Московский Авиационный Институт(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторные работы №3-6 по курсу “Компьютерная Графика”

Студент:	Т.А.Габдуллин
Преподаватель:	Г. С. Филиппов
Группа:	М8О-306Б
Оценка:	
Подпись:	

Лабораторные работы №3-6

Тема:Основы построения фотореалистичных изображений, Ознакомление с технологией OpenGL, Создание шейдерных анимационных эффектов в OpenGL

Задача: Аппроксимировать заданное тело выпуклым многогранником с использованием OpenGL. Создать шейдерные анимационные эффекты. Точность аппроксимации задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель закраски для случая одного источника света. Параметры освещения и отражающие свойства материала задаются пользователем в диалоговом режиме.

Вариант:4-гранная усеченная правильная пирамида

Исходныйкод

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import numpy
import sys
import threading
import time
from itertools import cycle

xrot = 0
yrot = 0
zrot = 0
r1 = 3
r2 = 1.5
h = 3.25
count = 4
intensiv = 10
reflection = 116
light_coord = (20, 30, 30)
size1 = 4

def drawBox(p):
   global xrot, yrot, count, reflection, r1, r2, count, h
   glPushMatrix()
   glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, (0.2, 0.8, 0.0, 0.8))
   glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, (0.2, 0.8, 0.0, 0.8))
   glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 128 - reflection)
   draw(r1, r2, count + 1, h)
   glPopMatrix()
   glutSwapBuffers()
```

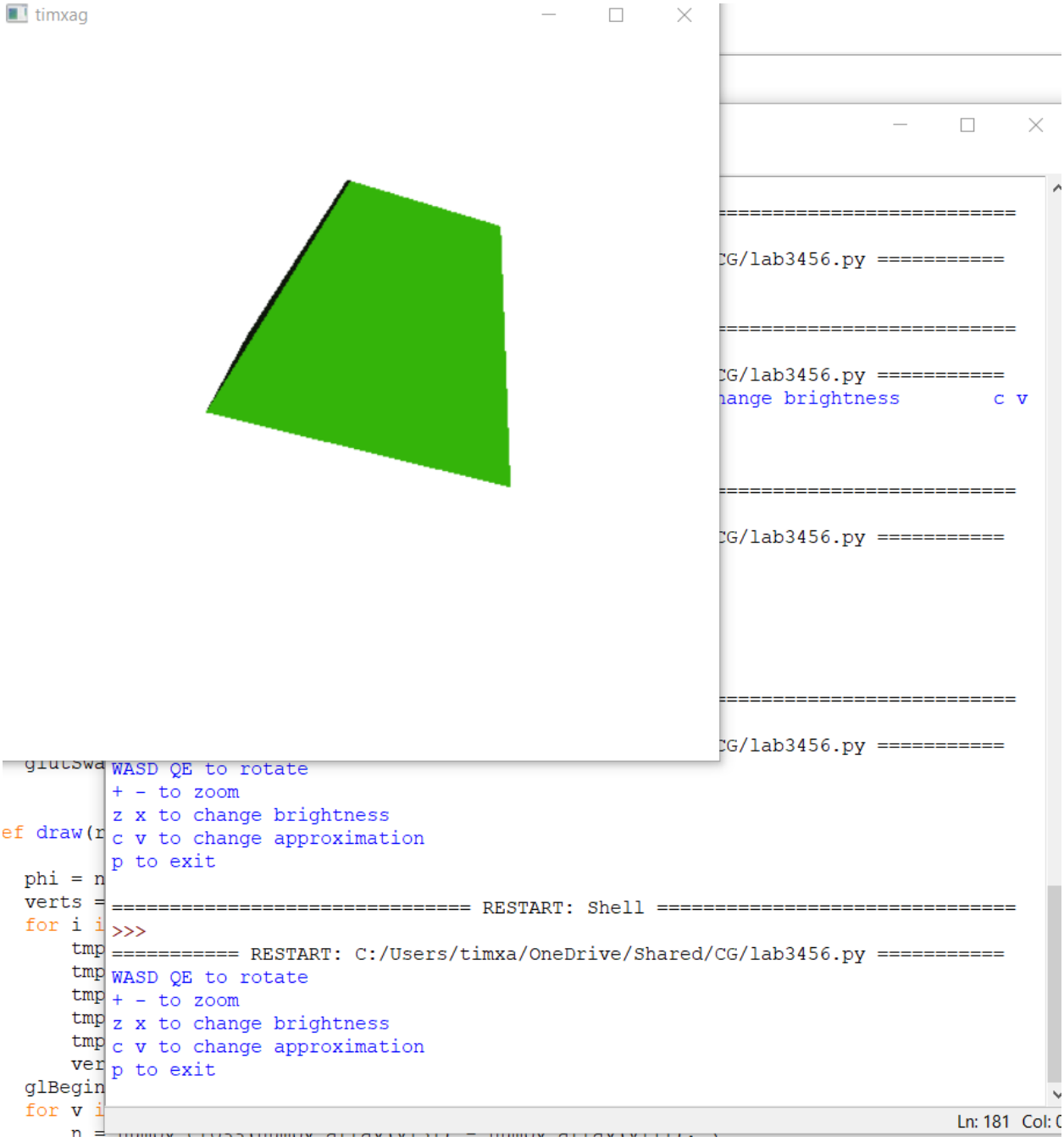
```
def draw(r1, r2, count, h):

    phi = numpy.linspace(0, 360, count)/180.0*numpy.pi
    verts = []
    for i in range(len(phi) - 1):
        tmp = []
        tmp.append((r1*numpy.cos(phi[i]), r1*numpy.sin(phi[i]), 0))
        tmp.append((r2*numpy.cos(phi[i]), r2*numpy.sin(phi[i]), h))
        tmp.append((r2*numpy.cos(phi[i+1]), r2*numpy.sin(phi[i+1]), h))
        tmp.append((r1*numpy.cos(phi[i+1]), r1*numpy.sin(phi[i+1]), 0))
        verts.append(tmp)
    glBegin(GL_QUADS)
    for v in verts:
        n = numpy.cross(numpy.array(v[3]) - numpy.array(v[1]), \
                    numpy.array(v[0]) - numpy.array(v[1]))
        glNormal3fv(n)
        glVertex3fv(v[0])
        glVertex3fv(v[1])
        glVertex3fv(v[2])
        glVertex3fv(v[3])
    glEnd()
    glBegin(GL_TRIANGLES)

    l = [(r1*numpy.cos(phi[i]), r1*numpy.sin(phi[i]), 0) for i in
range(len(phi) - 1)]
    coord_centr = numpy.array([0, 0, 0])
    l2 = [(r2*numpy.cos(phi[i]), r2*numpy.sin(phi[i]), h) for i in
range(len(phi) - 1)]
    for i in range(1, len(l)):
        n = numpy.cross(coord_centr - numpy.array(l[i]), \
                    numpy.array(l[i - 1]) - numpy.array(l[i]))
        glNormal3fv(n)
        glVertex3fv(l[i - 1])
        glVertex3fv(l[i])
        glVertex3fv(coord_centr)
    n = numpy.cross(coord_centr - numpy.array(l[0]), \
                    numpy.array(l[-1]) - numpy.array(l[0]))
    glNormal3fv(n)
    glVertex3fv(l[-1])
    glVertex3fv(l[0])
    glVertex3fv(coord_centr)
    coord_centr = numpy.array([0, 0, h])
    for i in range(1, len(l2)):
        n = numpy.cross(coord_centr - numpy.array(l2[i]), \
                    numpy.array(l2[i - 1]) - numpy.array(l2[i]))
        glNormal3fv(n)
        glVertex3fv(l2[i - 1])
        glVertex3fv(l2[i])
        glVertex3fv(coord_centr)
    n = numpy.cross(coord_centr - numpy.array(l2[0]), \
                    numpy.array(l2[-1]) - numpy.array(l2[0]))
    glNormal3fv(n)
    glVertex3fv(l2[-1])
    glVertex3fv(l2[0])
    glVertex3fv(coord_centr)

    glEnd()
```

```
def init():
    glClearColor(255, 255, 255, 1.0)
    glClearDepth(1.0)
    glEnable(GL_DEPTH_TEST)
    glShadeModel(GL_FLAT)
    glDepthFunc(GL_LEQUAL)
    glEnable(GL_DEPTH_TEST)
    glEnable(GL_NORMALIZE)
    glHint(GL_POLYGON_SMOOTH_HINT, GL_NICEST)
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)
    glEnable(GL_LIGHTING)
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE)
    glEnable(GL_NORMALIZE)


def reshape(width, height):
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(60.0, float(width)/float(height), 1.0, 60.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1, 0.0)


def display():
    global size1
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(10, 10, 10, 0, 0, 0, 0, 0, 1)
    glTranslatef(size1, size1, size1)
    lightning()
    glRotatef(xrot, 1, 0, 0)
    glRotatef(yrot, 0, 0, 1)
    glRotatef(zrot, 0, 1, 0)
    drawBox(1)


def specialkeys(key, x, y):
    global xrot, yrot, zrot, size1
    if key == b'w':
        xrot += 2
    elif key == b's':
        xrot -= 2
    elif key == b'a':
        yrot += 2
    elif key == b'd':
        yrot -= 2
    elif key == b'q':
        zrot += 2
    elif key == b'e':
        zrot -= 2
    elif key == b'=':
        size1 += 1
    elif key == b'-':
        size1 -= 1
    elif key == b'z':
```

```
        bri_change(intensiv + 5)
        lightning()
    elif key == b'x':
        bri_change(intensiv - 5)
        lightning()
    elif key == b'c':
        app_change(count + 1)
    elif key == b'v':
        app_change(count - 1)
    elif key == b'p':
       exit(0)
    glutPostRedisplay()

def lightning():
    global intensiv, light_coord
    glEnable(GL_LIGHT0)
    l_dif = (2.0, 2.0, 3.0)
    glLightfv(GL_LIGHT0,GL_DIFFUSE,l_dif)
    l_dir = (light_coord[0], light_coord[1], light_coord[2], 1.0)
    glLightfv(GL_LIGHT0,GL_POSITION,l_dir)
    att = float(101 - intensiv)/25.0
    rad = numpy.sqrt(pow(light_coord[0],2) + \
        pow(light_coord[1],2) + pow(light_coord[2],2))
    kQ = att/(3.0*rad*rad)
    kL = att/(3.0*rad)
    kC = att/3.0
    glLightf(GL_LIGHT0,GL_CONSTANT_ATTENUATION,kC)
    glLightf(GL_LIGHT0,GL_LINEAR_ATTENUATION,kL)
    glLightf(GL_LIGHT0,GL_QUADRATIC_ATTENUATION,kQ)
    glEnable(GL_LIGHT1)
    l_dif1 = (0.1,0.1,0.1)
    l_dir1 = (0.0,0.0,-100.0,1.0)
    glLightfv(GL_LIGHT1,GL_POSITION,l_dir1)
    glLightfv(GL_LIGHT1,GL_DIFFUSE,l_dif1)

def rotate():
    global zrot
    speed = [1/10000]
    for val in cycle(speed):
        begin = time.time()
        while time.time() - begin < 1:
            zrot += val
            glutPostRedisplay()

def bri_change(x):
     global intensiv
     intensiv = x
     lightning()
     glutPostRedisplay()
     return 0

def app_change(x):
     global count
     count = x
     glutPostRedisplay()
     return 0

def main():
    glutInit(sys.argv)
```

```
    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH)
    glutInitWindowSize(500, 500)
    glutInitWindowPosition(0, 0)
    glutCreateWindow(b"timxag")
    glutDisplayFunc(display)
    glutReshapeFunc(reshape)
    glutKeyboardFunc(specialkeys)
    init()
    t = threading.Thread(target=rotate)
    t.daemon = True
    t.start()
    glutMainLoop()


if __name__ == "__main__":
    print("WASD QE to rotate\n+ - to zoom\nz x to change brightness\nc v to
change approximation\np to exit")
    main()
```

Скриншоты



Выводы

Выполнив данные лабораторные работы, я познакомился с библиотекой OpenGL, которая позволяет создавать фотореалистичные изображения и трехмерные объекты.