Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа № 4 по курсу «Компьютерная графика»

Студент: Е. А. Медведев
Преподаватель: Г. С. Филипов
Группа: М8О-301Б

Дата:
Оценка:
Подпись:

Москва, 2024

Тема лабораторной работы: Освещение и работа с шейдерами

1 Цель лабораторной работы

В этой лабораторной работе вы научитесь работать с освещением в 3D-пространстве, используя различные типы источников света, и освоите основы написания шейдеров. Вы реализуете освещение объектов в сцене с использованием простейших моделей освещения и настроите эффекты при помощи вершинных и фрагментных шейдеров.

2 Требования

Вы должны использовать C++ (OpenGL+SFML). Программа должна корректно отображать освещение с учетом типов источников света, используя написанные вами шейдеры. В каждом варианте задания должны быть задействованы как минимум один тип освещения (например, направленное освещение, точечный источник света или прожектор)

3 Вариант 2: Точечный источник света и зеркальные блики

Постройте сферу в 3D-пространстве. Реализуйте точечный источник света (Point Light), используя модель Фонга для освещения. Включите зеркальные блики (specular highlights) для более реалистичного отражения света на поверхности объекта. Дополнительно: Добавьте управление положением источника света с помощью клавиатуры.

Описание работы программы

Программа создает графическое окно с использованием SFML и OpenGL, в котором пользователь может перемещаться по 3D-пространству с использованием управления камерой. Основные возможности программы:

- Перемещение камеры вперёд, назад, влево и вправо с помощью клавиш **W, S, A, D**;
- Изменение направления взгляда (углов поворота камеры) с помощью движения мыши;
- Отображение сфер в сцене, созданных с использованием OpenGL, с градиентным цветом для каждой грани;
- Управление положением источника света с помощью клавиш со стрелками.

Все изменения отображаются в реальном времени, обеспечивая плавное взаимодействие пользователя с 3D-пространством.

Код программы

```
#include <SFML/Graphics.hpp>
#include <SFML/Window.hpp>
#include <SFML/OpenGL.hpp>
#include <GL/glu.h>
#include <cmath>


float cameraX = 0.0f, cameraY = 0.0f, cameraZ = 5.0f;
float pitch = 0.0f, yaw = -90.0f;
float cameraSpeed = 0.05f, mouseSensitivity = 0.1f;

float lightPosX = 0.0f, lightPosY = 2.0f, lightPosZ = 2.0f;
float lightSpeed = 0.1f;

float toRadians(float degrees) {
    return degrees * M_PI / 180.0f;
}

void getDirection(float &dirX, float &dirY, float &dirZ) {
    dirX = cosf(toRadians(yaw)) * cosf(toRadians(pitch));
    dirY = sinf(toRadians(pitch));
    dirZ = sinf(toRadians(yaw)) * cosf(toRadians(pitch));
}
```

```
24
25      void drawSphere(float x, float y, float z, float radius, int segments) {
26          for (int i = 0; i <= segments; ++i) {
27              float lat0 = M_PI * (-0.5f + (float)(i - 1) / segments);
28              float z0 = sinf(lat0);
29              float zr0 = cosf(lat0);
30
31              float lat1 = M_PI * (-0.5f + (float)i / segments);
32              float z1 = sinf(lat1);
33              float zr1 = cosf(lat1);
34
35              glBegin(GL_QUAD_STRIP);
36              for (int j = 0; j <= segments; ++j) {
37                  float lng = 2 * M_PI * (float)(j - 1) / segments;
38                  float xSegment = cosf(lng);
39                  float ySegment = sinf(lng);
40
41                  float factor = (float)j / segments;
42                  float gray = factor * 0.8f + 0.2f;
43
44                  glColor3f(gray, gray, gray);
45
46                  glVertex3f(x + radius * xSegment * zr0, y + radius * ySegment * zr0, z +
                        radius * z0);
47                  glVertex3f(x + radius * xSegment * zr1, y + radius * ySegment * zr1, z +
                        radius * z1);
48              }
49              glEnd();
50          }
51      }
52
53      void setPerspective(float fov, float aspectRatio, float nearPlane, float farPlane)
            {
54          glMatrixMode(GL_PROJECTION);
55          glLoadIdentity();
56          gluPerspective(fov, aspectRatio, nearPlane, farPlane);
57          glMatrixMode(GL_MODELVIEW);
58      }
59
60      void setupLighting() {
61          glEnable(GL_LIGHTING);
62          glEnable(GL_LIGHT0);
63
64          GLfloat light_position[] = { lightPosX, lightPosY, lightPosZ, 1.0f };
65          glLightfv(GL_LIGHT0, GL_POSITION, light_position);
66
67          GLfloat light_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };
68          glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
69
```

```
70          GLfloat light_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
71          glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
72
73          GLfloat light_ambient[] = { 0.2f, 0.2f, 0.2f, 1.0f };
74          glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
75
76          GLfloat mat_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };
77          GLfloat mat_shininess[] = { 50.0f };
78          glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
79          glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
80
81          glEnable(GL_COLOR_MATERIAL);
82          glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
83      }
84
85      int main() {
86          sf::RenderWindow window(sf::VideoMode(800, 600), "3D Shapes with Lighting", sf
                ::Style::Close | sf::Style::Titlebar);
87          window.setFramerateLimit(60);
88          window.setMouseCursorVisible(false);
89
90          glEnable(GL_DEPTH_TEST);
91          setPerspective(45.0f, 800.0f / 600.0f, 0.1f, 100.0f);
92          setupLighting();
93
94          while (window.isOpen()) {
95              sf::Event event;
96              while (window.pollEvent(event)) {
97                  if (event.type == sf::Event::Closed)
98                      window.close();
99              }
100
101             float dirX, dirY, dirZ;
102             getDirection(dirX, dirY, dirZ);
103             if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
104                 cameraX += dirX * cameraSpeed;
105                 cameraY += dirY * cameraSpeed;
106                 cameraZ += dirZ * cameraSpeed;
107             }
108             if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
109                 cameraX -= dirX * cameraSpeed;
110                 cameraY -= dirY * cameraSpeed;
111                 cameraZ -= dirZ * cameraSpeed;
112             }
113             if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
114                 cameraX -= cameraSpeed * sinf(toRadians(yaw));
115                 cameraZ += cameraSpeed * cosf(toRadians(yaw));
116             }
117             if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
```

```
118                 cameraX += cameraSpeed * sinf(toRadians(yaw));
119                 cameraZ -= cameraSpeed * cosf(toRadians(yaw));
120             }
121 
122             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Space)) {
123                 cameraY += cameraSpeed;
124             }
125             if (sf::Keyboard::isKeyPressed(sf::Keyboard::LShift)) {
126                 cameraY -= cameraSpeed;
127             }
128 
129             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
130                 lightPosY += lightSpeed;
131             }
132             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
133                 lightPosY -= lightSpeed;
134             }
135             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
136                 lightPosX -= lightSpeed;
137             }
138             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
139                 lightPosX += lightSpeed;
140             }
141 
142             glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
143             glLoadIdentity();
144             glTranslatef(-cameraX, -cameraY, -cameraZ);
145 
146             setupLighting();
147             drawSphere(0.0f, 0.0f, 0.0f, 1.0f, 20);
148 
149             window.display();
150         }
151 
152         return 0;
153     }
```

Примеры работы программы:

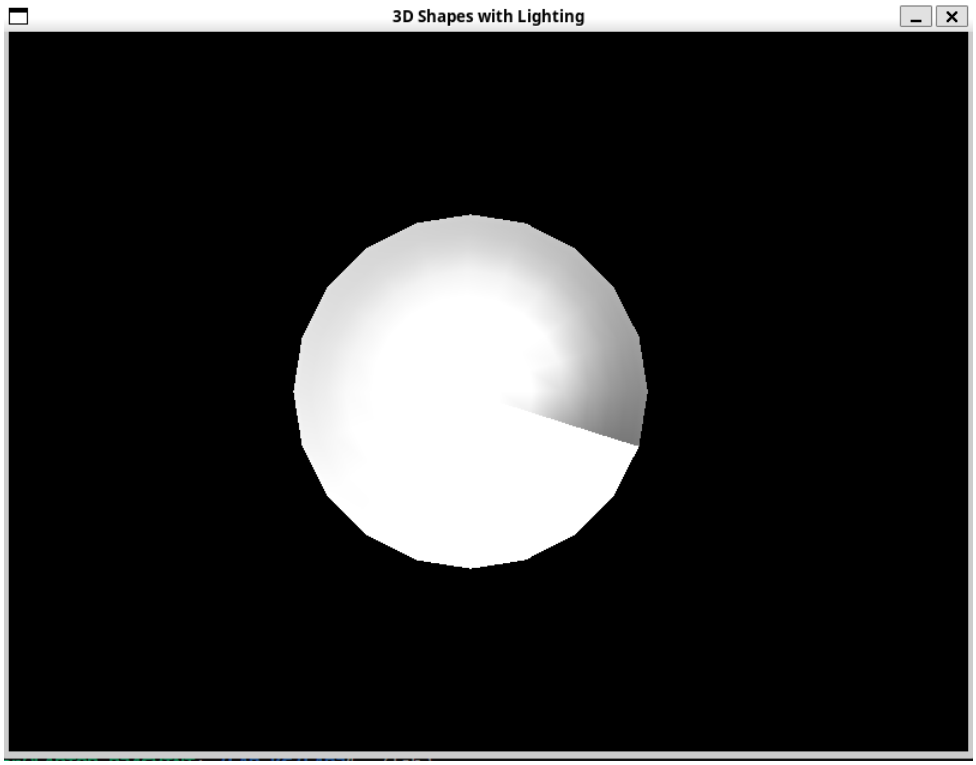

Рис. 1: Трёхмерные фигуры

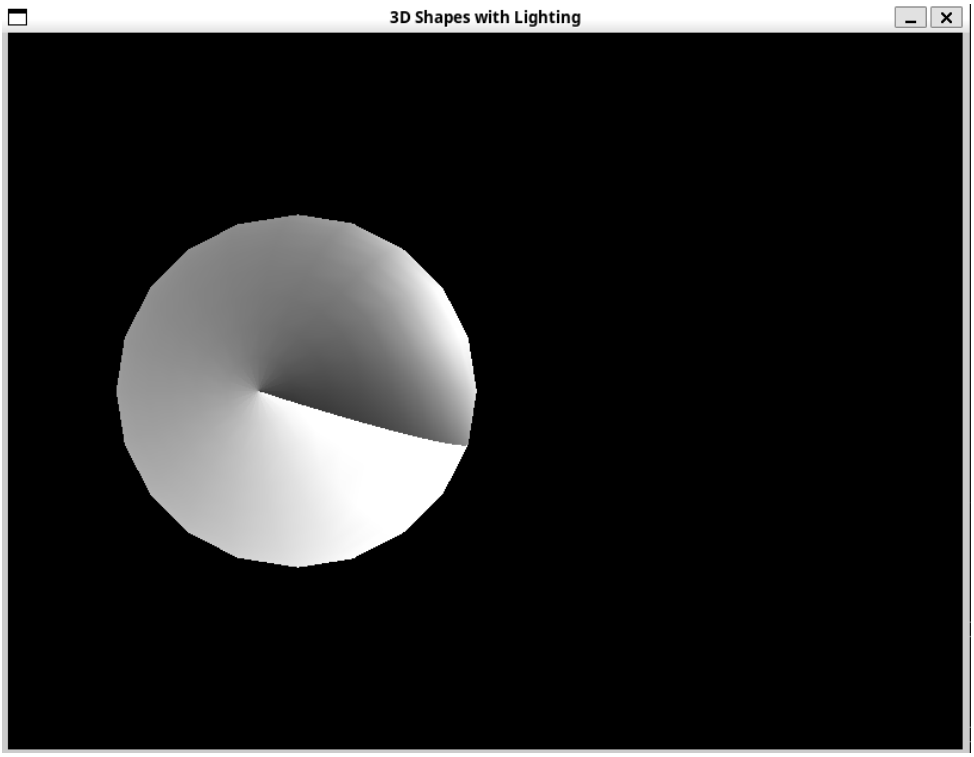

Рис. 2: Трансформация фигур

Результаты работы программы

В результате выполнения программы был создан графический интерфейс для отображения трёхмерных объектов с использованием SFML и OpenGL. Программа позволяет управлять положением камеры и отображать объекты, такие как сфера, с градиентной заливкой цвета, в реальном времени.

Основные функции программы

Программа реализует следующие возможности:

- **Управление камерой:**
  - **W** — перемещение камеры вперёд.
  - **S** — перемещение камеры назад.
  - **A** — перемещение камеры влево.
  - **D** — перемещение камеры вправо.
  - **Space** — перемещение камеры вверх.
  - **Shift** — перемещение камеры вниз.
- **Управление освещением:**
  - Стрелки влево и вправо — перемещение источника света по оси X.
  - Стрелки вверх и вниз — перемещение источника света по оси Y.
- **Отрисовка объектов:**
  - Сфера с плавным градиентом цвета.
- **Настройки камеры:**
  - Изменение перспективы и масштаба объектов в сцене.

Графический интерфейс

- Окно программы имеет размер 800×600 пикселей.
- Камера перемещается в трёхмерном пространстве, обеспечивая реалистичное восприятие объектов.
- Для отображения сцены используется OpenGL с перспективной проекцией.
- Источник света можно перемещать, изменяя освещение сцены в реальном времени.

Пример работы программы

1. Изначальное состояние камеры: положение $(0, 0, 5)$, углы вращения ($yaw = -90°, pitch = 0°$).
2. При нажатии клавиши **W**:
   - Положение камеры изменяется на $(0, 0, 4.95)$.
   - Отображение объектов обновляется с учетом нового положения камеры.
3. При нажатии клавиш **A** и **D**:
   - Камера перемещается влево и вправо, а объекты остаются на месте.
   - Отображение остаётся стабильным.
4. Управление освещением:
   - При движении источника света с помощью стрелок, освещение объектов изменяется.
5. Отрисовка объектов:
   - Сфера отображается с плавным градиентом цвета, создающим эффект освещённости.

Тестирование работы программы

- **Тестирование трансформаций:** Управление камерой и освещением работает корректно. Объекты отображаются правильно при изменении положения камеры и источника света.
- **Производительность:** Частота кадров составляет 60 FPS, что обеспечивает плавное взаимодействие с 3D-сценой.
- **Стабильность:** Исключений и ошибок в работе программы не выявлено.

Выводы

Программа успешно реализовала управление 3D-камерой и освещением, а также отрисовку сфер с градиентной заливкой цвета. Интерфейс позволяет пользователю взаимодействовать с 3D-пространством, и все функции работают стабильно и корректно.