**Минобрнауки РФ**

**ЮЗГУ**

**ФФиПИ**

Кафедра программной инженерии

**ЛАБОРАТОРНАЯ РАБОТА №11**

**АЛГОРИТМЫ ПОСТРОЕНИЯ КРИВЫХ И ПОВЕРХНОСТЕЙ**

Выполнил: ст. гр. ПО-22б

Алешин К.Р.

Проверил: Ефремов В. В.

Курск, 2024 г

**Задание:** создать приложение, реализующее построение кривых и поверхностей методами Безье, Чайкина, Ду-Сабина.

Создать приложение, дающее возможность:

1) реализовывать построение квадратичной кривой Безье;

а) возможность задания произвольного количества вершин;

б) поэтапное выполнение с расчетом промежуточных вершин;

в) выбор параметра разбиения.

2) реализовывать построение кубической кривой Безье;

а) возможность задания произвольного количества вершин;

б) поэтапное выполнение с расчетом промежуточных вершин;

в) выбор параметра разбиения;

3) реализовывать построение кривой Чайкина;

4) реализовывать построение поверхности Безье:

а) возможность задания произвольного количества массива исходных вершин;

б) выбор параметра разбиения;

5) реализовывать построение поверхности Ду-Сабина:

а) возможность задания произвольного количества вершин;

б) поэтапное выполнение с расчетом точек грани;

6) реализовать возможность отображения:

а) точек исходного контрольного полигона;

б) вершин уточненного контрольного полигона;

в) выбора количества разбиений.

**Листинг программы:**import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

def calculate\_bezier\_curve(control\_points, num\_points=100):

"""Рассчитать точки кривой Безье для заданных контрольных точек."""

n = len(control\_points) - 1

t\_values = np.linspace(0, 1, num\_points)

curve\_points = np.zeros((num\_points, 2))

for i in range(num\_points):

t = t\_values[i]

point = np.zeros(2)

for k in range(n + 1):

bernstein = (math.comb(n, k) \* (t \*\* k) \* ((1 - t) \*\* (n - k)))

point += bernstein \* control\_points[k]

curve\_points[i] = point

return curve\_points

def calculate\_bezier\_surface(control\_grid, u\_steps=10, v\_steps=10):

"""Рассчитать точки поверхности Безье."""

rows, cols = control\_grid.shape[:2]

u\_values = np.linspace(0, 1, u\_steps)

v\_values = np.linspace(0, 1, v\_steps)

surface\_points = np.zeros((u\_steps, v\_steps, 3))

for i, u in enumerate(u\_values):

for j, v in enumerate(v\_values):

point = np.zeros(3)

for m in range(rows):

for n in range(cols):

bernstein\_u = (math.comb(rows - 1, m) \* (u \*\* m) \* ((1 - u) \*\* (rows - 1 - m)))

bernstein\_v = (math.comb(cols - 1, n) \* (v \*\* n) \* ((1 - v) \*\* (cols - 1 - n)))

point += bernstein\_u \* bernstein\_v \* control\_grid[m, n]

surface\_points[i, j] = point

return surface\_points

def plot\_bezier\_curve(control\_points, curve\_points):

"""Отобразить кривую Безье и её контрольные точки."""

plt.figure(figsize=(8, 6))

plt.plot(control\_points[:, 0], control\_points[:, 1], 'ro-', label="Контрольные точки")

plt.plot(curve\_points[:, 0], curve\_points[:, 1], 'b-', label="Кривая Безье")

plt.legend()

plt.title("Кубическая кривая Безье")

plt.grid()

plt.show()

def plot\_bezier\_surface(control\_grid, surface\_points):

"""Отобразить поверхность Безье и её контрольную сетку."""

fig = plt.figure(figsize=(10, 8))

ax = fig.add\_subplot(111, projection='3d')

# Отобразить контрольную сетку

for i in range(control\_grid.shape[0]):

ax.plot(control\_grid[i, :, 0], control\_grid[i, :, 1], control\_grid[i, :, 2], 'ro-')

for j in range(control\_grid.shape[1]):

ax.plot(control\_grid[:, j, 0], control\_grid[:, j, 1], control\_grid[:, j, 2], 'ro-')

# Отобразить поверхность

X = surface\_points[:, :, 0]

Y = surface\_points[:, :, 1]

Z = surface\_points[:, :, 2]

ax.plot\_surface(X, Y, Z, color='b', alpha=0.6)

ax.set\_title("Поверхность Безье")

plt.show()

def calculate\_doosabin\_surface(vertices, num\_iterations=1):

"""Рассчитать точки поверхности Ду-Сабина (схема упрощена)."""

faces = []

for i in range(len(vertices)):

v0 = vertices[i]

v1 = vertices[(i + 1) % len(vertices)]

faces.append((v0 + v1) / 2)

for \_ in range(num\_iterations):

new\_faces = []

for i in range(len(faces)):

v0 = faces[i]

v1 = faces[(i + 1) % len(faces)]

new\_faces.append((v0 + v1) / 2)

faces = new\_faces

return np.array(faces)

def plot\_doosabin\_surface(vertices, result):

"""Отобразить поверхность Ду-Сабина."""

plt.figure(figsize=(8, 6))

plt.plot(vertices[:, 0], vertices[:, 1], 'ro-', label="Исходные вершины")

plt.plot(result[:, 0], result[:, 1], 'b-', label="Поверхность Ду-Сабина")

plt.legend()

plt.title("Поверхность Ду-Сабина")

plt.grid()

plt.show()

def main():

while True:

print("\nВыберите действие:")

print("1. Построение кривой Безье")

print("2. Построение поверхности Безье")

print("3. Построение поверхности Ду-Сабина")

print("4. Выход")

choice = input("Введите номер действия: ")

if choice == "1":

n = int(input("Введите количество контрольных точек: "))

control\_points = []

for i in range(n):

x, y = map(float, input(f"Введите координаты вершины {i + 1} (x y): ").split())

control\_points.append((x, y))

control\_points = np.array(control\_points)

num\_points = int(input("Введите количество точек на кривой: "))

curve\_points = calculate\_bezier\_curve(control\_points, num\_points)

plot\_bezier\_curve(control\_points, curve\_points)

elif choice == "2":

rows = int(input("Введите количество строк контрольной сетки: "))

cols = int(input("Введите количество столбцов контрольной сетки: "))

control\_grid = []

for i in range(rows):

row = []

for j in range(cols):

x, y, z = map(float, input(f"Введите координаты вершины ({i}, {j}) (x y z): ").split())

row.append((x, y, z))

control\_grid.append(row)

control\_grid = np.array(control\_grid)

u\_steps = int(input("Введите количество шагов по U: "))

v\_steps = int(input("Введите количество шагов по V: "))

surface\_points = calculate\_bezier\_surface(control\_grid, u\_steps, v\_steps)

plot\_bezier\_surface(control\_grid, surface\_points)

elif choice == "3":

n = int(input("Введите количество вершин: "))

vertices = []

for i in range(n):

x, y = map(float, input(f"Введите координаты вершины {i + 1} (x y): ").split())

vertices.append((x, y))

vertices = np.array(vertices)

iterations = int(input("Введите количество итераций: "))

result = calculate\_doosabin\_surface(vertices, iterations)

plot\_doosabin\_surface(vertices, result)

elif choice == "4":

print("Выход из программы.")

break

else:

print("Неверный выбор. Попробуйте снова.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Пример работы программы:  
Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание**

**Изображение выглядит как текст, электроника, снимок экрана, программное обеспечение

Автоматически созданное описание**

**Изображение выглядит как снимок экрана, диаграмма, линия, Прямоугольник

Автоматически созданное описание**