ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
"МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)"

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

Курсовой проект по курсу
«Компьютерная графика»

Группа: М8о-307Б-18
Студент:
Тояков Артем Олегович
Преподаватель:
Филиппов Глеб Сергеевич
Оценка:
Дата:
Москва, 2021

Оглавление

1.Постановка задачи

Задание: 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.Структура программы

1. cp.py

3.Описание программы

ЯП: Python
ОС: Ubuntu 18.04

Библиотеки: numpy, matplotlib.pyplot, mpl_toolkits.mplot3d, math

matplotlib.pyplot.plot(*args, **kwargs) - Создает график

matplotlib.pyplot.show() - Отображает окно с графиком

numpy.arange(arg1,arg2) - Создания последовательностей чисел

Ход работы:

- Создаем кривую Безье (на основе полиномов Бернштейна). Вычисляем значения координат кривой с высокой точностью(1000 измерений).
- Задаем 3 точки для построения первой кривой. С помощью преобразований вращаем эти три точки на заданный угол относительно заданной прямой.
- Отрисовываем получившиеся после преобразования кривые, получая поверхность вращения.

4.Листинг программы

```
import matplotlib as mpl

import numpy as np

from scipy.special import comb

from matplotlib import pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import math

from matplotlib.ticker import MaxNLocator

import pylab

from mayavi import mlab

from matplotlib import cm

from numpy.random import randn

from scipy import array, newaxis



global ar1x, ar1y, ar1z, ar2x, ar2y, ar2z

global bc1x, bc1y, bc1z, bc2x, bc2y, bc2z, bc3x, bc3y, bc3z

### minvalue 0

# 1-2 точки оси вращения

ar1x = int(input())

ar1y = int(input())

ar1z = int(input())


ar2x =int(input())
```

```
ar2y = int(input())

ar2z =int(input())


#1-3 точки кривой Безье

bc1x =int(input())

bc1y =int(input())

bc1z =int(input())


bc2x = int(input())

bc2y = int(input())

bc2z = int(input())


bc3x= int(input())

bc3y= int(input())

bc3z= int(input())




def bernstein_poly(i, n, t):

    return comb(n, i) * (t**(n - i)) * (1 - t)**i




def bezier_curve(points, nTimes=1000):

    nPoints = len(points)

    xPoints = np.array([p[0] for p in points])

    yPoints = np.array([p[1] for p in points])
```

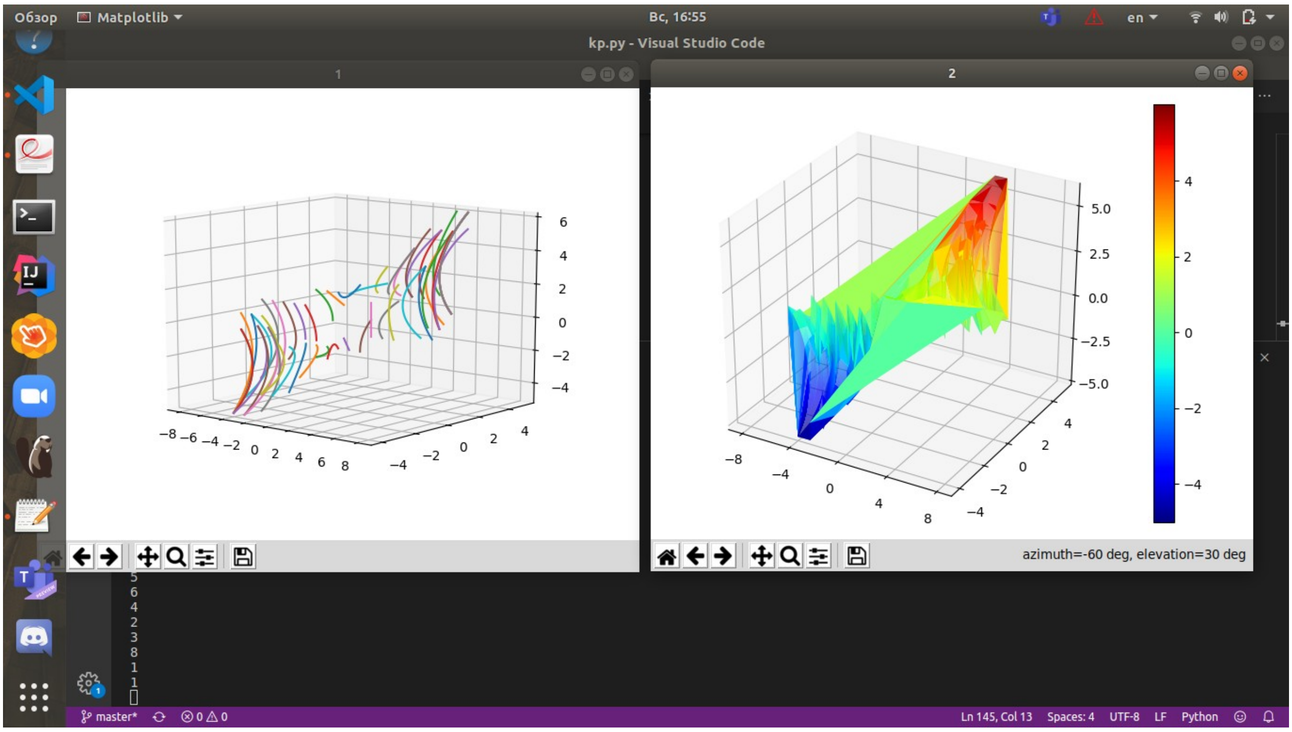
```
    zPoints = np.array([p[2] for p in points])

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

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

    xvals = np.dot(xPoints, polynomial_array)
    yvals = np.dot(yPoints, polynomial_array)
    zvals = np.dot(zPoints, polynomial_array)

    return xvals, yvals, zvals


from math import pi ,sin, cos

def R(theta, u):
    return [[cos(theta) + u[0]**2 * (1-cos(theta)),
             u[0] * u[1] * (1-cos(theta)) - u[2] * sin(theta),
             u[0] * u[2] * (1 - cos(theta)) + u[1] * sin(theta)],
            [u[0] * u[1] * (1-cos(theta)) + u[2] * sin(theta),
             cos(theta) + u[1]**2 * (1-cos(theta)),
             u[1] * u[2] * (1 - cos(theta)) - u[0] * sin(theta)],
            [u[0] * u[2] * (1-cos(theta)) - u[1] * sin(theta),
             u[1] * u[2] * (1-cos(theta)) + u[0] * sin(theta),
```

```
        cos(theta) + u[2]**2 * (1-cos(theta))]]


def Rotate(pointToRotate, point1, point2, theta):
    u= []
    squaredSum = 0
    for i,f in zip(point1, point2):
        u.append(f-i)
        squaredSum += (f-i) **2

    u = [i/squaredSum for i in u]

    r = R(theta, u)
    rotated = []

    for i in range(3):
        rotated.append(round(sum([r[j][i] * pointToRotate[j] for j in range(3)])))

    return rotated



if __name__ == "__main__":
    nPoints = 3
```

```
points = [[bc1x,bc1y,bc1z],[bc2x,bc2y,bc2z],[bc3x,bc3y,bc3z]]
xpoints = [p[0] for p in points]
ypoints = [p[1] for p in points]
zpoints = [p[2] for p in points]
xvals, yvals, zvals = bezier_curve(points, nTimes=1000)
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(xvals, yvals, zvals, label='bezier')
p1=[ar1x,ar1y,ar1z]
p2=[ar2x,ar2y,ar2z]

radiane = 0
angle = pi/12

xtvals=xvals
ytvals=yvals
ztvals=zvals

while angle <= 2*pi:
    pp1 = Rotate(points[0], p1, p2, angle)
    pp2 = Rotate(points[1], p1, p2, angle)
    pp3 = Rotate(points[2], p1, p2, angle)
    npoints=[pp1,pp2,pp3]
    xnvals, ynvals, znvals = bezier_curve(npoints, nTimes=1000)
    xtvals = np.append(xtvals, xnvals)
```

```
        ytvals = np.append(ytvals,ynvals)

        ztvals = np.append( ztvals , znvals )

        ax.plot(xnvals, ynvals, znvals, label='bezier')

        angle = angle + pi/24

    plt.gcf().canvas.set_window_title("1")

    fig = plt.figure()

    ax = fig.add_subplot(111, projection='3d')


    surf = ax.plot_trisurf(xtvals, ytvals, ztvals, cmap=cm.jet, linewidth=0)

    fig.colorbar(surf)


    ax.xaxis.set_major_locator(MaxNLocator(5))

    ax.yaxis.set_major_locator(MaxNLocator(6))

    ax.zaxis.set_major_locator(MaxNLocator(5))


    fig.tight_layout()

    plt.gcf().canvas.set_window_title("2")

    plt.show()




pylab.show()
```

5.Результат работы



6.Вывод

В ходе данной работы я познакомился с такими понятиями : полином Бернштейна, поверхность вращения и кривая безье. Также в ходе данной работы я освоил построение кривой Безье, используя полином Берштейна.

7.Список литературы

1. Описание кривой Безье. Алгоритм построения Url: https://ru.wikipedia.org/wiki/Кривая_Безье#Квадратичные_кривые