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 №4

Москва

2023

CODE:

from OpenGL.GL import \*

import glfw

from PIL import Image

import numpy

import math

G = 5

INITIAL\_CUBE\_VELOCITY = 0 # начальная скорость куба

CUBE\_HEIGHT\_RANGE = (0, 10000)

cube\_velocity = INITIAL\_CUBE\_VELOCITY

cube\_height = CUBE\_HEIGHT\_RANGE[1]

theta = 0

rot = 0

scale = 1

is\_texturing\_enabled = True

def normalize(x, x\_range, normalization\_range):

a, b = normalization\_range

x\_min, x\_max = x\_range

return (b - a) \* ((x - x\_min) / (x\_max - x\_min)) + a

def program():

if not glfw.init():

return

window = glfw.create\_window(800, 800, "Lab6", None, None)

if not window:

glfw.terminate()

return

glfw.make\_context\_current(window)

glfw.set\_key\_callback(window, key\_callback)

setup()

while not glfw.window\_should\_close(window):

prepare()

display()

glfw.swap\_buffers(window)

glfw.poll\_events()

glfw.destroy\_window(window)

glfw.terminate()

def key\_callback(window, key, scancode, action, mods):

global rot, scale, is\_texturing\_enabled

if action == glfw.REPEAT or action == glfw.PRESS:

if key == glfw.KEY\_RIGHT:

rot -= 3

if key == glfw.KEY\_LEFT:

rot += 3

if key == glfw.KEY\_UP:

scale += 0.02

if key == glfw.KEY\_DOWN:

scale -= 0.02

if key == glfw.KEY\_C:

is\_texturing\_enabled = not is\_texturing\_enabled

def enable\_texturing():

global is\_texturing\_enabled

if is\_texturing\_enabled:

glEnable(GL\_TEXTURE\_2D)

def disable\_texturing():

global is\_texturing\_enabled

if is\_texturing\_enabled:

glDisable(GL\_TEXTURE\_2D)

def setup():

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

glFrustum(-0.1, 0.1, -0.1, 0.1, 0.2, 1000)

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

glEnable(GL\_DEPTH\_TEST)

glEnable(GL\_LIGHTING)

glEnable(GL\_LIGHT0)

glEnable(GL\_NORMALIZE)

glEnable(GL\_COLOR\_MATERIAL)

glColorMaterial(GL\_FRONT, GL\_DIFFUSE)

glShadeModel(GL\_SMOOTH)

load\_texture()

def load\_texture():

img = Image.open("text.bmp")

img\_data = numpy.array(list(img.getdata()), numpy.int8)

glBindTexture(GL\_TEXTURE\_2D, glGenTextures(1))

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR)

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE)

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, img.size[0], img.size[1], 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, img\_data)

def prepare():

glClearColor(0, 0, 0, 0)

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

def display():

global CUBE\_HEIGHT\_RANGE, cube\_velocity, cube\_height, theta, rot, scale

glPushMatrix()

glRotatef(-60, 1, 0, 0)

glRotatef(33, 0, 0, 1)

glTranslatef(2, 3, -2.5)

glRotatef(rot, 0, 0, 1)

glScalef(scale, scale, scale)

glPushMatrix()

if cube\_height - cube\_velocity > CUBE\_HEIGHT\_RANGE[0]:

cube\_height -= cube\_velocity

if cube\_velocity < 0 and cube\_velocity + G > 0:

cube\_velocity = 0

else:

cube\_velocity += G

else:

cube\_height = CUBE\_HEIGHT\_RANGE[0]

cube\_velocity = -cube\_velocity

glRotatef(45, 0, 0, 1)

glTranslatef(0, 0, normalize(cube\_height, CUBE\_HEIGHT\_RANGE, (0.5, 1)))

glScalef(0.35, 0.35, 0.35)

draw\_cube()

glPopMatrix()

glPushMatrix()

glRotatef(45, 0, 1, 0)

glLightfv(GL\_LIGHT0, GL\_POSITION, (0, 0, 1, 0))

glTranslatef(0, 0, 2)

glScalef(0.2, 0.2, 0.2)

glColor3f(1, 1, 1)

draw\_plane()

glPopMatrix()

glPopMatrix()

theta += 0.9

object\_matrix = numpy.identity(4)

n = 10

def draw\_cube():

global n

globalMas = []

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 = []

globalMas = numpy.array(globalMas)

enable\_texturing()

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

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

glBegin(GL\_POLYGON)

glNormal3f(1.0, 0.0, 0.0)

glTexCoord2f(1.0, 0.0)

glVertex3f(globalMas[i + 1][j][0], globalMas[i + 1][j][1], globalMas[i + 1][j][2])

glTexCoord2f(0.0, 0.0)

glVertex3f(globalMas[i][j][0], globalMas[i][j][1], globalMas[i][j][2])

glTexCoord2f(0.0, 1.0)

glVertex3f(globalMas[i][j + 1][0], globalMas[i][j + 1][1], globalMas[i][j + 1][2])

glTexCoord2f(1.0, 1.0)

glVertex3f(globalMas[i + 1][j + 1][0], globalMas[i + 1][j + 1][1], globalMas[i + 1][j + 1][2])

glEnd()

disable\_texturing()

def draw\_plane():

verticies = (

-1, -1, 0,

1, -1, 0,

1, 1, 0,

-1, 1, 0

)

normals = (

-1, -1, 3,

1, -1, 3,

1, 1, 3,

-1, 1, 3

)

glEnableClientState(GL\_VERTEX\_ARRAY)

glEnableClientState(GL\_NORMAL\_ARRAY)

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

glNormalPointer(GL\_FLOAT, 0, normals)

glDrawArrays(GL\_TRIANGLE\_FAN, 0, 4)

glDisableClientState(GL\_VERTEX\_ARRAY)

glDisableClientState(GL\_NORMAL\_ARRAY)

def main():

try:

program()

except Exception as e:

print('Error')

print(e)

if \_\_name\_\_ == '\_\_main\_\_':

main()

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.

GLFW:

GLFW (Graphics Library Framework) - это библиотека для работы с окнами и обработки ввода в контексте OpenGL.

Она предоставляет функциональность для создания окна, управления событиями мыши и клавиатуры, установки функций обратного вызова и других операций, необходимых для работы с OpenGL.

PIL (Python Imaging Library):

PIL - это библиотека для работы с изображениями в Python.

В данном коде PIL используется для загрузки текстуры из файла "text.bmp" и преобразования ее в формат, совместимый с OpenGL.

numpy:

numpy - это библиотека для работы с массивами и матрицами в Python.

В данном коде numpy используется для создания и манипуляций с массивами точек, которые представляют вершины куба и плоскости.

II. Практическая реализация:

Инициализация и настройка окна:

В функции program происходит инициализация окна и контекста OpenGL с помощью библиотеки GLFW.

Устанавливаются функции обратного вызова для обработки событий клавиатуры.

Выполняется настройка OpenGL, включая настройку проекции, включение освещения, настройку текстуры и другие параметры.

Функции обратного вызова:

Функция key\_callback вызывается при нажатии клавиш клавиатуры.

Она обрабатывает нажатия определенных клавиш, влияющих на вращение и масштабирование объектов, а также включение/выключение текстурирования.

Загрузка текстуры:

Функция load\_texture использует библиотеку PIL для загрузки текстуры из файла "text.bmp".

Текстура привязывается к контексту OpenGL и настраиваются параметры ее поведения, такие как повторение и фильтрация.

Подготовка к отображению:

Функция prepare устанавливает цвет очистки буферов кадра и очищает буферы цвета и глубины.

Отображение:

Функция display отвечает за отображение сцены.

Она применяет преобразования для положения и ориентации объектов, а также для анимации движения куба.

Внутри функции вызываются функции draw\_cube и draw\_plane для отрисовки куба и плоскости соответственно.

Функция draw\_cube:

Функция draw\_cube рисует куб, состоящий из полигонов.

Для построения геометрии куба используется параметр n, который определяет количество сегментов в каждом измерении.

Функция использует массив точек, созданных с помощью numpy, для определения вершин полигонов.

Полигоны заполняются текстурой, если текстурирование включено.

Функция draw\_plane:

Функция draw\_plane рисует плоскость.

Вершины плоскости и их нормали определяются вручную.

Плоскость отображается с помощью функций glVertexPointer и glNormalPointer.

III. Заключение:

Данный код реализует простую программу с использованием OpenGL для отображения куба и плоскости. Она позволяет пользователю вращать и масштабировать объекты, а также включать и выключать текстурирование. Код демонстрирует применение функций библиотек GLFW, PIL и numpy для работы с окном, текстурами и массивами соответственно.