Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6 по курсу**

**«Компьютерная графика»**

Выполнил: Лисин Р.С.

Группа: М8О-306Б-20

Преподаватель: Филиппов Г.С.

Дата: \_\_\_\_\_\_\_\_\_\_\_

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Лабораторная работа №6**

**Создание шейдерных анимационных эффектов в OpenGL**

**Задание:** Для поверхности, созданной в л.р. №5, обеспечить выполнение следующего шейдерного эффекта.

**Вариант 5:** Анимация. Координата X изменяется по закону X \* cos(t), координата Y изменяется по закону Y = Y \* sin(X + t).

**Описание**

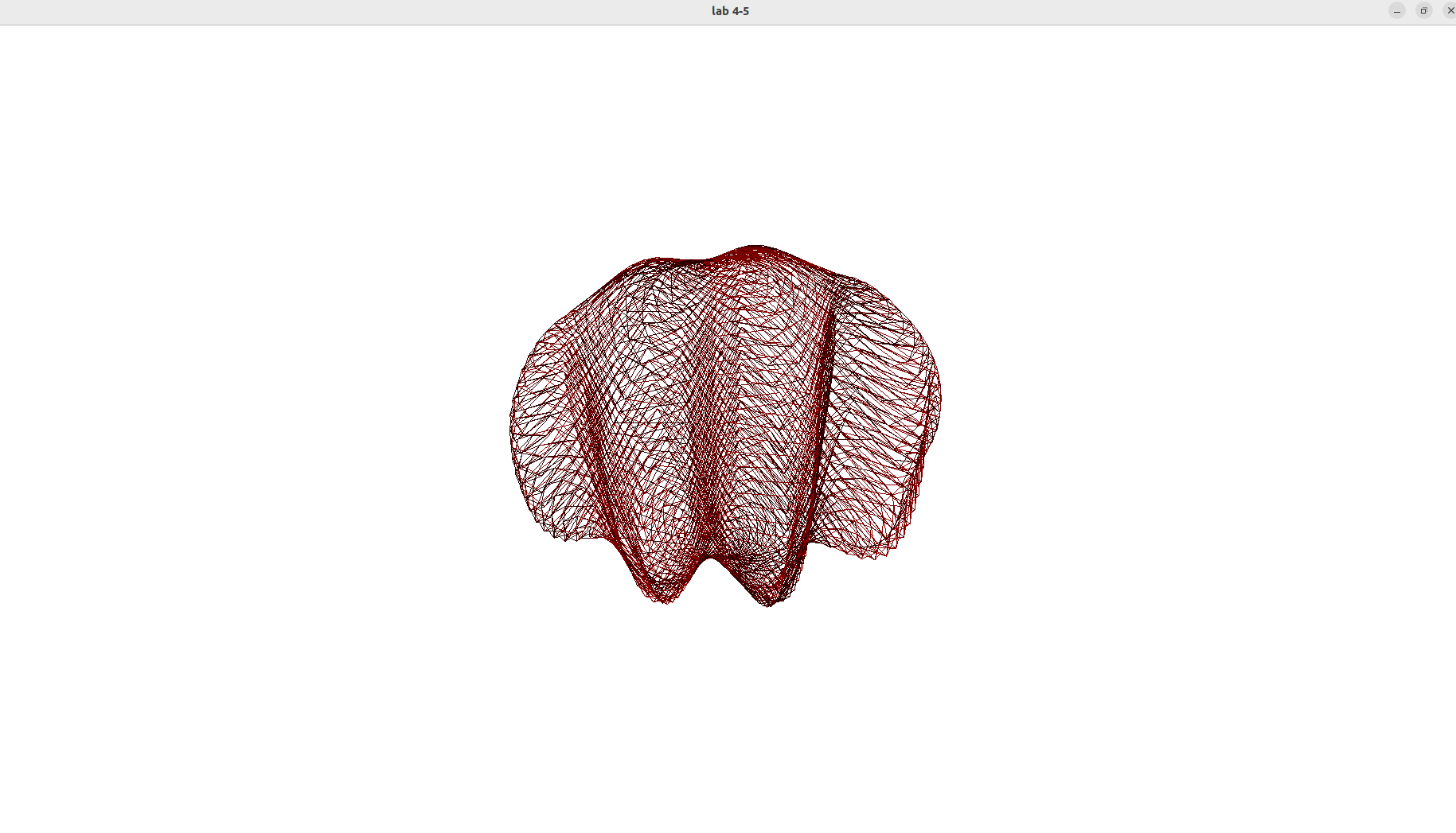
Для выполнения этой работы я воспользовался библиотекой PyOpenGL. Это аналог обычного OpenGL, но для языка Python. В программе предусмотрена возможность вращения фигуры по всем осям, изменения размера фигуры, изменения интенсивности источника рассеянного освещения и изменения параметра аппроксимации. Всё это выполняется при нажатии определённых клавиш на клавиатуре. Настройки управления выводятся пользователю в консоли при запуске программы.

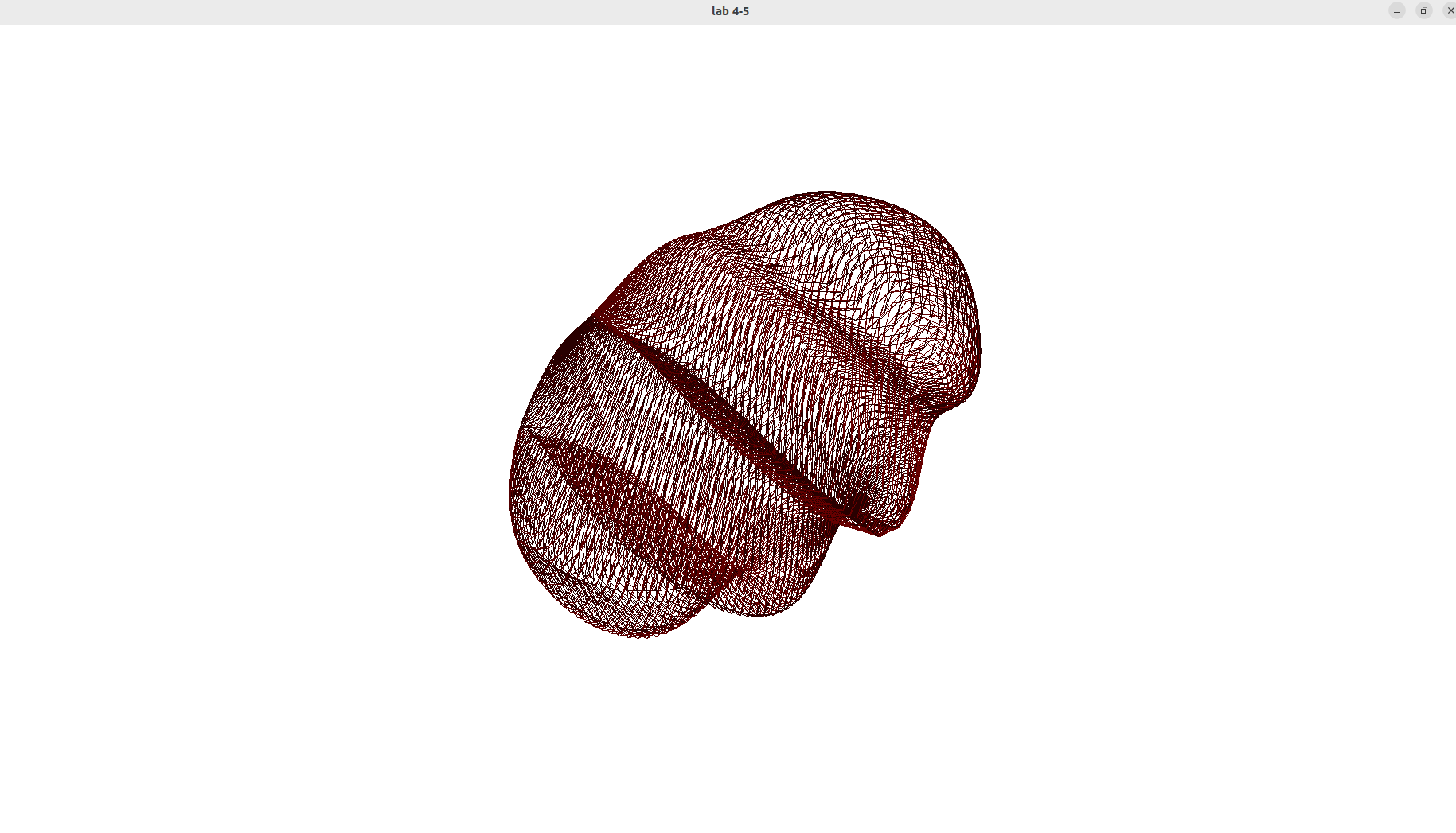
Сначала я изменил функцию отрисовки эллипсоида. В соответствии с заданием изменил законы для координат x и y, меняя t от 0 до 2π по кругу. Для выполнения шейдерного эффекта я создаю фоновый поток, в который передаю функцию, которая просто бесконечно перерисовывает фигуру с некоторым перерывом для плавности анимации.

**Исходный код**

import math  
import numpy as np  
import time  
import threading  
import sys  
  
from OpenGL.GL import \*  
from OpenGL.GLU import \*  
from OpenGL.GLUT import \*  
  
  
# параметры освещения  
light\_pos = (20, 30, 30) # положение источника света  
light\_intensity = 5 # интенсивность света  
reflection = 115 # параметр отражения  
# фоновое освещение - окружающее освещеие, которое всегда будет придавать объекту некоторый оттенок  
ambient = [0.8, 0.0, 0.0, 0.5]  
# диффузное освещение - имитирует воздействие на объект направленного источника света  
diffuse = [1.0, 0.0, 0.0, light\_intensity]  
# зеркальный свет - устанавливает цвет блика на объекте  
specular = [1.0, 0.0, 0.0, light\_intensity]  
  
# вращение  
x\_rot = 0  
y\_rot = -40  
z\_rot = 0  
  
# параметры эллипсоида  
approximation = 35 # количество образующих  
size = 1  
a, b, c = 6, 4, 4  
cur\_t = 0  
  
  
def init():  
 glClearColor(255, 255, 255, 1.0) # белый цвет для первоначальной закраски  
 glClearDepth(1.0)  
 glEnable(GL\_DEPTH\_TEST) # обновляем буфер глубины  
 glDepthFunc(GL\_LEQUAL)  
 glHint(GL\_POLYGON\_SMOOTH\_HINT, GL\_NICEST) # сглаженные полигоны, больше пикселей для отрисовки  
 glHint(GL\_PERSPECTIVE\_CORRECTION\_HINT, GL\_NICEST) # хорошее качество текстур, цветов  
 glEnable(GL\_NORMALIZE)  
 glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, ambient) # определяем текущую модель освещения  
 glEnable(GL\_LIGHTING) # включаем освещение  
 glLightModelf(GL\_LIGHT\_MODEL\_TWO\_SIDE, GL\_TRUE) # вершины заднего многоугольника зажигаются с помощью параметров  
 # заднего материала и имеют обратную норму перед вычислением уравнения освещения  
  
  
def ellipsoid():  
 global a, b, c, approximation, cur\_t  
 latitude\_delta = math.pi / approximation  
 longitude\_delta = 2 \* math.pi / approximation  
 vertices = []  
 t = np.linspace(0, 2 \* np.pi, 1000)  
 if cur\_t >= len(t):  
 cur\_t = 0  
  
 for i in range(approximation + 1):  
 lat = i \* latitude\_delta  
 for j in range(approximation + 1):  
 lon = j \* longitude\_delta  
 x = a \* math.sin(lat) \* math.cos(lon)  
 y = b \* math.sin(lat) \* math.sin(lon) \* math.sin(x + t[cur\_t])  
 x \*= math.cos(t[cur\_t])  
 z = c \* math.cos(lat)  
 vertices.append([x, y, z])  
 for i in range(approximation + 1):  
 glBegin(GL\_TRIANGLE\_STRIP)  
 for j in range(approximation + 1):  
 glVertex3fv(vertices[j + i \* approximation])  
 glVertex3fv(vertices[j + (i + 1) \* approximation])  
 glEnd()  
 cur\_t += 1  
  
  
def display():  
 glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(10, 10, 10, 0, 0, 0, 0, 0, 2)  
 glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)  
 glTranslatef(size, size, size)  
 init\_lighting()  
 glRotatef(x\_rot, 1, 0, 0)  
 glRotatef(y\_rot, 0, 0, 1)  
 glRotatef(z\_rot, 0, 1, 0)  
  
 glPushMatrix() # сохраняем текущее положение "камеры"  
 glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, diffuse)  
 glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, specular)  
 glMaterialf(GL\_FRONT\_AND\_BACK, GL\_SHININESS, 128 - reflection)  
 ellipsoid()  
 glPopMatrix() # возвращаем сохраненное положение "камеры"  
 glutSwapBuffers() # выводим все нарисованное в памяти на экран  
  
  
def init\_lighting():  
 glEnable(GL\_LIGHT0) # включаем один источник света  
 glLightfv(GL\_LIGHT0, GL\_POSITION, light\_pos) # определяем положение источника света  
  
 l\_dif = (2.0, 2.0, 3.0, light\_intensity)  
 glLightfv(GL\_LIGHT0, GL\_DIFFUSE, l\_dif)  
 l\_dir = (light\_pos[0], light\_pos[1], light\_pos[2], 1.0)  
 glLightfv(GL\_LIGHT0, GL\_POSITION, l\_dir)  
  
 # делаем затухание света  
 attenuation = float(101 - light\_intensity) / 25.0  
 distance = math.sqrt(pow(light\_pos[0], 2) + pow(light\_pos[1], 2) + pow(light\_pos[2], 2))  
 constant\_attenuation = attenuation / 3.0  
 linear\_attenuation = attenuation / (3.0 \* distance)  
 quadratic\_attenuation = attenuation / (3.0 \* distance \* distance)  
 glLightf(GL\_LIGHT0, GL\_CONSTANT\_ATTENUATION, constant\_attenuation)  
 glLightf(GL\_LIGHT0, GL\_LINEAR\_ATTENUATION, linear\_attenuation)  
 glLightf(GL\_LIGHT0, GL\_QUADRATIC\_ATTENUATION, quadratic\_attenuation)  
  
  
def reshape(width, height):  
 glViewport(0, 0, width, height)  
 glMatrixMode(GL\_PROJECTION)  
 glLoadIdentity()  
 gluPerspective(60, float(width) / float(height), 1.0, 60.0) # 1) угол, под которым пользователь видит фигуру, по y;  
 # 2) отношение x/y, которое задаёт положение по x; 3) расстояние до ближней плоскости; 4) до дальней плоскости  
 glMatrixMode(GL\_MODELVIEW)  
 glLoadIdentity()  
 gluLookAt(0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1, 0)  
  
  
def specialkeys(key, x, y):  
 global x\_rot, y\_rot, z\_rot, size, approximation, light\_intensity  
 if key == b'w':  
 x\_rot += 5 # вращаем на 5 градусов по оси X  
 if key == b's':  
 x\_rot -= 5 # вращаем на -5 градусов по оси X  
 if key == b'a':  
 y\_rot += 5 # вращаем на 5 градусов по оси Y  
 if key == b'd':  
 y\_rot -= 5 # вращаем на -5 градусов по оси Y  
 if key == b'q':  
 z\_rot += 5 # вращаем на 5 градусов по оси Z  
 if key == b'e':  
 z\_rot -= 5 # вращаем на -5 градусов по оси Z  
 if key == b'=':  
 size += 1 # увеличиваем размер на 1  
 if key == b'-':  
 size -= 1 # уменьшаем размер на 1  
 if key == b'p':  
 approximation += 1 # увеличиваем число образующих на 1  
 if key == b'o':  
 approximation -= 1 # уменьшаем число образующих на 1  
 approximation = max(10, approximation)  
 if key == b'l':  
 light\_intensity += 5 # увеличиваем интенсивность света на 5  
 light\_intensity = min(100, light\_intensity)  
 if key == b'k':  
 light\_intensity -= 5 # уменьшаем интенсивность света на 5  
 light\_intensity = max(-100, light\_intensity)  
  
 glutPostRedisplay() # вызываем процедуру перерисовки  
  
  
def change\_coordinates():  
 while True:  
 glutPostRedisplay()  
 time.sleep(0.01)  
  
  
def main():  
 glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH) # используем двойную буферизацию и формат RGB  
 glutInitWindowSize(500, 500)  
 glutInitWindowPosition(0, 0)  
 glutInit(sys.argv) # инициализируем opengl  
 glutCreateWindow("lab 4-5")  
 glutDisplayFunc(display) # определяем функцию для отрисовки  
 glutReshapeFunc(reshape) # определяем функцию для масштабирования  
 glutKeyboardFunc(specialkeys) # определяем функцию для обработки нажатия клавиш  
 init()  
  
 t = threading.Thread(target=change\_coordinates)  
 t.daemon = True  
 t.start()  
  
 glutMainLoop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Rotation:")  
 print("OX: W S")  
 print("OY: A D")  
 print("OZ: Q E")  
 print()  
 print("Change figure size: - +")  
 print("Change approximation: o p")  
 print("Change light intensity: k l")  
 main()

**Результат работы**

****

****

**Выводы**

Выполнив эту лабораторную работу, я научился выполнять шейдерные эффекты для трёхмерных фигур. Для этого нужно просто использовать фоновый поток.