Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Курсовой проет

По курсу «Компьютерная графика»

«Каркасная визуализация заданной поверхности»

Студент: Красоткин С.А.
Группа: М80-308Б-19
Преподаватель: Филиппов Г.С.
Оценка

Москва, 2021

Постановка задачи

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа.

Вариант задачи

Вариант **10**. Кинематическая поверхность. Образующая – **кардиоида**, направляющая – **кубическая кривая Безье 3D**

Основные определения

- **Кинематическая поверхность**: совокупность положений некоторой линии, перемещающейся в пространстве по определённым правилам.
- **Образующая**: линия, движущаяся в пространстве по определённым правилам, вдоль какой-либо кривой.
- **Направляющая**: линия, вдоль которой двигается образующая кривой.
- **Кривая Безье**: кривая вид которой: $B(t) = \sum_{k=0}^{n} P_k b_{k,n}(t),\ 0 \le t \le 1$, где P_k– опорные точки, а $b_{k,n}$ – базисные функции кривой: $b_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$. Таким образом кубическая кривая Безье имеет вид:
  $B(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3)$
- **Кардиоида**: кривая, полученная при отслеживании положения заранее заданной точки на одной окружности, вращающейся по поверхности другой. В прямоугольный координатах её параметризация имеет вид: $x = 2a\,cos(t) - a\,cos\,2t,\ y = 2asint - sin2t.$

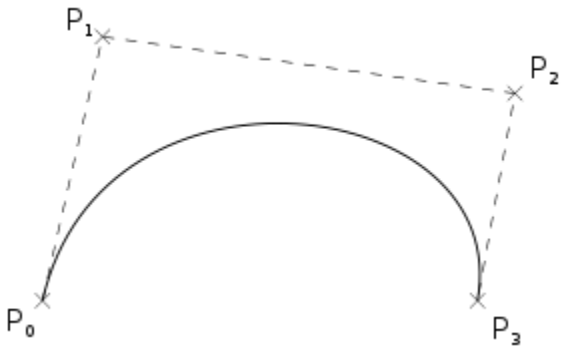

Рис. 1: Кубическая кривая Безье

Рис. 2: Плоская кардиода

Описание программы

Проект состоит из единственного файла 10.py. В нём считываются данными тремя способами: базисные, с консоли или файла. Потом находятся точки кубической кривой Безье, которая послужит направляющей и следом за этим считаются точки кардиоиды.

Для удобства приближения реализован обработчик события прокрутки колеса мыши.

Главный код лабораторной работы

```
def get_bezier(P1, P2, T1, T2, steps):    # interpolation of the Bezier
curve by the formula
    res = []
    for t in range(steps):
        s = t / steps
        h1 = (1 - s) ** 3
        h2 = 3 * s * (1 - s) ** 2
        h3 = 3 * s ** 2 * (1 - s)
        h4 = s ** 3
        res.append(h1 * P1 + h2 * P2 + h3 * T1 + h4 * T2)
    return res

x, y, z = zip(*curve)
e = 30
ell = []
for p in curve:                              # cardioda ( каждая точка кривой
служит центром, кардиоды задается параметрически)
    points = []
    for j in range(0, e + 1):
        tmp_x = 2*300*cos(j * 7 / e) - 300*cos(2*j * 7 / e) + p[0]
        tmp_y = p[1] * 2
        tmp_z = 2*300*sin(j * 7 / e) - 300*sin(2*j * 7 / e) + p[2]
```

```
            points.append((tmp_x, tmp_y, tmp_z))
        points = np.array(points)
        ell.append(points)

verts = []                                  # main array of vertices
for i in range(len(ell) - 1):
    for j in range(len(ell[i])):
        verts.append([
            ell[i][j], ell[(i + 1) % len(ell)][j],
            ell[(i + 1) % len(ell)][(j + 1) % len(ell[i])],
            ell[i][(j + 1) % len(ell[i])]
        ])
```

Демонстрация

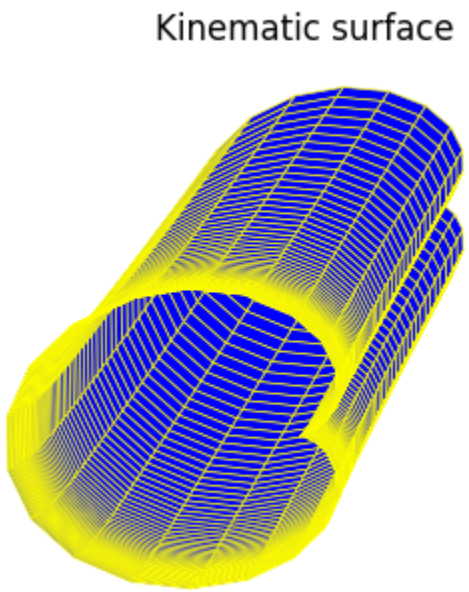

Рис. 3: Тест 1

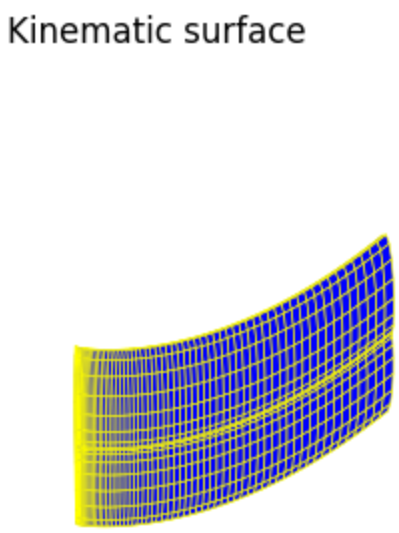

Рис. 4: Тест 2

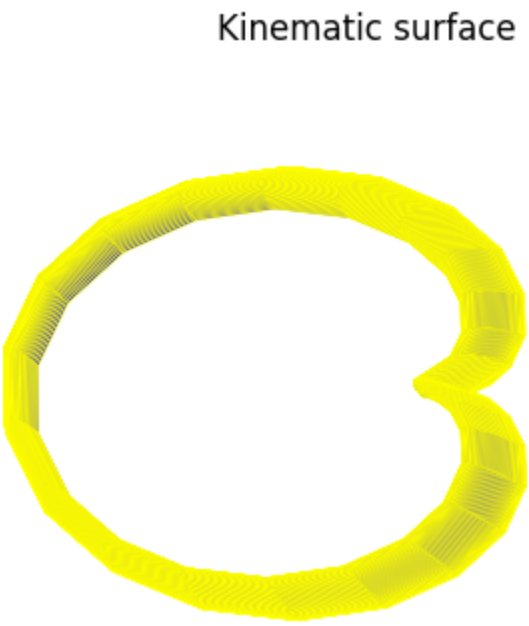

Рис. 5: Тест 3

Возможные улучшения

1. Можно добавить работу со светом для поверхности через OpenGL.
2. Стоит хранить точки в одном массиве, также в случае добавления света ещё для каждой точки вектор нормали.
3. Следует добавить слайдеры для регулировки опорных точек

4. Также можно добавить управление с клавиатуры.

Выводы

В итоге научился строить кинематическую поверхность с направляющей кубической кривой Безье и образующей кардиоидой. Замечу, что реализация достаточно адаптивная под другие направляющие и образующие. В частности можно заменить направляющую по квадратную кривую Безье и образующую астроиду.

В ходе выполнения узнал про многочлены Бернштейна, через которые можно сделать кривую Безье.