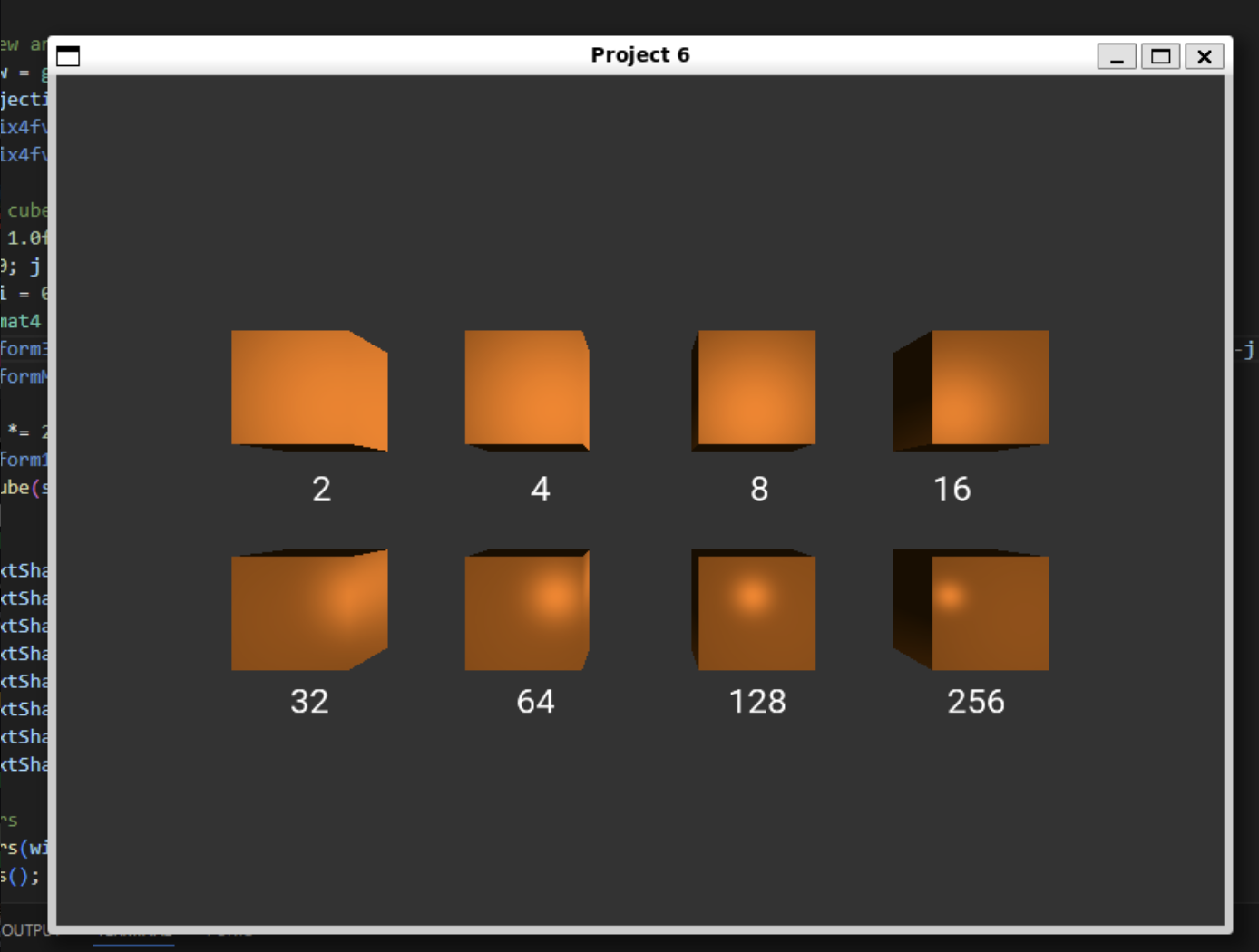
Project 6: Specular Lighting, Objects, Illumination and Shaders

CST-310 Computer Graphics

Angel Velazquez, Kyler Harden

October 27, 2024

**Documentation**

****

**Theoretical Background**

1. **Lighting and Illumination Models**:

• **Phong Reflection Model**: This is the shading model used in this application. It divides the lighting effect into three components:

• **Ambient**: A global light that affects the entire scene uniformly.

• **Diffuse**: This depends on the angle between the light source and the surface normal, giving surfaces a matte appearance.

• **Specular**: This component simulates the reflective, shiny parts of the surface, which is influenced by the shininess parameter.

2. **Shininess**:

• The shininess controls the spread of the specular highlight. A higher shininess value results in a smaller, more concentrated highlight, giving the surface a shinier appearance. Lower values produce a more spread-out highlight, simulating a rougher surface.

**Mathematical Concepts**

1. **Lighting Calculations**:

• **Ambient Light**: A constant light that is uniformly applied:

• **Diffuse Lighting**: Dependent on the angle between the light direction and the surface normal:

• **Specular Lighting**: Calculated using the reflection of the light and the view direction:

• The final color is calculated as:

2. **Matrix Transformations**:

• **Model Matrix**: Transforms object coordinates into world coordinates.

• **View Matrix**: Transforms world coordinates into camera/view space.

• **Projection Matrix**: Transforms 3D coordinates into 2D screen space for rendering.

**Programming Concepts**

1. **Shader Programming**:

• **Vertex Shader**: Transforms the 3D coordinates of vertices into the appropriate space using matrices (model, view, and projection matrices). It also calculates normals.

• **Fragment Shader**: Performs per-pixel lighting calculations using the Phong reflection model.

2. **Interactivity**:

• After displaying the initial cubes, the user is prompted to enter a new shininess value, and the program updates the shininess of the cubes accordingly.

**Aesthetic Decisions**

1. **Lighting Position**: The light source is positioned close to each cube to make the specular highlight and shininess effect more visible.

2. **Color Choices**: The object color is an orange hue (RGB: 0.5, 0.25, 0.0), and the light is white (RGB: 1.0, 1.0, 1.0) to better show the shading effects.

3. **Cube Arrangement**: The cubes are arranged in a grid-like 2x4 pattern to visually compare the shininess values.

**Algorithm Explanation**

**Main Algorithm Flowchart:**

+-------------------------------+

| Initialize OpenGL and |

| GLFW for Window Setup|

+-------------------------------+

|

v

+----------------------------+

| Setup Shader Programs |

| (Vertex and Fragment) |

+----------------------------+

|

v

+------------------------------------+

| Setup Text Rendering System |

| with FreeType for Labels |

+------------------------------------+

|

v

+--------------------------------+

| Enter Main Render Loop |

+--------------------------------+

|

v

+--------------------------------------+

| Query Shininess from the User |

+--------------------------------------+

|

v

+--------------------------------------+

| Adjust Lighting, View, Model |

| and Projection Matrices |

+--------------------------------------+

|

v

+-----------------------------------------+

| Render Cubes with New Shininess|

+-----------------------------------------+

|

v

+----------------------+

| Swap Buffers and |

| Poll for Events |

+----------------------+

**Detailed Algorithm for Rendering Cubes with Shininess:**

1. **Input**:

• Receive shininess input from the user.

2. **Cube Setup**:

• Vertices of the cube are defined with positions and normals.

• A buffer is created to store the vertex data (VAO and VBO).

3. **Matrix Setup**:

• **Model Matrix**: Translates the cube in space.

• **View Matrix**: Places the camera in space relative to the cube.

• **Projection Matrix**: Converts 3D world coordinates to 2D screen coordinates.

4. **Lighting Calculations**:

• Calculate ambient, diffuse, and specular lighting in the fragment shader.

• The shininess value directly affects the size and intensity of the specular highlights.

5. **Drawing Loop**:

• For each cube in the 2x4 grid:

• Apply the corresponding shininess value (which increases by a factor of 2 for each cube).

• Draw the cube.

• Display the corresponding shininess value on the screen using FreeType text rendering.

**Mesh and Lighting Algorithm:**

1. **Mesh Creation**:

• The cube is represented using vertex positions and normals.

• The vertices are sent to the GPU for rendering using a VAO (Vertex Array Object) and VBO (Vertex Buffer Object).

2. **Illumination**:

• In the fragment shader, calculate the light’s effect on each pixel using Phong reflection.

• Adjust the intensity of the specular highlight based on the shininess factor.

3. **Shininess**:

• The specular highlight is calculated based on the shininess value. The user’s input directly adjusts the shininess, affecting how sharp the highlights appear.

The code is structured to take shininess values as input, allowing users to control the reflectivity of the objects in real time. This interaction is crucial for understanding how shininess influences the appearance of objects in 3D rendering.