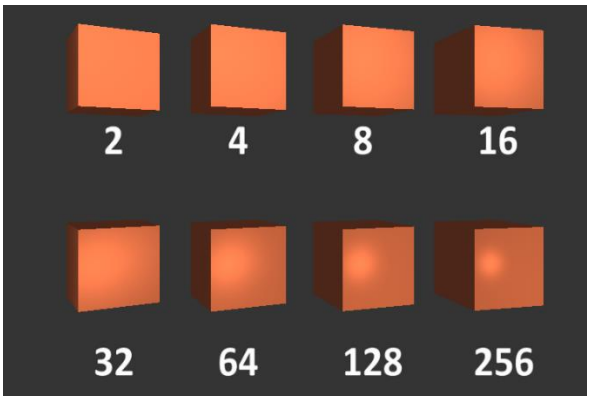
Programming Implementation

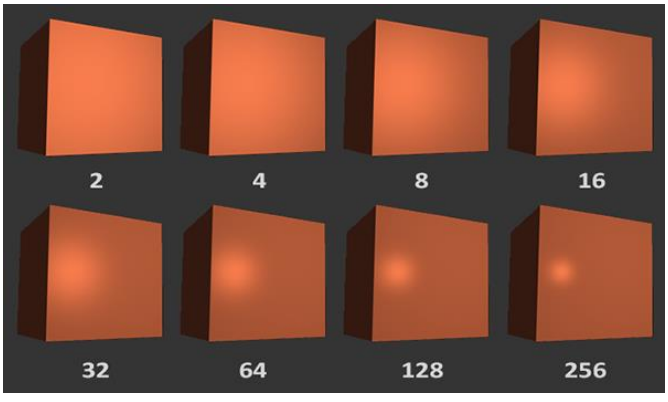
Flowchart



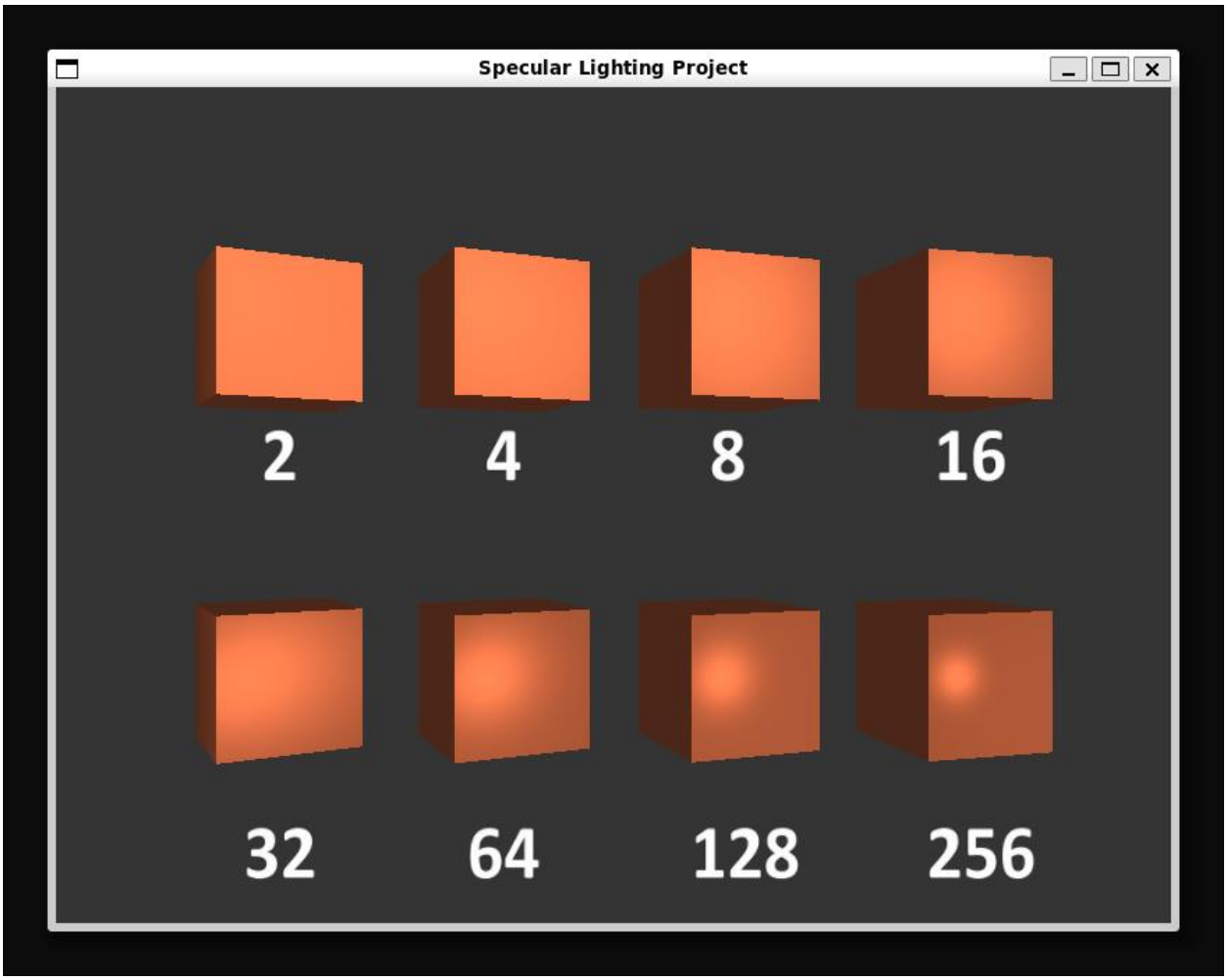
Results

Here are the results of our project:





Here you can see our rendering shown on the left vs the original picture on the right.



Testing and Validation

Throughout development, the project was continually tested and debugged. It is currently working fully as expected without any compile-time errors or logic errors.

References

de Vries, J. (n.d.). Basic Lighting. LearnOpenGL. https://learnopengl.com/Lighting/Basic-Lighting