Project 10: Advanced Shaders 2

CST-310 Computer Graphics

Angel Velazquez, Kyler Harden

December 15, 2024

**1. Core Graphics Concepts**

**Coordinate Systems & Transformations**

- Uses right-handed coordinate system

- Implements model-view-projection (MVP) matrix transformations

- Camera system uses Euler angles (Yaw, Pitch, Roll) for orientation

Example from Camera.h:

```38:46:Camera.h

glm::vec3 Position; // Position vector

glm::vec3 Front; // Front dir vector

glm::vec3 Up; // Up dir vector

glm::vec3 Right; // Right dir vector

glm::vec3 WorldUp; // WorldUp vector

// Euler Angles

float Yaw; // yaw

float Pitch; // pitch

float Roll; // roll

```

**Rendering Pipeline**

1. Vertex Processing

2. Primitive Assembly

3. Rasterization

4. Fragment Processing

5. Frame Buffer Operations

**2. Mathematical Models**

**1. Matrix Transformations**

- Model Matrix: Object's position/orientation in world space

- View Matrix: Camera's position/orientation

- Projection Matrix\*\*: Perspective projection (45° FOV)

Example implementation:

```279:282:main.cpp

glm::mat4 view = glm::mat4(1.0f); // Initialize view to identity

view = camera.GetViewMatrix(); // Set view based on camera

glm::mat4 projection = glm::perspective(45.0f, (GLfloat)WIDTH / (GLfloat)HEIGHT, 0.1f, 100.0f); // Initialize projection using initial values

glm::mat4 model = glm::mat4(1.0f); // Initialize model to be 4x4 identity

```

**2. Euler Angles**

Used for camera rotation with three angles:

- Yaw (rotation around Y-axis)

- Pitch (rotation around X-axis)

- Roll (rotation around Z-axis)

Mathematical formulas used:

```cpp

front.x = cos(yaw) \* cos(pitch)

front.y = sin(pitch)

front.z = sin(yaw) \* cos(pitch)

```

Implementation in Camera.h:

```131:133:Camera.h

front.y = sin(glm::radians(Pitch));// Define y value for front using sin(pitch)

front.z = sin(glm::radians(Yaw)) \* cos(glm::radians(Pitch)); // Define z value for front using sin(Yaw) \* cos(Pitch)

Front = glm::normalize(front); // Normalize vector

```

**3. Vector Operations**

- Normalization

- Cross products (for camera orientation)

- Dot products (for lighting calculations)

Example in sphere.frag:

```20:24:sphere.frag

// diffuse

vec3 norm = normalize(Normal); // Normalizes normal

vec3 lightDir = normalize(lightPos - FragPos); // Gets lightDir

float diff = max(dot(norm, lightDir), 0.0); // Gets diff

vec3 diffuse = diff \* lightColor; // Sets diffuse

```

**3. Lighting Models**

**Phong Lighting Model**

Components:

1. Ambient Lighting: Constant base illumination

2. Diffuse Lighting: Direction-dependent reflection

3. Specular Lighting: View-dependent highlights

Implementation:

```16:24:sphere.frag

// ambient

float ambientStrengh = 0.8; // Set ambient strength

vec3 ambient = ambientStrengh \* lightColor; // Sets ambient

// diffuse

vec3 norm = normalize(Normal); // Normalizes normal

vec3 lightDir = normalize(lightPos - FragPos); // Gets lightDir

float diff = max(dot(norm, lightDir), 0.0); // Gets diff

vec3 diffuse = diff \* lightColor; // Sets diffuse

```

**4. Texture Mapping**

**UV Mapping**

- 2D texture coordinates mapped to 3D surfaces

- Uses SOIL library for image loading

- Implements texture wrapping and filtering

**5. Geometric Models**

**Mesh Generation**

- Vertices stored with:

- Position (x,y,z)

- Normal vectors

- Texture coordinates

- Triangle-based faces

- Uses indexed drawing for efficiency

Example vertex structure:

```101:109:main.cpp

GLfloat vertices[] = {

// Coordinates: 3 Position, 3 Color, 2 Texture

// Back face of cube

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, // Bottom left

0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f, 0.0f, -1.0f, // Bottom right

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 0.0f, -1.0f, // Upper right

0.5f, 0.5f, -0.5f, 1.0f, 1.0f, 0.0f, 0.0f, -1.0f, // Upper right

-0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, -1.0f, // Upper left

-0.5f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, // Bottom left

```

**6. Programming Concepts**

**Object-Oriented Design**

- Encapsulation of camera properties and behaviors

- Model and mesh classes for 3D object management

- Shader class for GPU program management

**Memory Management**

- RAII pattern for OpenGL resources

- Proper cleanup of buffers and textures

- Smart handling of texture and mesh data

Example resource cleanup:

```446:450:main.cpp

// Deallocate resources

glDeleteVertexArrays(1, &VAO); // Deallocate vertex arrays

glDeleteBuffers(1, &VBO); // Deallocate buffers

glfwTerminate(); // Terminate window

return 0; // Returns 0 for end of int main()

```

**7. Aesthetic Decisions**

**Visual Elements**

- Checkerboard pattern with alternating colors

- Skybox implementation with Yokohama textures

- Bump mapping for enhanced surface detail

- Purple and white color scheme for contrast

Example checkerboard implementation:

```305:311:main.cpp

for (int i = 0; i < 8; i++) { // For 8 rows

for (int j = 0; j < 8; j++) { // For 8 columns

if ((i+j) % 2 == 0) { // Check if i+j is odd or even for color purposes

glUniform3f(squareColorLoc, 1.0f, 0.0f, 1.0f); // If even square color is purple --> pass purple to uniform

} else {

glUniform3f(squareColorLoc, 1.0f, 1.0f, 1.0f); // If even square color is white --> pas white to uniform

}

```

This codebase demonstrates a comprehensive understanding of computer graphics principles, combining mathematical concepts with practical implementation techniques to create an interactive 3D scene.