Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 6

по курсу «Алгоритмы компьютерной графики»

Студент группы ИУ9-41Б Горбунов А. Д.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

а. Базовой лабораторной работой является лабораторная работа №3 (модельно-видовые преобразования и преобразования проецирования).

б. Определить параметры модели освещения OpenGL (свойства источника света, свойства материалов (поверхностей), характеристики глобальной модели освещения).

в. Исследовать один из методов повышения реалистичности получаемых изображений сцены (учет ослабления интенсивности света с расстоянием от источника).

г. Реализовать один из алгоритмов анимации (моделирование движения тела (с заданной начальной скоростью) при условии абсолютно упругого отражения объекта от границ некоторого ограничивающего объема (регулярной формы))

д. Реализовать наложение текстуры (загрузка из файла *.bmp или процедурная генерация) с возможностью отключения использование текстуры для определения свойств поверхности (модулирование коэффициента диффузного отражения)

2 Теория

Построение реалистических изображений

• восприятие света включает как физические, так и психологические процессы, что необходимо учитывать при построении реалистических изображений;

• примеры влияния восприятия:

– эффект полос Маха (Mach bands);

– одновременный контраст;

• при моделировании освещения используют упрощенные эмпирические модели, отражающие компромисс между реалистичностью и объёмом необходимых вычислений:

– прямолинейное распространение света (light rays);

– точечные источники света;

– простая модель освещения (модель Фонга), BRDF (bidirectional reflectance

distribution function), излучательность (radiosity);

– интерполяционное закрашивание (Гуро, Фонг).

Освещение

• световая энергия, падающая на поверхность, может быть:

– поглощена (absorption);

– отражена (reflection);

– пропущена (transmission);

• количество поглощенной, отраженной или пропущенной энергии зависит от длины волны света (соответственно, изменяется распределение энергии и объект выглядит цветным, а цвет объекта определяется поглощаемыми длинами волн);

• если объект поглощает весь падающий свет, то он невидим и называется абсолютно черным телом;

• свойства отраженного света зависят:

– от строения, направления и формы источника света;

– от ориентации и свойств поверхности (диффузное и зеркальное отражение).

Диффузное отражение (diffuse scattering): закон косинусов Ламберта

• диффузное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается;

• диффузно отраженный свет рассеивается равномерно по всем направлениям, следовательно, положение наблюдателя не имеет значения;

• свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта (Lambert): интенсивность отраженного света пропорциональна косинусу угла между направлением света и нормалью к поверхности:

• где I — интенсивность отраженного света,

• I_l — интенсивность точечного источника,

• k_d — коэффициент диффузного отражения (0 <= k_d <= 1),

• 0 — угол между направлением света и нормалью к поверхности

• цвет определяется свойствами материала;

• коэффициент диффузного отражения k_d зависит от материала и длины волны света, но в простых моделях освещения обычно считается постоянным.

Абсолютно упругие соударения

• при абсолютно упругих соударениях:

– не происходит деформации объектов при соударении;

– применимы законы сохранения импульса и энергии;

• центральное соударение шаров:

– законы сохранения импульса и массы:

$v_{1i}m_1 + v_{2i}m_2 = v_1 f m_1 + v_{2f}m_2$

$m_1 v_{1i}^2 + m_2 v_{2i}^2 = m_1 v_{1f}^2 + m_2 v_{2f}^2$

– решение:

$v_{1f} = (2m_2 v_{2i} + (m_1 - m_2)v_{1i})/(m_1 + m_2)$

$v_{2f} = (2m_2 v_{1i} + (m_2 - m_1)v_{2i})/(m_1 + m_2)$

• соударение шара с наклонной плоскостью:

$y + kx + b = 0$

$(x - xc)^2 + (y - yc)^2 = R^2$

+ 2 параметрических уравнения перемещения центра

+ $D = 0$ (касание)

• соударение шара с горизонтальной / вертикальной плоскостью:

– угол падения равен углу отражения;

– модуль скорости после соударения не изменяется.

Текстуры

• наложение текстуры выполняется с помощью функции отображения текстурного и объектного координатных пространств:

– задается текстурная функция (texture map) texture(s,t) в так называемом текстурном пространстве (texture space), которая генерирует значения цвета или яркости для каждого значения s,t e[0,1];

– производится отображение на требуемую поверхность, заданную в объектном пространстве (s,t)->(x,y,z);

– производится стандартное преобразование для вывода на экран;

• на практике может решаться обратная задача:

- определение текстурных координат, соответствующих заданному значению экранных;

• значения текстурной функции могут использоваться:

– для задания интенсивности грани;

– для определения коэффициентов отражения.

3 Код решения

Файл main.py

```
#include <GLFW/glfw3.h>
#include <cmath>
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
#include "iostream"
using std::cos, std::sin;
int mode = 1;
int lightMode = 1;
int degreeMode = 1;
int timeMode = 0;
float degree_y = 0.0;
float degree_x = 0.0;
float move_y = 0.0;
float move_x = 0.0;
float osnov_x = 0.1;
float osnov_y = 0.0;
float flying_speed = 0;
float V = 3.14 * pow(10,-4);
float acl = pow(10,-4);
int width = 1000;
int height = 1000;
GLuint textureID;
void key_callback(GLFWwindow *window, int key, int scancode, int action, int mods
{
    if (action == GLFW_PRESS || action == GLFW_REPEAT)
    {
        if (key == GLFW_KEY_ESCAPE)
        {
            glfwSetWindowShouldClose(window, GL_TRUE);
        }
```

```
else if (key == GLFW_KEY_UP)
{
    degree_y += 0.2;
}
else if (key == GLFW_KEY_DOWN)
{
    degree_y -= 0.2;
}
else if (key == GLFW_KEY_LEFT)
{
    degree_x += 0.2;
}
else if (key == GLFW_KEY_RIGHT)
{
    degree_x -= 0.2;
}
else if (key == GLFW_KEY_D)
{
    move_x += 0.2;
}
else if (key == GLFW_KEY_A)
{
    move_x -= 0.2;
}
else if (key == GLFW_KEY_W)
{
    move_y += 0.2;
}
else if (key == GLFW_KEY_S)
{
    move_y -= 0.2;
}
else if (key == GLFW_KEY_L)
```

```
{
    osnov_x += 0.1;
}
else if (key == GLFW_KEY_K)
{
    osnov_x -= 0.1;
}
else if (key == GLFW_KEY_I)
{
    osnov_y += 0.1;
}
else if (key == GLFW_KEY_O)
{
    osnov_y -= 0.1;
}
else if (key == GLFW_KEY_SPACE)
{
    mode = (mode + 1) % 2;
    if (mode == 0)
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    else
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

}
else if (key == GLFW_KEY_1)
{
    lightMode = (lightMode + 1) % 2;
    glDisable(GL_LIGHT0);
}
else if (key == GLFW_KEY_2)
{
    degreeMode = (degreeMode + 1) % 2;
}
```

```
        else if (key == GLFW_KEY_3)
        {
            timeMode = (timeMode + 1) % 2;
        }
    }
}
void light()
{
    glPushMatrix();
    glLoadIdentity();
    glTranslatef(1, 1, 1);
    GLfloat material_diffuse[] = {0.75, 0.75, 0.75, 0.0};
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, material_diffuse);
    GLfloat light2_diffuse[] = {1, 1, 1};
    GLfloat light2_position[] = {0, 0, 0, 1.0};
    glEnable(GL_LIGHT0);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light2_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light2_position);
    glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 0.0);
    glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.2);
    glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.4);
    glPopMatrix();
}
void texture()
{
    int width_1, height_1, channels;
    unsigned char* image = stbi_load("./../texture.bmp", &width_1, &height_1, &c
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEA
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEA
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_N
```

```
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NE
    if (image){
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width_1, height_1, 0, GL_
        std::cout << "1 " << std::endl;
    }
    std::cout << "2 " << std::endl;
    stbi_image_free(image);
}
void move_object()
{
    flying_speed -= V;
    V += acl;
    if(flying_speed < -2.2 or flying_speed > 2.2)
        V = -V;
}
void display(GLFWwindow* window)
{
    glClearColor (0.3, 0.3, 0.3, 0.0);
    glEnable(GL_DEPTH_TEST);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glPushMatrix();
    glTranslatef(0.0f + move_x, 0.0f + move_y + flying_speed, 0.0f);
    glRotatef(degree_y * 50.f, 1.f, 0.f, 0.f);
    glRotatef(degree_x * 50.f, 0.f, 1.f, 0.f);
    glBegin(GL_QUAD_STRIP);
    glColor3f(0.4f, 0.4f, 1.0f);
    for (int i = 0; i <= 360; i += 10)
    {
        float angle = i * M_PI / 180 ;
        glTexCoord2f(1 * cos(angle) + osnov_x, 0.5 * sin(angle) + osnov_y);
        glVertex3f(1 * cos(angle) + osnov_x, 0.5 * sin(angle) + osnov_y, 0.0);
        glTexCoord2f(1 * cos(angle), 0.5 * sin(angle));
```

```
        glVertex3f(1 * cos(angle), 0.5 * sin(angle), 1);
    }
    glEnd();
    glBegin(GL_POLYGON);
    glNormal3f(1, 1, -1);
    glColor3f(1.0f, 0.3f, 0.3f);
    for (int i = 0; i <= 360; i++)
    {
        float angle = i * M_PI / 180;
        glTexCoord2f(1 * cos(angle) + osnov_x, 0.5 * sin(angle) + osnov_y);
        glVertex3f(1 * cos(angle) + osnov_x, 0.5 * sin(angle) + osnov_y, 0.0);
    }
    glEnd();
    glBegin(GL_POLYGON);
    glNormal3f(1, 1, 1);
    glColor3f(0.5f, 0.7f, 0.7f);
    for (int i = 0; i <= 360; i++)
    {
        float angle = i * M_PI / 180;
        glTexCoord2f(1 * cos(angle), 0.5 * sin(angle));
        glVertex3f(1 * cos(angle), 0.5 * sin(angle), 1);
    }
    glEnd();
    glPopMatrix();
    GLfloat spec[] = {1, 1, 1, 1};
    GLfloat emiss[] = {0, 0, 0, 1};
    GLfloat shin = 50;
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE)
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  spec);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, &shin);
    glMaterialfv(GL_FRONT_AND_BACK, GL_EMISSION,  emiss);
}
int main()
```

```
{
    if (!glfwInit())
        return -1;
    GLFWwindow* window = glfwCreateWindow(width, height, "Lab 6", NULL, NU
    if (!window) {
        glfwTerminate();
        return -1;
    }
    glViewport(0, 0, width, height);
    glfwMakeContextCurrent(window);
    glfwSetKeyCallback(window, key_callback);
    glScalef(0.25,0.25, 0.25);
    glEnable(GL_LIGHTING);
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
    glEnable(GL_NORMALIZE);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    texture();
    while (!glfwWindowShouldClose(window))
    {
        display(window);
        if(degreeMode)
            degree_x += 0.01;
        if(timeMode)
            move_object();
        if(lightMode)
            light();
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwTerminate();
    return 0;
}
```

4 Заключение

В данной работе я изучил возможности языка C++ в работе с библиотекой OpenGL, а именно научился создавать источники света с помощь функций: glLightfv, glLightf, glMaterialfv, glColorMaterial. А также научился накладывать текстуры на обЪект с помощью функций: glGenTextures, glTexParameterf, glTexImage2D, glTexCoord2f, glBindTexture.

5 Результат запуска

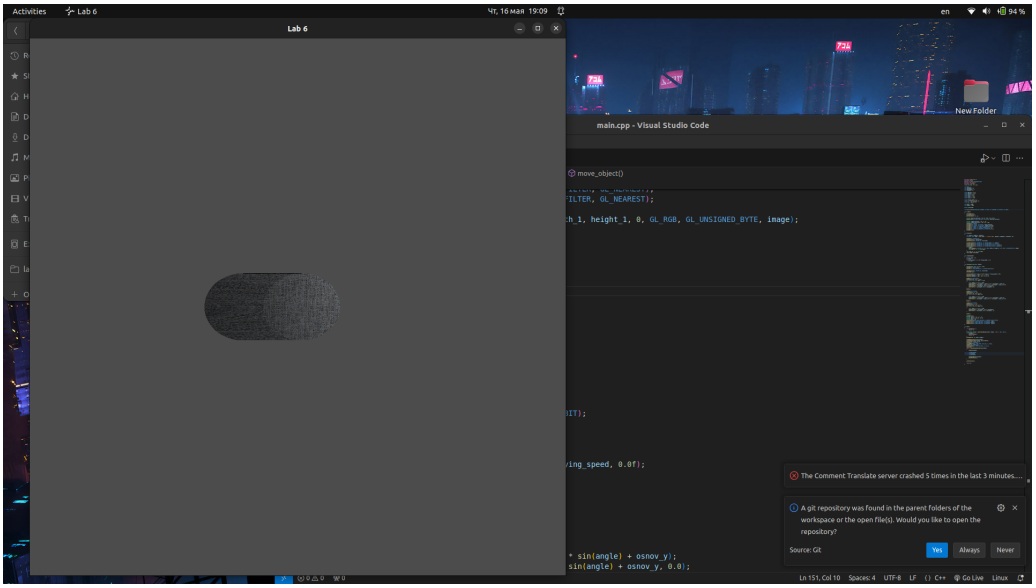

Рис. 1 — ОбЪект с текстурой и освещением белого цвета

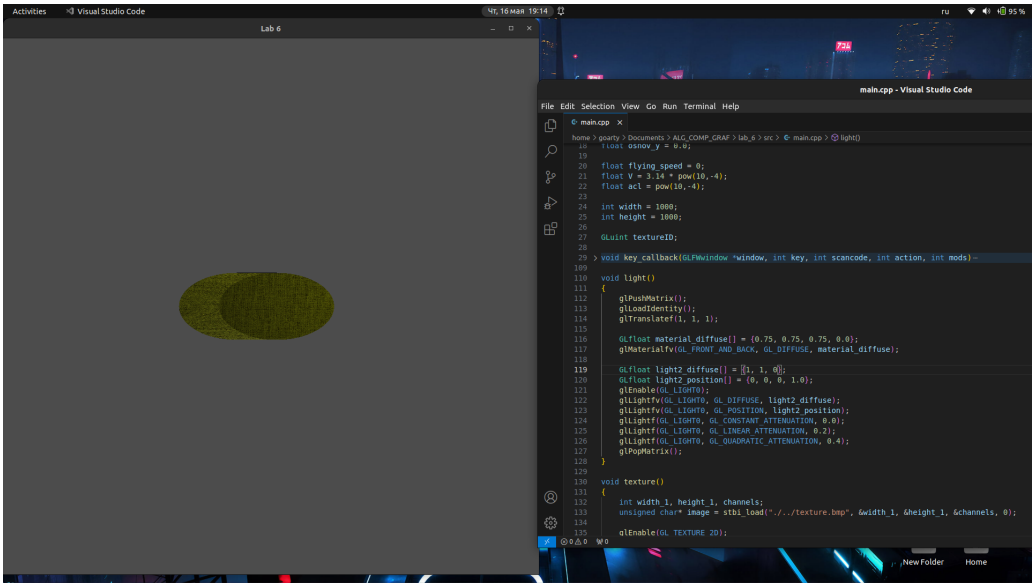

Рис. 2 — ОбЪект с текстурой и освещением жёлтого цвета