Федеральное государственное бюджетное образовательное

учреждение высшего образования

«Московский государственный технический университет имени Н.Э.Баумана (национальный исследовательский университет)»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Джабаров Р.А. ИУ9-32Б

Компьютерная графика.

Лабораторная работа №8

Москва

2023

CODE:

# -\*- coding: utf-8 -\*-

import math

from OpenGL.GL import \*

from OpenGL.GLUT import \*

#import sys

from random import random

import numpy as np

global pointcolor

object\_matrix = np.identity(4)

last\_mouse\_x = 0

last\_mouse\_y = 0

n = 10

wireframe\_mode = False

def handle\_key\_press(key, x, y):

global n, scale

if key == b'q':

n -= 1

if key == b'e':

n += 1

global wireframe\_mode

if key == b'x':

wireframe\_mode = not wireframe\_mode

if wireframe\_mode:

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glDisable(GL\_CULL\_FACE)

else:

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL)

glEnable(GL\_CULL\_FACE)

elif key == b'o':

scale -= 0.1

elif key == b'p':

scale += 0.1

glutPostRedisplay()

def mouse\_wheel\_callback(wheel, direction, x, y):

global scale

if direction > 0:

scale += 0.1

else:

scale -= 0.1

glutPostRedisplay()

def handle\_mouse\_down(button, state, x, y):

global mouse\_down, last\_mouse\_x, last\_mouse\_y

if button == 1 and state == 1:

last\_mouse\_x = x

last\_mouse\_y = y

def handle\_mouse\_move(x, y):

global object\_matrix, last\_mouse\_x, last\_mouse\_y

delta\_x = x - last\_mouse\_x

delta\_y = y - last\_mouse\_y

rotation\_y = np.identity(4)

rotation\_x = np.identity(4)

rotation\_y[0, 0] = np.cos(delta\_x / 100.0)

rotation\_y[2, 0] = -np.sin(delta\_x / 100.0)

rotation\_y[0, 2] = np.sin(delta\_x / 100.0)

rotation\_y[2, 2] = np.cos(delta\_x / 100.0)

rotation\_x[1, 1] = np.cos(delta\_y / 100.0)

rotation\_x[2, 1] = np.sin(delta\_y / 100.0)

rotation\_x[1, 2] = -np.sin(delta\_y / 100.0)

rotation\_x[2, 2] = np.cos(delta\_y / 100.0)

object\_matrix = np.dot(rotation\_x, object\_matrix)

object\_matrix = np.dot(rotation\_y, object\_matrix)

last\_mouse\_x = x

last\_mouse\_y = y

glutPostRedisplay()

scale = 1.0

def specialkeys(key, x, y):

global pointcolor

if key == GLUT\_KEY\_UP:

glRotatef(5, 1, 0, 0)

if key == GLUT\_KEY\_DOWN:

glRotatef(-5, 1, 0, 0)

if key == GLUT\_KEY\_LEFT:

glRotatef(5, 0, 1, 0)

if key == GLUT\_KEY\_RIGHT:

glRotatef(-5, 0, 1, 0)

if key == GLUT\_KEY\_END:

pointcolor = [[random(), random(), random()], [random(), random(), random()], [random(), random(), random()]]

def create\_shader(shader\_type, source):

shader = glCreateShader(shader\_type)

glShaderSource(shader, source)

glCompileShader(shader)

return shader

globalMas = []

pointdata = []

def draw():

global n, globalMas, scale, pointdata

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glEnableClientState(GL\_VERTEX\_ARRAY)

glEnableClientState(GL\_COLOR\_ARRAY)

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

vectors = []

for i in range(n+1):

theta = 2 \* math.pi \* i / n

for j in range(n+1):

phi = math.pi \* j / n

x = math.cos(theta) \* math.sin(phi)

y = math.sin(theta) \* math.sin(phi)

z = math.cos(phi)

vectors.append([x, y, z])

globalMas.append(vectors)

vectors = []

for i in range(len(globalMas) - 1):

for j in range(len(globalMas[0]) - 1):

pointdata.append(globalMas[i + 1][j][0])

pointdata.append(globalMas[i + 1][j][1])

pointdata.append(globalMas[i + 1][j][2])

pointdata.append(globalMas[i][j][0])

pointdata.append(globalMas[i][j][1])

pointdata.append(globalMas[i][j][2])

pointdata.append(globalMas[i][j + 1][0])

pointdata.append(globalMas[i][j + 1][1])

pointdata.append(globalMas[i][j + 1][2])

pointdata.append(globalMas[i + 1][j + 1][0])

pointdata.append(globalMas[i + 1][j + 1][1])

pointdata.append(globalMas[i + 1][j + 1][2])

#globalMas = np.array(globalMas)

#print((globalMas))

glPushMatrix()

glMultMatrixf(object\_matrix)

glScalef(scale\*0.25, scale\*0.25, scale\*0.25)

glVertexPointer(3, GL\_FLOAT, 0, pointdata)

glColorPointer(3, GL\_FLOAT, 0, pointcolor)

glDrawArrays(GL\_QUADS, 0, 12 \* n \* n)

glDisableClientState(GL\_VERTEX\_ARRAY)

glDisableClientState(GL\_COLOR\_ARRAY)

glPopMatrix()

glutSwapBuffers()

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB)

glutInitWindowSize(800, 800)

glutInitWindowPosition(50, 50)

glutInit()

glutCreateWindow(b"Shaders!")

glutReshapeFunc(lambda w, h: glViewport(0, 0, w, h))

glEnable(GL\_CULL\_FACE)

glutDisplayFunc(draw)

print(globalMas)

glutMouseFunc(handle\_mouse\_down)

glutMotionFunc(handle\_mouse\_move)

glutIdleFunc(draw)

glutSpecialFunc(specialkeys)

glClearColor(0.2, 0.2, 0.2, 1)

vertex = create\_shader(GL\_VERTEX\_SHADER, """

varying vec4 vertex\_color;

void main(){

gl\_Position = gl\_ModelViewProjectionMatrix \* gl\_Vertex;

vertex\_color = gl\_Color;

}""")

fragment = create\_shader(GL\_FRAGMENT\_SHADER, """

varying vec4 vertex\_color;

void main() {

gl\_FragColor = vertex\_color;

}""")

program = glCreateProgram()

glAttachShader(program, vertex)

glAttachShader(program, fragment)

glLinkProgram(program)

glUseProgram(program)

print(globalMas)

print(globalMas)

print(globalMas)

print(globalMas)

pointcolor = []

print(pointdata)

for i in range(12 \* n \* n):

pointcolor.append([1, i%2, 0])

glutMainLoop()

I. Теория:

OpenGL:

OpenGL (Open Graphics Library) - это кросс-платформенный интерфейс программирования приложений (API) для работы с графикой.

Он предоставляет функции и возможности для рендеринга трехмерных объектов, настройки освещения, работы с текстурами и другими графическими эффектами.

GLUT:

GLUT (OpenGL Utility Toolkit) - это набор инструментов для работы с окнами, обработки событий и управления вводом в OpenGL.

Он предоставляет функции для создания окна, обработки событий мыши и клавиатуры, а также для отрисовки сцены.

Матрицы и преобразования:

В данном коде используются матрицы для преобразования объектов в сцене.

Матрица object\_matrix представляет матрицу моделирования-вида и используется для вращения объектов с помощью мыши.

Для вращения объектов используются матрицы поворота rotation\_x и rotation\_y, которые формируются на основе перемещения мыши.

Шейдеры:

Шейдеры - это программы, выполняющиеся на графическом процессоре (GPU) и управляющие процессом рендеринга в OpenGL.

В данном коде создаются вершинный и фрагментный шейдеры с помощью функции create\_shader.

Шейдеры компилируются и привязываются к программе с помощью функций glCompileShader и glAttachShader.

II. Практическая реализация:

Инициализация окна и OpenGL:

В функции glutInitDisplayMode задается режим отображения окна, включая двойную буферизацию и использование RGB цветов.

Задаются размеры окна, его позиция и создается окно с помощью функции glutCreateWindow.

Устанавливается функция обратного вызова glutReshapeFunc для обработки изменений размеров окна.

Обработка событий мыши и клавиатуры:

Функция handle\_key\_press обрабатывает нажатия клавиш клавиатуры.

Она изменяет параметры, такие как количество сегментов n и режим отображения каркаса wireframe\_mode.

Функция mouse\_wheel\_callback обрабатывает вращение колесика мыши для масштабирования объектов.

Функции handle\_mouse\_down и handle\_mouse\_move обрабатывают движения мыши для вращения объектов.

Отрисовка сцены:

Функция draw отвечает за отрисовку сцены.

Вначале очищаются буферы цвета и глубины с помощью функции glClear.

Затем включаются клиентские состояния массивов вершин и цветов с помощью функций glEnableClientState.

Задается матрица моделирования-вида с помощью функции glMultMatrixf, основанная на матрице object\_matrix и масштабе scale.

Функции glVertexPointer и glColorPointer указывают на массивы вершин и цветов соответственно.

Функция glDrawArrays выполняет отрисовку объектов.

В конце отключаются клиентские состояния массивов вершин и цветов с помощью функций glDisableClientState.

Функция glutSwapBuffers обновляет экран, отображая новый кадр.

Инициализация шейдеров:

Создаются вершинный и фрагментный шейдеры с помощью функции create\_shader.

Шейдеры компилируются и привязываются к программе с помощью функций glCompileShader и glAttachShader.

Созданная программа активируется с помощью функции glUseProgram.

Основной цикл приложения:

Вызывается функция glutMainLoop, которая запускает основной цикл приложения.

Он обрабатывает события мыши и клавиатуры, а также вызывает функцию draw для отрисовки сцены.

III. Заключение:

Данный код демонстрирует использование OpenGL для отображения трехмерного объекта (куба) и его взаимодействия с помощью мыши и клавиатуры. Он использует GLUT для создания окна и обработки событий, а также шейдеры для настройки процесса рендеринга. Код позволяет изменять количество сегментов куба, включать/выключать режим каркаса, вращать и масштабировать объекты.