Московский авиационный институт
(Национальный исследовательский университет)

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

Курсовая работа по курсу “Компьютерная графика”

Студент: Ильиных Вадим Максимович
Группа: 80-301
Преподаватель: Чернышов Л.Н.
Дата:
Оценка:

Москва, 2021

1. Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

Вариант 13:
Поверхность вращения. Образующая – кривая Безье 3D 2-й степени

2. Описание программы

Программа написана на ЯП Python и состоит из двух файлов - **main.py** и **widgets.py** с использованием библиотек pygame, PyOpengl, tkinter, transformations**.** В начале пользователю предлагается ввести название файла с координатами вектора, вокруг которого будет вращаться кривая, и координаты точек, из которых впоследствии будет получена кривая Безье.

Затем открывается окно pygame с полученной поверхностью вращения. В окне управления можно вращать источник света, менять точность полученной поверхности. Также реализовано построение поверхности вращения в режиме реального времени. При нажатии на “включить изменение кривой” поверхность, построенная из координат файла, будет сброшена и пользователь может вручную настроить координаты вектора вращения и координаты точек.

Основные методы:

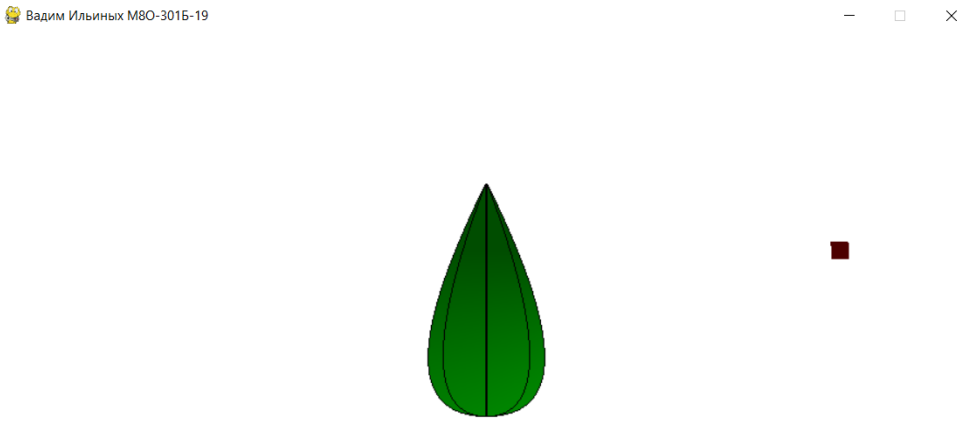
- bezier_curve(points, nTimes) - возвращает массивы x, y, z точек полученной кривой Безье. Points - двумерный массив, хранящий исходные точки, nTimes - точность полученной кривой Безье.
- calculate_points() - вычисляет точки поверхности вращения с указанной точностью.
- init_light() - инициализирование и задание параметров света.

3. Набор тестов

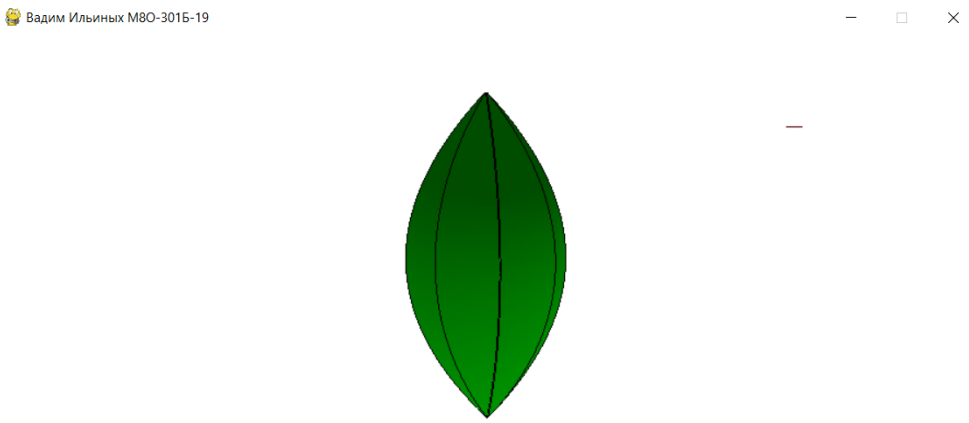
test1.txt	**test2.txt**
0 (координата x вектора вращения) 0 (координата y вектора вращения) 1 (координата z вектора вращения) 0 (координата x первой точки) 0 (координата y первой точки) 0 (координата z первой точки) 5 (координата x второй точки) 0 (координата y второй точки) 0 (координата z второй точки) 0 (координата x третьей точки) 0 (координата y третьей точки) 10 (координата z третьей точки)	0 (координата x вектора вращения) 0 (координата y вектора вращения) 1 (координата z вектора вращения) 0 (координата x первой точки) 0 (координата y первой точки) 0 (координата z первой точки) 5 (координата x второй точки) 0 (координата y второй точки) 5 (координата x второй точки) 0 (координата x третьей точки) 0 (координата y третьей точки) 10 (координата z третьей точки)

4. Результат выполнения тестов

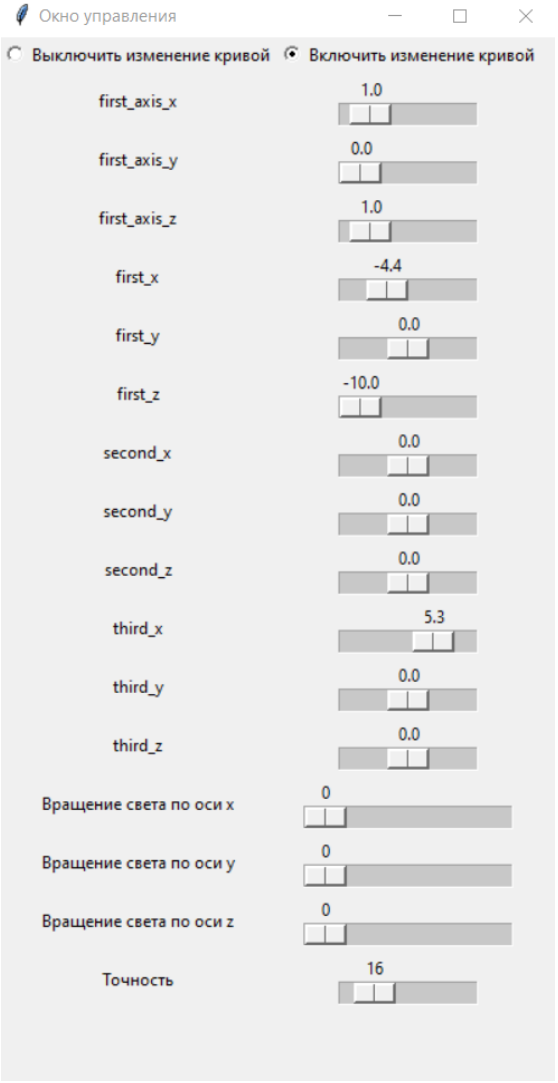
1) text1.txt:

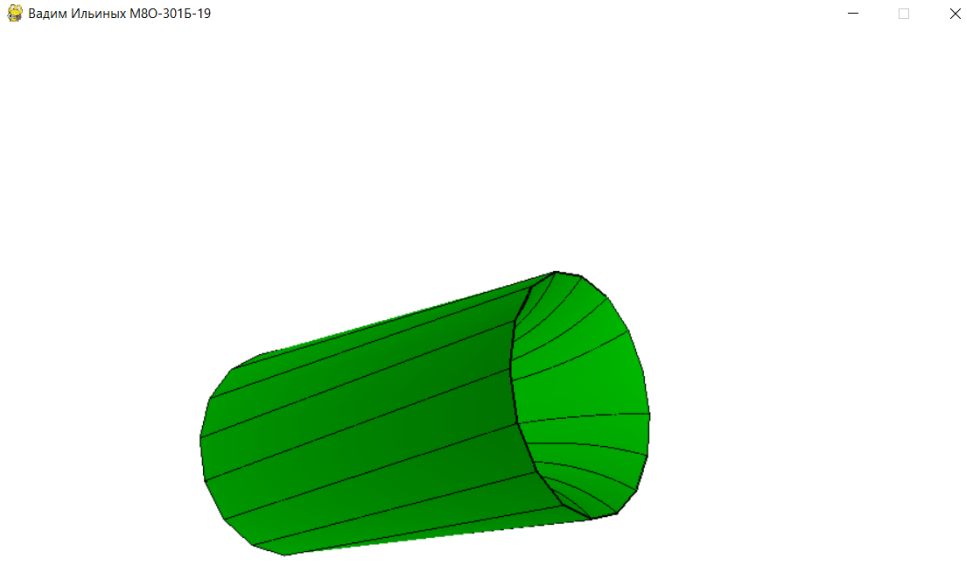


2) test2.txt:



3) вручную заданная кривая с параметрами, указанными ниже





5. Листинг программы

main.py

```
# Ильиных Вадим М8О-301Б-19
# Составить и отладить программу, обеспечивающую каркасную визуализацию
порции поверхности заданного типа.
# Исходные данные готовятся самостоятельно и вводятся из файла или в панели
ввода данных. Должна быть обеспечена
# возможность тестирования программы на различных наборах исходных данных.
Программа должна обеспечивать выполнение
# аффинных преобразований для заданной порции поверхности, а также
возможность управлять количеством изображаемых
# параметрических линий. Для визуализации параметрических линий поверхности
разрешается использовать только функции
# отрисовки отрезков в экранных координатах.
# Вариант 13:
# Поверхность вращения. Образующая – кривая Безье 3D 2-й степени

import numpy as np
import pygame
import threading
from transformations import *
from math import *
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
from scipy.special import comb
from widgets import *

vertex = (-1, -1, 1, 1, -1, 1, 1, 1, 1, -1, 1, 1)


def vec_len(x, y, z):
    return sqrt(x*x + y*y + z*z)
```

```
with open(str(input("Введите название файла: "))) as file:
    ar2x = float(file.readline())
    ar2y = float(file.readline())
    ar2z = float(file.readline())

    bc1x = float(file.readline())
    bc1y = float(file.readline())
    bc1z = float(file.readline())

    bc2x = float(file.readline())
    bc2y = float(file.readline())
    bc2z = float(file.readline())

    bc3x = float(file.readline())
    bc3y = float(file.readline())
    bc3z = float(file.readline())


def bernstein_poly(i, n, t):
    return comb(n, i) * (t ** i) * (1 - t) ** (n - i)


def bezier_curve(points, nTimes):
    nPoints = len(points)
    xPoints = np.array([p[0] for p in points])
    yPoints = np.array([p[1] for p in points])
    zPoints = np.array([p[2] for p in points])

    t = np.linspace(0.0, 1.0, nTimes)

    polynomial_array = np.array(
        [bernstein_poly(i, nPoints - 1, t) for i in range(0, nPoints)])

    # вычисляет скалярное произведение двух массивов.
    xvals = np.dot(xPoints, polynomial_array)
    yvals = np.dot(yPoints, polynomial_array)
    zvals = np.dot(zPoints, polynomial_array)

    return xvals, yvals, zvals


def calculate_points():
    check = return_light()
    accuracy = return_accuracy()
    global ar2x, ar2y, ar2z
    global bc1x, bc1y, bc1z
    global bc2x, bc2y, bc2z
    global bc3x, bc3y, bc3z
    if check:
        bc1x = return_first_x(); bc1y = return_first_y(); bc1z =
return_first_z()
        bc2x = return_second_x(); bc2y = return_second_y(); bc2z =
return_second_z()
        bc3x = return_third_x(); bc3y = return_third_y(); bc3z =
return_third_z()

    points = [[bc1x, bc1y, bc1z], [bc2x, bc2y, bc2z], [bc3x, bc3y, bc3z]]

    xvals, yvals, zvals = bezier_curve(points, accuracy)

    if check:
```

```
        ar2x = return_first_axis_x(); ar2y = return_first_axis_y(); ar2z =
return_first_axis_z()

    length = vec_len(ar2x, ar2y, ar2z)
    if length == 0:
        length = 1
        ar2z = 1

    p2 = [ar2x / length, ar2y / length, ar2z / length]

    num_curves = return_parts()
    radiane = 2*pi / num_curves
    angle = radiane

    xtvals = xvals
    ytvals = yvals
    ztvals = zvals

    while angle <= 2 * pi + 10**-6:
        m1 = rotation_matrix(angle, p2, points[0])
        m2 = rotation_matrix(angle, p2, points[1])
        m3 = rotation_matrix(angle, p2, points[2])

        pp1 = np.dot(points[0], m1[:3,:3].T)
        pp2 = np.dot(points[1], m2[:3,:3].T)
        pp3 = np.dot(points[2], m3[:3,:3].T)

        npoints = [pp1, pp2, pp3]
        xnvals, ynvals, znvals = bezier_curve(npoints, accuracy)
        xtvals = np.append(xtvals, xnvals)
        ytvals = np.append(ytvals, ynvals)
        ztvals = np.append(ztvals, znvals)
        angle += radiane

    vert = []
    vert_lines = []
    vert_gran = []

    for i in range(0, num_curves):
        for j in range(0, 2):
            vert_gran.append(xtvals[i*accuracy + j*accuracy])
            vert_gran.append(ytvals[i*accuracy + j*accuracy])
            vert_gran.append(ztvals[i*accuracy + j*accuracy])

    for i in range(0, num_curves):
        for j in range(0, 2):
            vert_gran.append(xtvals[i*accuracy + j*accuracy + accuracy - 1])
            vert_gran.append(ytvals[i*accuracy + j*accuracy + accuracy - 1])
            vert_gran.append(ztvals[i*accuracy + j*accuracy + accuracy - 1])

    for i in range(0, len(xtvals)):
        vert_lines.append(xtvals[i])
        vert_lines.append(ytvals[i])
        vert_lines.append(ztvals[i])

    for l in range(0, num_curves):
        for i in range(0, accuracy):
            for j in range(0, 2):
                vert.append(xtvals[accuracy*l + i + accuracy*j])
                vert.append(ytvals[accuracy*l + i + accuracy*j])
                vert.append(ztvals[accuracy*l + i + accuracy*j])
```

```
    for i in range(0, accuracy):
        for j in range(0, 2):
            vert.append(xtvals[i + accuracy * j])
            vert.append(ytvals[i + accuracy * j])
            vert.append(ztvals[i + accuracy * j])

    return vert, vert_lines, vert_gran


def init_light():
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glEnable(GL_COLOR_MATERIAL)
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE)

    glPushMatrix()

    alpha = int(return_rotating_x())
    betta = int(return_rotating_y())
    gamma = int(return_rotating_z())
    intens = 0.3
    intens_diff = 1.0
    #
    glRotatef(alpha, 1, 0, 0)
    glRotatef(betta, 0, 1, 0)
    glRotatef(gamma, 0, 0, 1)

    k = 15

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, (intens, intens, intens, 1))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (intens_diff, intens_diff, intens_diff,
1))
    glLightfv(GL_LIGHT0, GL_POSITION, (0.8 * k, 0.3 * k, 0.6 * k, 1))

    glTranslatef(0.8 * k, 0.3 * k, 0.6 * k)

    glScalef(0.3, 0.3, 0.3)
    glColor3f(1, 0, 0)
    draw_light()
    glPopMatrix()


def draw_light():
    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 0, vertex)
    glDrawArrays(GL_TRIANGLE_FAN, 0, 4)
    glDisableClientState(GL_VERTEX_ARRAY)


if __name__ == "__main__":
    t1 = threading.Thread(target=make_window)
    t1.start()
    nPoints = 3

    pygame.init()
    (width, height) = (900, 700)
    screen = pygame.display.set_mode((width, height), OPENGL | DOUBLEBUF)
    pygame.display.set_caption('Вадим Ильиных М8О-301Б-19')
    gluPerspective(45, (width / height), 0.1, 50.0)
    glTranslatef(0.0, 0.0, -30)
```

```
glLineWidth(2)
glEnable(GL_DEPTH_TEST)

while True:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            quit()
        if event.type == pygame.MOUSEMOTION:
            pressed = pygame.mouse.get_pressed(3)
            if pressed[0]:
                glRotatef(2, event.rel[1], event.rel[0], 0)
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 4:
                glScalef(1.1, 1.1, 1.1)
            elif event.button == 5:
                glScalef(0.9, 0.9, 0.9)

    key = pygame.key.get_pressed()

    if key[pygame.K_LEFT]:
        glRotatef(1, 0, -1, 0)
    if key[pygame.K_RIGHT]:
        glRotatef(1, 0, 1, 0)
    if key[pygame.K_UP]:
        glRotatef(1, -1, 0, 0)
    if key[pygame.K_DOWN]:
        glRotatef(1, 1, 0, 0)
    if key[pygame.K_q]:
        glRotatef(1, 0, 0, -1)
    if key[pygame.K_e]:
        glRotatef(1, 0, 0, 1)
    if key[pygame.K_KP_PLUS] or key[pygame.K_PLUS]:
        glScalef(1.1, 1.1, 1.1)
    if key[pygame.K_MINUS] or key[pygame.K_KP_MINUS]:
        glScalef(0.9, 0.9, 0.9)
    if key[pygame.K_r]:
        glLoadIdentity()
        gluPerspective(45, (width / height), 0.1, 50.0)
        glTranslatef(0.0, 0.0, -40)
        glLineWidth(2)

    vert, vert_lines, vert_gran = calculate_points()

    glClearColor(1, 1, 1, 1)
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    init_light()

    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 0, vert)
    glColor(0,1,0)
    glDrawArrays(GL_TRIANGLE_STRIP, 0, len(vert)//3)
    glVertexPointer(3, GL_FLOAT, 0, vert_lines)
    glColor(0,0,0,1)
    glDrawArrays(GL_LINE_STRIP, 0, len(vert_lines)//3)
    glVertexPointer(3, GL_FLOAT, 0, vert_gran)
    glColor(0, 0, 0, 1)
    glDrawArrays(GL_LINES, 0, len(vert_gran) // 3)
    glDisableClientState(GL_VERTEX_ARRAY)
```

```
        glDisable(GL_LIGHT0)
        glDisable(GL_LIGHTING)
        glDisable(GL_COLOR_MATERIAL)

        pygame.display.flip()
        clock = pygame.time.Clock()
```

widgets.py

```
import tkinter as tk
from tkinter import *

first_axis_x = 0
first_axis_y = 0
first_axis_z = 0
first_x = 0
first_y = 0
accuracy = 120
first_z = 0
second_x = 0
second_y = 0
second_z = 0
third_x = 0
third_y = 0
third_z = 0
light_change = 0
rotating_x = 0
rotating_y = 0
rotating_z = 0
parts = 8


def return_parts():
    return int(parts)


def return_accuracy():
    return int(accuracy)


def return_light():
    return light_change


def return_first_axis_x():
    return float(first_axis_x)


def return_first_axis_y():
    return float(first_axis_y)


def return_first_axis_z():
    return float(first_axis_z)


def return_first_x():
    return float(first_x)
```

```
def return_first_y():
    return float(first_y)


def return_first_z():
    return float(first_z)


def return_second_x():
    return float(second_x)


def return_second_y():
    return float(second_y)


def return_second_z():
    return float(second_z)


def return_third_x():
    return float(third_x)


def return_third_y():
    return float(third_y)


def return_third_z():
    return float(third_z)


def return_rotating_x():
    return rotating_x


def return_rotating_y():
    return rotating_y


def return_rotating_z():
    return rotating_z


def change_accuracy(value):
    global accuracy
    accuracy = value


def change_first_axis_x(value):
    global first_axis_x
    first_axis_x = value


def change_first_axis_y(value):
    global first_axis_y
    first_axis_y = value


def change_first_axis_z(value):
    global first_axis_z
```

```
    first_axis_z = value


def change_first_x(value):
    global first_x
    first_x = value


def change_first_y(value):
    global first_y
    first_y = value


def change_first_z(value):
    global first_z
    first_z = value


def change_second_x(value):
    global second_x
    second_x = value


def change_second_y(value):
    global second_y
    second_y = value


def change_second_z(value):
    global second_z
    second_z = value


def change_third_x(value):
    global third_x
    third_x = value


def light_on():
    global light_change
    light_change = 1


def light_off():
    global light_change
    light_change = 0


def change_third_y(value):
    global third_y
    third_y = value


def change_third_z(value):
    global third_z
    third_z = value


def change_rotating_x(value):
    global rotating_x
    rotating_x = value
```

```
def change_rotating_y(value):
    global rotating_y
    rotating_y = value


def change_rotating_z(value):
    global rotating_z
    rotating_z = value


def change_parts(value):
    global parts
    parts = value


def make_window():
    window = tk.Tk()
    window.title("Окно управления")
    window.geometry('400x750')
    count = 0
    var1 = IntVar()

    rad1 = Radiobutton(window, text="Выключить изменение кривой", value=1,
variable=var1, command=light_off)
    rad2 = Radiobutton(window, text="Включить изменение кривой", value=2,
variable=var1, command=light_on)
    rad1.grid(row=count, column=0)
    rad2.grid(row=count, column=1)
    count += 1

    scale_big_axis = tk.Scale(window, from_=0, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_axis_x)
    label_big_axis = tk.Label(window, text="first_axis_x")
    label_big_axis.grid(row=count, column=0)
    scale_big_axis.grid(row=count, column=1)
    count += 1

    scale_small_axis = tk.Scale(window, from_=0, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_axis_y)
    label_small_axis = tk.Label(window, text="first_axis_y")
    label_small_axis.grid(row=count, column=0)
    scale_small_axis.grid(row=count, column=1)
    count += 1

    scale_height = tk.Scale(window, from_=0, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_axis_z)
    label_height = tk.Label(window, text="first_axis_z")
    label_height.grid(row=count, column=0)
    scale_height.grid(row=count, column=1)
    count += 1

    scale_big_axis = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_x)
    label_big_axis = tk.Label(window, text="first_x")
    label_big_axis.grid(row=count, column=0)
    scale_big_axis.grid(row=count, column=1)
    count += 1
```

```
    scale_small_axis = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_y)
    label_small_axis = tk.Label(window, text="first_y")
    label_small_axis.grid(row=count, column=0)
    scale_small_axis.grid(row=count, column=1)
    count += 1

    scale_height = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_first_z)
    label_height = tk.Label(window, text="first_z")
    label_height.grid(row=count, column=0)
    scale_height.grid(row=count, column=1)
    count += 1

    scale_accuracy = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_second_x)
    label_accuracy = tk.Label(window, text="second_x")
    label_accuracy.grid(row=count, column=0)
    scale_accuracy.grid(row=count, column=1)
    count += 1

    scale_rotating_x = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_second_y)
    label_rotating_x = tk.Label(window, text="second_y")
    label_rotating_x.grid(row=count, column=0)
    scale_rotating_x.grid(row=count, column=1)
    count += 1

    scale_rotating_y = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_second_z)
    label_rotating_y = tk.Label(window, text="second_z")
    label_rotating_y.grid(row=count, column=0)
    scale_rotating_y.grid(row=count, column=1)
    count += 1

    scale_rotating_z = tk.Scale(window, from_=-10, to=10, resolution=0.1,
orient=HORIZONTAL, command=change_third_x)
    label_rotating_z = tk.Label(window, text="third_x")
    label_rotating_z.grid(row=count, column=0)
    scale_rotating_z.grid(row=count, column=1)
    count += 1

    scale_intensivity = tk.Scale(window, from_=-10, to=10, orient=HORIZONTAL,
resolution=0.1, command=change_third_y)
    label_intensivity = tk.Label(window, text="third_y")
    label_intensivity.grid(row=count, column=0)
    scale_intensivity.grid(row=count, column=1)
    count += 1

    scale_diffuse = tk.Scale(window, from_=-10, to=10, orient=HORIZONTAL,
resolution=0.1, command=change_third_z)
    label_diffuse = tk.Label(window, text="third_z")
    label_diffuse.grid(row=count, column=0)
    scale_diffuse.grid(row=count, column=1)
    count += 1

    scale_rotating_x = tk.Scale(window, from_=0, to=360, orient=HORIZONTAL,
length=150, command=change_rotating_x)
    label_rotating_x = tk.Label(window, text="Вращение света по оси x")
    label_rotating_x.grid(row=count, column=0)
    scale_rotating_x.grid(row=count, column=1)
```

```
    count += 1

    scale_rotating_y = tk.Scale(window, from_=0, to=360, orient=HORIZONTAL,
length=150, command=change_rotating_y)
    label_rotating_y = tk.Label(window, text="Вращение света по оси y")
    label_rotating_y.grid(row=count, column=0)
    scale_rotating_y.grid(row=count, column=1)
    count += 1

    scale_rotating_z = tk.Scale(window, from_=0, to=360, orient=HORIZONTAL,
length=150, command=change_rotating_z)
    label_rotating_z = tk.Label(window, text="Вращение света по оси z")
    label_rotating_z.grid(row=count, column=0)
    scale_rotating_z.grid(row=count, column=1)
    count += 1

    # scale_accuracy = tk.Scale(window, from_=120, to=1000,
orient=HORIZONTAL, length=150, command=change_accuracy)
    # label_accuracy = tk.Label(window, text="Точность")
    # label_accuracy.grid(row=count, column=0)
    # scale_accuracy.grid(row=count, column=1)
    # count += 1

    scale_parts = tk.Scale(window, from_=8, to=64, orient=HORIZONTAL,
command=change_parts)
    label_parts = tk.Label(window, text="Точность")
    label_parts.grid(row=count, column=0)
    scale_parts.grid(row=count, column=1)
    count += 1

    tk.mainloop()
```

6. Вывод

В ходе выполнения этой работы я познакомился с кривыми Безье 2-й степени, поверхностями вращения. Опыт, полученный при написании лабораторных работ в ходе курса “Компьютерная графика” существенно помог в написании курсового проекта.

При вращении кривой Безье вокруг некоторой оси, могут получаться красивые и порой забавные поверхности, что порой позволяло веселиться при написании данной работы.

Литература

1. Справочник по Python [Электронный ресурс]. URL: https://jenyay.net/Matplotlib/Widgets (дата обращения: 2.12.2021).
2. Справочник по поверхностям вращения [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B2%D0%B5%D1%80%D1%85%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B2%D1%80%D0%B0%D1%89%D0%B5%D0%BD%D0%B8%D1%8F
(дата обращения: 2.12.2021)
3. Справочник по матрицам поворота [Электронный ресурс]. URL: https://api-2d3d-cad.com/euler_angles_quaternions/
(дата обращения: 2.12.2021)