

**Московский авиационный институт
(Национальный исследовательский университет)**

Институт: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

**Лабораторная работа № 4-5
по курсу «Компьютерная графика»**

Студент:	Обыденкова Ю. Ю.
Группа:	М8О-308Б-18
Вариант:	2
Преподаватель:	Филиппов Г.С.
Оценка:	
Дата:	

Москва, 2020

Ознакомление с технологией OpenGL 2.1

Постановка задачи

Создать графическое приложение с использованием OpenGL. Изобразить прямой эллиптический конус с использованием средств OpenGL 2.1. Использовать буфер вершин. Точность аппроксимации тела задается пользователем. Обеспечить возможность вращения и масштабирования многогранника и удаление невидимых линий и поверхностей. Реализовать простую модель освещения на GLSL.

Параметры освещения задаются пользователем в диалоговом режиме.

Вариант задания: 2. Прямой эллиптический конус

Общие сведения о программе

Язык программирования: Python

Библиотеки: numpy, OpenGL.GL, OpenGL.GLU, OpenGL.GLUT, sys

program - пустой объект шейдерной программы

glAttachShader(program, vertex) - присоединяем вершинный шейдер к программе.

glLinkProgram(program) - собираем шейдерную программу

glRotate - вращения объекта вдоль оси

gluOrtho2D - определяем границы рисования

glClearColor - серый цвет для первоначальной закраски

glutSwapBuffers() - выводим все нарисованное в памяти на экран

glEnable(GL_CULL_FACE) - отрисовываем только видимую часть грани

init - функция, которая вычисляет вершины прямоугольников, в котором

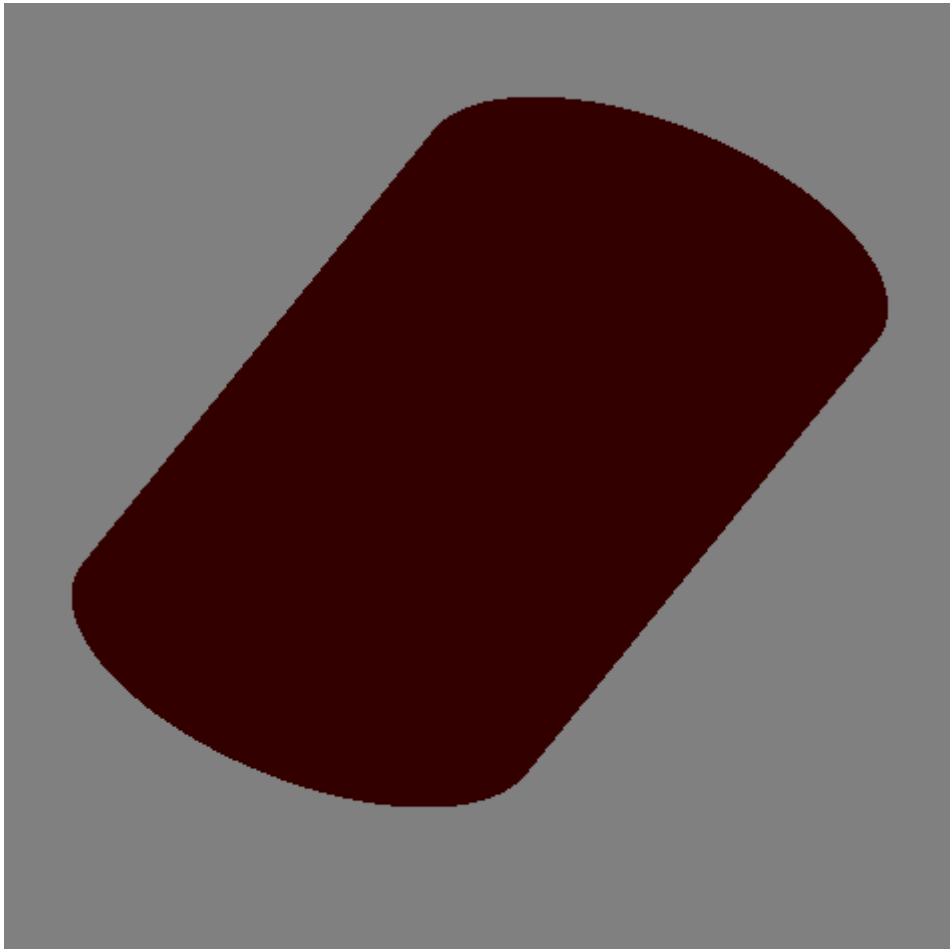
vertexes[i] - вершины нижнего эллипса, а vertexes[i + n] - вершины верхнего эллипса.

init_shaders - создание шейдеров

Руководство по использованию программы

```
~:$ python3 "4-5.py"
```

В начале программа предлагает ввести точность (кол-во вершин). Затем вводим параметры освещений: фонового и направленного. После ввода значений, программа выдаёт построенный график.



Код программы

```
# Импортируем все необходимые библиотеки:
import numpy as np
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import sys

vertexes_count = int(input("Точность: "))
ambient = np.array([float(x) for x in input('Ambient: ').split()])
diffuse = np.array([float(x) for x in input('Diffuse: ').split()])
```

```
vertexes = None
```

```
colors = None
```

```
indexes = None
```

```
indexes_bot = None
```

```
indexes_top = None
```

```
normals = None
```

```
height = 0.75
```

```
uniforms = [b"ambient", b"diffuse"]
```

```
locations = dict()
```

```
def init_shaders():
```

```
    global uniforms
```

```
    global locations
```

```
    global ambient
```

```
    global diffuse
```

```
vertex = create_shader(GL_VERTEX_SHADER, """
```

```
    varying vec4 vertex_color;
```

```
    uniform vec3 ambient;
```

```
    uniform vec3 diffuse;
```

```
    attribute vec3 a_pos;
```

```
    attribute vec3 a_normal;
```

```
    void main(){
```

```
        gl_Position = gl_ModelViewProjectionMatrix * vec4(a_pos, 1.0);
```

```
        vec3 light_dir = normalize(vec3(0.0, 0.0, -1.0));
```

```
        vec3 normal = normalize(vec3(gl_ModelViewMatrix * vec4(a_normal,  
1.0)));
```

```
        float intensity = max(0.8 * dot(normal, light_dir), 0.0);
```

```
        vertex_color = vec4(intensity * diffuse + 0.2 * ambient, 1.0);
```

```
    }""")
```

```
fragment = create_shader(GL_FRAGMENT_SHADER, """
```

```

    varying vec4 vertex_color;
    void main() {
        gl_FragColor = vertex_color;
    }""")
# Создаем пустой объект шейдерной программы
program = glCreateProgram()
# Присоединяем вершинный шейдер к программе
glAttachShader(program, vertex)
# Присоединяем фрагментный шейдер к программе
glAttachShader(program, fragment)
# "Собираем" шейдерную программу
glLinkProgram(program)

for uniform in uniforms:
    locations[uniform] = glGetUniformLocation(program, uniform)

glUseProgram(program)

glUniform3f(locations[b"ambient"], 1.0, 0.0, 0.0)
glUniform3f(locations[b"ambient"], 1.0, 0.0, 0.0)

glEnableVertexAttribArray(0)
glBindAttribLocation(program, 0, b"a_pos")

glEnableVertexAttribArray(1)
glBindAttribLocation(program, 1, b"a_normal")

# Процедура подготовки шейдера (тип шейдера, текст шейдера)
def create_shader(shader_type, source):
    # Создаем пустой объект шейдера

```

```
shader = glCreateShader(shader_type)
# Привязываем текст шейдера к пустому объекту шейдера
glShaderSource(shader, source)
# Компилируем шейдер
glCompileShader(shader)
# Возвращаем созданный шейдер
return shader
```

```
def calculate_coordinates(n=10, a=0.25, b=0.2):
    t = np.linspace(0, 2 * np.pi, n)
    x = a * np.cos(t)
    y = b * np.sin(t)
    return x, y
```

```
# Процедура инициализации
```

```
def init():
```

```
    init_shaders()
```

```
    global vertexes_count
```

```
    global vertexes
```

```
    global colors
```

```
    global indexes
```

```
    global indexes_top
```

```
    global indexes_bot
```

```
    global normals
```

```
    n = vertexes_count
```

```
    x, y = calculate_coordinates(n)
```

```
vertexes = np.empty(shape=(2 * n, 3), dtype=GLfloat)
indexes = np.empty(shape=(n, 4), dtype=GLushort)
indexes_top = np.empty(shape=n, dtype=GLushort)
indexes_bot = np.empty(shape=n, dtype=GLushort)
```

```
for i in range(n):
    vertexes[i] = y[i], 0.0, x[i]
    vertexes[i + n] = y[i], height, x[i]
```

```
for i in range(n):
    indexes[i] = i, (i + 1) % n, (i + 1) % n + n, i + n
```

```
for i in range(n):
    indexes_bot[i] = (n - 1) - i
    indexes_top[i] = i + n
```

```
normals = np.empty(shape=(2 * n, 3), dtype=GLfloat)
```

```
for i in range(n):
    vec1 = vertexes[(i - 1) % n] - vertexes[i]
    vec2 = vertexes[(i + 1) % n] - vertexes[i]
    vec3 = vertexes[i + n] - vertexes[i]
    norm_to_side_1 = np.cross(vec1, vec3)
    norm_to_side_2 = np.cross(vec3, vec2)
    norm_to_side_3 = 10 * np.cross(vec2, vec1)

    normals[i] = (norm_to_side_1 + norm_to_side_2 + norm_to_side_3)
    normals[i] /= np.linalg.norm(normals[i])

    normals[i + n] = (norm_to_side_1 + norm_to_side_2 - norm_to_side_3)
    normals[i + n] /= np.linalg.norm(normals[i + n])
```

```
glEnable(GL_CULL_FACE)
glCullFace(GL_FRONT)
glClearColor(0.5, 0.5, 0.5, 1.0)      # Серый цвет для первоначальной
защитки
gluOrtho2D(-1.0, 1.0, -1.0, 1.0)      # Определяем границы рисования
по горизонтали и вертикали
```

```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_TRUE, 0, vertexes)
glVertexAttribPointer(1, 3, GL_FLOAT, GL_TRUE, 0, normals)
```

```
# Процедура обработки специальных клавиш
```

```
def specialkeys(key, x, y):
```

```
    # Обработчики для клавиш со стрелками
```

```
    if key == GLUT_KEY_UP:      # Клавиша вверх
```

```
        glRotate(+1.0, 1, 0, 0) # Уменьшаем угол вращения по оси X
```

```
    if key == GLUT_KEY_DOWN:    # Клавиша вниз
```

```
        glRotate(-1.0, 1, 0, 0) # Увеличиваем угол вращения по оси X
```

```
    if key == GLUT_KEY_LEFT:    # Клавиша влево
```

```
        glRotate(-1.0, 0, 1, 0) # Уменьшаем угол вращения по оси Y
```

```
    if key == GLUT_KEY_RIGHT:   # Клавиша вправо
```

```
        glRotate(+1.0, 0, 1, 0) # Увеличиваем угол вращения по оси Y
```

```
glutPostRedisplay()      # Вызываем процедуру перерисовки
```

```
# Процедура перерисовки
```

```
def draw():
```

```
    global vertexes_count
```

```
    global indexes
```



```
global indexes_top  
global indexes_bot  
global ambient  
global diffuse
```

```
glClear(GL_COLOR_BUFFER_BIT) # Очищаем экран  
и заливаем серым цветом
```

```
glDrawElements(GL_QUADS, 4 * vertexes_count,  
GL_UNSIGNED_SHORT, indexes)
```

```
glDrawElements(GL_POLYGON, vertexes_count,  
GL_UNSIGNED_SHORT, indexes_top)
```

```
glDrawElements(GL_POLYGON, vertexes_count,  
GL_UNSIGNED_SHORT, indexes_bot)
```

```
glutSwapBuffers() # Выводим все нарисованное  
в памяти на экран
```

```
def main():
```

```
    # Здесь начинается выполнение программы
```

```
    # Использовать двойную буферизацию и цвета в формате RGB  
(Красный, Зеленый, Синий)
```

```
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)
```

```
    # Указываем начальный размер окна (ширина, высота)
```

```
    glutInitWindowSize(1200, 800)
```

```
    # Указываем начальное положение окна относительно левого верхнего  
    угла экрана
```

```
    glutInitWindowPosition(50, 50)
```

```
    # Инициализация OpenGL
```

```
    glutInit(sys.argv)
```

```
    # Создаем окно с заголовком
```

```
    glutCreateWindow("Laboratory Work #4-5")
```

```
    # Определяем процедуру, отвечающую за перерисовку
```

```
    glutDisplayFunc(draw)
```

```
# Определяем процедуру, отвечающую за обработку клавиш
glutSpecialFunc(specialkeys)
init()
# Запускаем основной цикл
glutMainLoop()
```

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

Вывод

Научилась создавать графическое приложение с использованием OpenGL и изображать прямой эллиптический конус с использованием средств OpenGL 2.1.