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 «Компьютерная графика»

«Построение реалистичных изображений»

Студент группы ИУ9-42Б Волохов А. В.

Преподаватель Цалкович П. А.

Москва 2024

1 Задача

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

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

Исследовать метод повышения реалистичности получаемых изображений сцены: влияние свойств материала поверхности (вариант А5);

Реализовать алгоритм анимации: моделирование движения тела (с заданной начальной скоростью) при условии абсолютно упругого отражения объекта от границ некоторого ограничивающего объема (вариант Б2);

Реализовать наложение текстуры: использование текстуры для определения интенсивности поверхности (вариант В1).

2 Основная теория

Модели освещения в OpenGL: Модель освещения в OpenGL включает несколько компонентов, позволяющих добиться реалистичного отображения объектов в 3D-пространстве. Освещение в OpenGL определяется набором источников света, которые могут быть точечными, направленными или прожекторами. Каждый источник света имеет свои свойства, такие как позиция, цвет, интенсивность и направление. Существует несколько типов освещения: окружающее (ambient), диффузное (diffuse) и зеркальное (specular). Окружающее освещение добавляет равномерный свет ко всем объектам сцены, диффузное освещение учитывает угол падения света на поверхность, а зеркальное освещение создает бликовые эффекты, имитируя отражение света от гладких поверхностей.

Свойства материалов и их влияние на реалистичность: Свойства материалов в OpenGL играют ключевую роль в том, как объекты будут взаимодействовать с источниками света. Материалы имеют параметры, такие как диффузный цвет (diffuse), зеркальный цвет (specular), эмиссивный цвет (emissive) и коэффи-

циент блеска (shininess). Диффузный цвет определяет основной цвет объекта, зеркальный цвет влияет на интенсивность бликов, эмиссивный цвет создает эффект самосветящегося материала, а коэффициент блеска определяет размер и резкость бликов. Комбинирование этих свойств позволяет создавать материалы, которые выглядят реалистично в различных условиях освещения.

Методы повышения реалистичности: Для повышения реалистичности изображений сцены, важно учитывать различные методы и техники. Один из таких методов - использование текстур. Текстуры позволяют добавлять детали и вариации на поверхность объектов, имитируя сложные узоры и неоднородности, которые трудно создать с помощью только цветовых параметров. Еще один метод - анимация объектов с учетом физики. Реалистичное движение объектов, например, с учетом начальной скорости и упругих столкновений с границами, добавляет динамику и правдоподобность сцене.

Анимация и моделирование движения: Анимация в OpenGL позволяет создавать движущиеся объекты, что добавляет реалистичность и динамику в сцены. В данном контексте моделирование движения тела с заданной начальной скоростью и условием абсолютно упругого отражения от границ объема представляет собой простую физическую симуляцию. При этом объект изменяет свою позицию в пространстве с течением времени, а при столкновении с границами сцены его скорость изменяет направление, имитируя упругое столкновение. Такой подход позволяет наблюдать за естественным движением объектов и их взаимодействием с окружением.

3 Практическая реализация

Код представлен в Листинге 1.

Листинг 1 - lab6.py

```
import math

import glfw
import numpy as np
from OpenGL.GL import *
from math import cos, sin

from PIL import Image

alpha = 0
beta = 0
size = 0.5
fill = True

torus_position = [0.0, 0.0, 0.0]
torus_velocity = [0.0001, 0.0002, 0.0005]


def main():
    if not glfw.init():
        return
    window = glfw.create_window(640, 640, "LAB 6", None, None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
    glfw.set_scroll_callback(window, scroll_callback)

    glEnable(GL_DEPTH_TEST)
    glDepthFunc(GL_LESS)

    setup_lighting()
    texture_id = load_texture("texture.jpg")

    while not glfw.window_should_close(window):
        display(window, texture_id)
    glfw.destroy_window(window)
    glfw.terminate()
```

```
def setup_lighting():
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)

    light_pos = [10.0, 10.0, 10.0, 1.0]
    glLightfv(GL_LIGHT0, GL_POSITION, light_pos)

    light_diffuse = [1.0, 1.0, 1.0, 1.0]
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse)
    light_specular = [1.0, 1.0, 1.0, 1.0]
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular)

    material_diffuse = [0.8, 0.8, 0.8, 1.0]
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material_diffuse)
    material_specular = [1.0, 1.0, 1.0, 1.0]
    glMaterialfv(GL_FRONT, GL_SPECULAR, material_specular)
    material_shininess = [100.0]
    glMaterialfv(GL_FRONT, GL_SHININESS, material_shininess)

def load_texture(filename):
    img = Image.open(filename)
    img_data = np.array(list(img.getdata()), np.uint8)
    texture_id = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, texture_id)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.width, img.height, 0, GL_RGB,
        GL_UNSIGNED_BYTE, img_data)
    return texture_id


def display(window, texture_id):
    global alpha
    global beta
    global size
    global torus_position
    global torus_velocity

    glLoadIdentity()
    glClear(GL_COLOR_BUFFER_BIT)
    glClear(GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_PROJECTION)

    def projection():
        alpha_rad = np.radians(alpha)
```

```
        beta_rad = np.radians(beta)

        rotate_y = np.array([
            [cos(alpha_rad), 0, sin(alpha_rad), 0],
            [0, 1, 0, 0],
            [-sin(alpha_rad), 0, cos(alpha_rad), 0],
            [0, 0, 0, 1]
        ])

        rotate_x = np.array([
            [1, 0, 0, 0],
            [0, cos(beta_rad), -sin(beta_rad), 0],
            [0, sin(beta_rad), cos(beta_rad), 0],
            [0, 0, 0, 1]
        ])

        glMultMatrixf(rotate_x)
        glMultMatrixf(rotate_y)

    def torus(R, r, N, n):
        glEnable(GL_TEXTURE_2D)
        glBindTexture(GL_TEXTURE_2D, texture_id)

        for i in range(N):
            for j in range(n):
                theta = (2 * math.pi / N) * i
                phi = (2 * math.pi / n) * j
                theta_next = (2 * math.pi / N) * (i + 1)
                phi_next = (2 * math.pi / n) * (j + 1)

                x0 = (R + r * cos(phi)) * cos(theta)
                y0 = (R + r * cos(phi)) * sin(theta)
                z0 = r * sin(phi)

                x1 = (R + r * cos(phi)) * cos(theta_next)
                y1 = (R + r * cos(phi)) * sin(theta_next)
                z1 = r * sin(phi)

                x2 = (R + r * cos(phi_next)) * cos(theta_next)
                y2 = (R + r * cos(phi_next)) * sin(theta_next)
                z2 = r * sin(phi_next)

                x3 = (R + r * cos(phi_next)) * cos(theta)
                y3 = (R + r * cos(phi_next)) * sin(theta)
                z3 = r * sin(phi_next)
```

```
                glBegin(GL_QUADS)

                glTexCoord2f(0.0, 0.0)
                glVertex3f(x0, y0, z0)

                glTexCoord2f(1.0, 0.0)
                glVertex3f(x1, y1, z1)

                glTexCoord2f(1.0, 1.0)
                glVertex3f(x2, y2, z2)

                glTexCoord2f(0.0, 1.0)
                glVertex3f(x3, y3, z3)

                glEnd()

        glDisable(GL_TEXTURE_2D)

    glLoadIdentity()

    projection()
    R = size
    r = size / 3

    for i in range(3):
        torus_position[i] += torus_velocity[i]

    for i in range(3):
        if torus_position[i] + size > 1.0 or torus_position[i] - size < -1.0:
            torus_velocity[i] *= -1.0

    glTranslatef(torus_position[0], torus_position[1], torus_position[2])
    torus(R, r, 40, 25)

    glfw.swap_buffers(window)
    glfw.poll_events()


def key_callback(window, key, scancode, action, mods):
    global alpha
    global beta

    if action == glfw.PRESS or action == glfw.REPEAT:
        if key == glfw.KEY_RIGHT:
            alpha += 3
        elif key == glfw.KEY_LEFT:
```

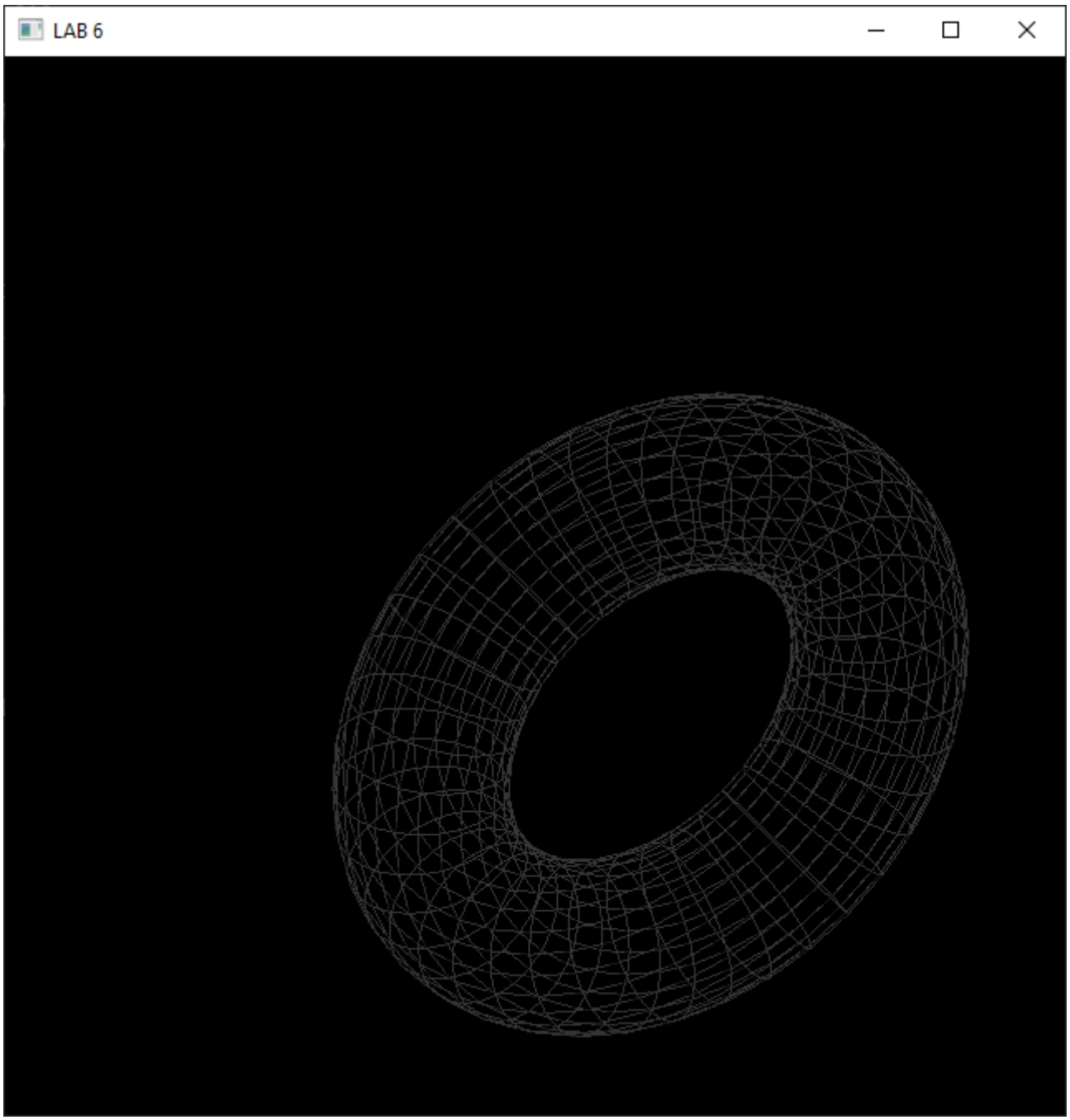
```
            alpha -= 3
        elif key == glfw.KEY_UP:
            beta -= 3
        elif key == glfw.KEY_DOWN:
            beta += 3
        elif key == glfw.KEY_F:
            global fill
            fill = not fill
            if fill:
                glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)
            else:
                glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        elif key == glfw.KEY_L and action == glfw.PRESS:
            light_enabled = glIsEnabled(GL_LIGHT0)
            if light_enabled:
                glDisable(GL_LIGHT0)
            else:
                glEnable(GL_LIGHT0)


def scroll_callback(window, xoffset, yoffset):
    global size

    if xoffset > 0:
        size -= yoffset / 10
    else:
        size += yoffset / 10


if __name__ == "__main__":
    main()
```

В результате работы программы получился следующий вывод:


LAB 6


LAB 6

4 Заключение

В данной лабораторной работе были реализованы различные аспекты создания реалистичных изображений с использованием OpenGL. Определены и настроены параметры модели освещения, исследованы свойства материалов, их влияние на внешний вид объектов, а также использованы текстуры для повышения реалистичности поверхности. Кроме того, был реализован алгоритм анимации, моделирующий движение объекта с учетом абсолютно упругого отражения от границ объема. Эти методы и техники в совокупности позволяют создавать визуально правдоподобные сцены, что является важным в изучении компьютерной графики.