



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Московский государственный технический университет  
имени Н. Э. Баумана  
(национальный исследовательский университет)»  
(МГТУ им. Н. Э. Баумана)

---

ФАКУЛЬТЕТ «»

---

КАФЕДРА «»

---

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

*«Моделирование построения поверхностных и  
объемных геометрий с помощью операции движения»*

*2024 г.*

# СОДЕРЖАНИЕ

	с.
<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Кривые и поверхности</b>	<b>4</b>
1.1 Способы описания кривых и поверхностей	4
1.1.1 Явный вид	4
1.1.2 Неявный вид	4
1.1.3 Параметрический вид	4
1.2 Кривые и поверхности Безье	5
1.2.1 Кривые Безье	5
1.2.2 Поверхности Безье	6
1.3 Рациональные кривые и поверхности Безье	8
1.3.1 Рациональные кривые Безье	8
1.3.2 Рациональные поверхности Безье	10
1.4 В-сплайны	11
1.4.1 В-сплайн кривая	11
1.4.2 Свойства В-сплайна	15
1.4.3 Производные В-сплайн кривой	16
1.4.4 В-сплайн поверхности	17
1.5 NURBS	18
1.5.1 Рациональный В-сплайн и NURBS	18
1.5.2 Производная NURBS-кривой	20
1.5.3 NURBS-поверхности	22
<b>2 Поверхности движения</b>	<b>23</b>
2.1 Поверхность выдавливания	23
2.2 Поверхность вращения	24
2.3 Поверхность сдвига	25
2.4 Кинематические поверхности	26
<b>3 Программная реализация</b>	<b>29</b>
3.1 Графическая оболочка	29
3.2 Инструменты создания поверхностей движения	30
3.2.1 Поверхность выдавливания	30
3.2.2 Поверхность вращения	31
<b>ЗАКЛЮЧЕНИЕ</b>	<b>32</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>33</b>
<b>ПРИЛОЖЕНИЕ А Алгоритмы</b>	<b>34</b>

# ВВЕДЕНИЕ

Курсовая работа – вид учебной работы, направленный на развитие практических навыков и умений, а также формирование компетенций обучающихся в процессе выполнения определенных видов заданий, связанных с будущей профессиональной деятельностью. Работа рассчитана на закрепление и применение полученных навыков в процессе учёбы.

Целью данной курсовой работы является решение заданных задач. Задачами этой работы является (с полным текстом задания можно ознакомиться в ПРИЛОЖЕНИИ Б):

- разработать формулы и алгоритмы, описывающие геометрии различных тел движения
- получить визуализацию тел движения, реализовать возможность изменения их формы.

Для моделирование построения поверхностных и объемных геометрий необходимо использовать методы геометрического моделирования. Поэтому в курсовой работе будут рассмотрены основные методы построения поверхностей и кривых, а именно кривые и поверхности Безье, рациональные кривые и поверхности Безье, B-Spline и NURBS.

Затем будут рассмотрены методы построения поверхностей выдавливания, вращения, сдвига, а также кинематических поверхностей.

В конце будет рассмотрена программная реализация изложенных методов, поддерживающая визуализацию полученных поверхностей и кривых и их изменение в реальном времени.

# 1 Кривые и поверхности

## 1.1 Способы описания кривых и поверхностей

Существует три основных подхода к описанию кривых и поверхностей.

### 1.1.1 Явный вид

Для кривой:

$$y = f(x), z = g(x)$$

Для поверхности:

$$z = f(x, y)$$

Этот метод имеет несколько недостатков:

- Нельзя однозначно описать замкнутые кривые, например, окружности.
- Полученное описание не обладает инвариантностью относительно поворотов.
- При попытке задать кривые с очень большими углами наклона возникают большие вычислительные сложности.

### 1.1.2 Неявный вид

$$f(x, y, z) = 0$$

Недостатки:

- Кривая в трёхмерном пространстве задаётся как пересечение двух поверхностей, т.е. требуется решать систему алгебраических уравнений.
- Сложности в процессе объединения неявно заданных фрагментов кривых

### 1.1.3 Параметрический вид

Параметрическое задание кривой и поверхности преодолевает недостатки явного и неявного способов описания. С его помощью можно задавать многозначные кривые, т.е. такие зависимости, которые могут принимать несколько значений при одном значении аргумента.

Для кривой:

$$\begin{cases} x = x(u) \\ y = y(u) \\ a \leq u \leq b \end{cases} \quad (1.1)$$

и также будем пользоваться обозначением

$$\mathbf{C}(u) = (x(u), y(u)), \quad a \leq u \leq b \quad (1.2)$$

Для поверхности:

$$\begin{cases} x = x(u, v) \\ y = y(u, v) \\ z = z(u, v) \\ a \leq u \leq b \\ c \leq v \leq d \end{cases} \quad (1.3)$$

и также будем пользоваться обозначением

$$\mathbf{S}(u, v) = (x(u, v), y(u, v), z(u, v)), \quad a \leq u \leq b, \quad c \leq v \leq d \quad (1.4)$$

## 1.2 Кривые и поверхности Безье

### 1.2.1 Кривые Безье

Пусть заданы  $n + 1$  точек  $\mathbf{P}_i = (x_i, y_i, z_i)$ , называемых *контрольными точками*. Они определяют форму и пространственное положение кривой.

Тогда *кривую Безье  $n$ -ой степени* можно задать с помощью уравнения:

$$\mathbf{C}(u) = \sum_{i=0}^n B_{i,n}(u) \mathbf{P}_i, \quad 0 \leq u \leq 1 \quad (1.5)$$

где  $B_{i,n}$  - полиномы Бернштейна.

$$B_{i,n}(u) = C_n^i u^i (1-u)^{n-i} = \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \quad (1.6)$$

Для вычисления точек кривой Безье удобно использовать алгоритм де Кастельжо:

## Листинг 1.1 – Псевдокод алгоритма де Кастельжо

```

1 deCasteljau(P, n, u, C)
2 {
3     /*Вычисление точки на кривой Безье*/
4     /*[in]: P, n, u*/
5     /*[out]: C (точка)*/
6     for(i=0; i<=n; i++) /* Используем локальный массив, */
7         Q[i] = P[i]      /* чтобы не изменить исходный массив
8                             контрольных точек */
9     for(k=1; k<=n; k++)
10         for(i=0; i<=n-k; i++)
11             Q[i] = (1.0-u)*Q[i] + u*Q[i+1]
12     C = Q[0]
13 }
```

Например, на Рисунке 1.1 показана кривая Безье для контрольных точек  $P_1 = (0, 0)$ ,  $P_2 = (0, 1)$ ,  $P_3 = (1, 2)$ ,  $P_4 = (3, 0)$ .

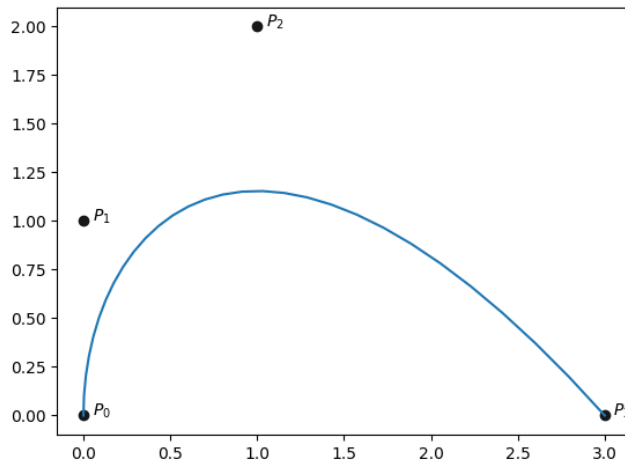


Рисунок 1.1 – Пример кривой Безье

### 1.2.2 Поверхности Безье

Пусть заданы контрольные точки  $P_{i,j}$ , где  $0 \leq i \leq n$  и  $0 \leq j \leq m$ . Тогда *поверхность Безье* можно задать с помощью следующего уравнения:

$$S(u, v) = \sum_{i=0}^n \sum_{j=0}^m B_{i,n}(u) B_{j,m}(v) P_{i,j}, \quad 0 \leq u, v \leq 1 \quad (1.7)$$

Аналогично кривым Безье, точки поверхности Безье можно находить с помощью алгоритма де Кастельжо из Листинга 1.1.

Листинг 1.2 – Псевдокод алгоритма де Кастельжо для поверхности

```
1 deCasteljauForSurface(P, n, m, u0, v0, S)
2 {
3     /*Вычисление точки на поверхности Безье*/
4     /*[in]: P, n, m, u0, v0*/
5     /*[out]: S (точка)*/
6     if (n <= m)
7     {
8         for(j=0; j<=m; j++) /* P[j][] - j-ая строка */
9             deCasteljau(P[j][], n, u0, Q[j]);
10        deCasteljau(Q, m, v0, S);
11    }
12    else
13    {
14        for(i=0; i<=n; i++)
15            deCasteljau(P[][i], m, v0, Q[i]);
16        deCasteljau(Q, n, u0, S);
17    }
18 }
```

На Рисунке 1.2 показан пример поверхности Безье для 15 контрольных точек.

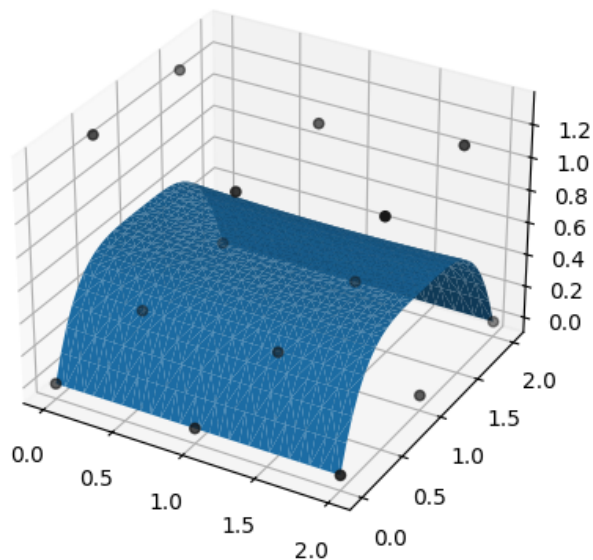


Рисунок 1.2 – Пример поверхности Безье

## 1.3 Рациональные кривые и поверхности Безье

### 1.3.1 Рациональные кривые Безье

Так как кривые Безье - полиномиальные кривые, они имеют существенный недостаток, а именно с их помощью невозможно задать некоторые виды кривых, такие как окружности, эллипсы, гиперболы и прочие. Данные виды кривых можно задать с помощью рациональных функций, то есть как частное двух полиномов.

$$x(u) = \frac{X(u)}{W(u)} \quad y(u) = \frac{Y(u)}{W(u)}, \quad (1.8)$$

где  $X(u)$ ,  $Y(u)$  и  $W(u)$  - полиномы.

Заметим также, что каждая координатная функция имеет одинаковый знаменатель  $W(u)$ .

Рациональные кривые с координатными функциями в виде (1.8) имеют элегантную геометрическую интерпретацию, которая дает эффективные методы построения этих кривых и небольшие требования к памяти компьютера.

Оказывается, что можно использовать однородные координаты, чтобы задать рациональные кривые в  $n$ -мерном пространстве с помощью полиномиальной кривой в  $(n + 1)$ -мерном пространстве.

Рассмотрим точку в евклидовом пространстве  $\mathbf{P} = (x, y, z)$ . Затем запишем точку  $\mathbf{P}$  как  $\mathbf{P}^\omega = (\omega x, \omega y, \omega z, \omega) = (X, Y, Z, W)$  в четырех-мерном пространстве, причем  $\omega \neq 0$ . Тогда  $\mathbf{P}$  можно получить из  $\mathbf{P}^\omega$  делением всех координат на четвертую координату  $W$ , то есть с помощью отображения  $P^\omega$  на гиперплоскость  $W = 1$

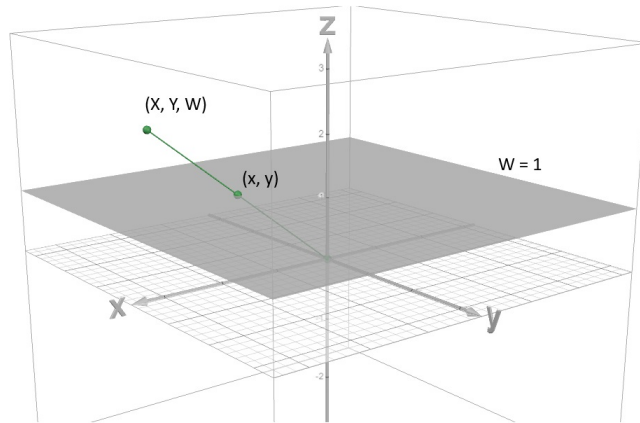


Рисунок 1.3 – Представление точки евклидова пространства в однородной форме для двумерного случая



Данное отображение  $H$  является перспективной проекцией с центром в начале координат:

$$\mathbf{P} = H\{\mathbf{P}^\omega\} = H\{(X, Y, Z, W)\} = \left(\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}\right) \quad (1.9)$$

Тогда для множества контрольных точек  $\{\mathbf{P}_i\}$  и множества весов  $\{\omega_i\}$  зададим множество взвешенных контрольных точек  $\mathbf{P}_i^\omega = (\omega_i x_i, \omega_i y_i, \omega_i z_i, \omega_i)$ . Тогда нерациональная (полиномиальная) кривая Безье в 4-х мерном пространстве

$$\mathbf{C}^\omega(u) = \sum_{i=0}^n B_{i,n}(u) \mathbf{P}_i^\omega \quad (1.10)$$

Уравнение (1.10) в координатном виде:

$$\begin{aligned} X(u) &= \sum_{i=0}^n B_{i,n}(u) \omega_i x_i & Y(u) &= \sum_{i=0}^n B_{i,n}(u) \omega_i y_i \\ Z(u) &= \sum_{i=0}^n B_{i,n}(u) \omega_i z_i & W(u) &= \sum_{i=0}^n B_{i,n}(u) \omega_i \end{aligned}$$

Заметим, что  $W \neq 0$  поскольку мы выбираем  $\omega_i > 0$ .

Применяя к (1.10) отображение (1.9), получим искомую рациональную кривую Безье в 3-х мерном пространстве, задающуюся формулами

$$\begin{aligned} x(u) &= \frac{X(u)}{W(u)} = \frac{\sum_{i=0}^n B_{i,n}(u) \omega_i x_i}{\sum_{i=0}^n B_{i,n}(u) \omega_i} \\ y(u) &= \frac{Y(u)}{W(u)} = \frac{\sum_{i=0}^n B_{i,n}(u) \omega_i y_i}{\sum_{i=0}^n B_{i,n}(u) \omega_i} \\ z(u) &= \frac{Z(u)}{W(u)} = \frac{\sum_{i=0}^n B_{i,n}(u) \omega_i z_i}{\sum_{i=0}^n B_{i,n}(u) \omega_i} \end{aligned}$$

или в векторной записи

$$\mathbf{C}(u) = \frac{\sum_{i=0}^n B_{i,n}(u)\omega_i \mathbf{P}_i}{\sum_{i=0}^n B_{i,n}(u)\omega_i} \quad (1.11)$$

Например, если взять  $\mathbf{P}_0 = (1, 0)$ ,  $\mathbf{P}_1 = (1, 1)$ ,  $\mathbf{P}_2 = (0, 1)$  и  $\omega_i = (1, 1, 2)$ , получим дугу окружности (Рисунок 1.4).

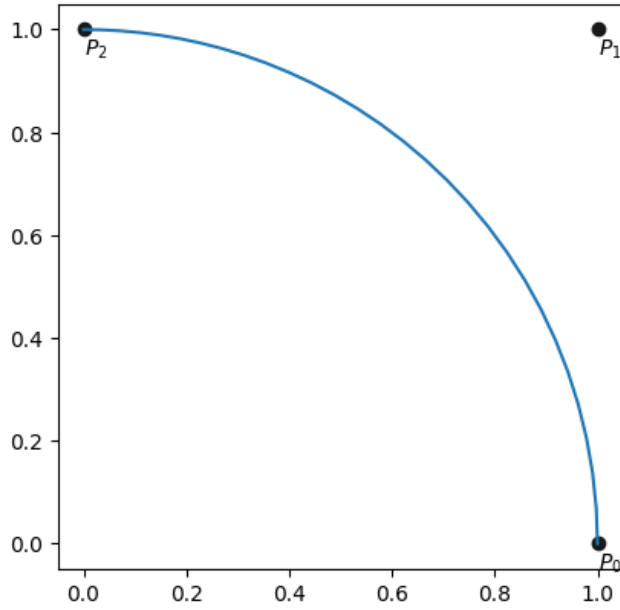


Рисунок 1.4 – Дуга окружности, построенная с помощью рациональной кривой Безье

Если веса всех вершин равны, то получим обычную кривую Безье, поскольку в таком случае знаменатель в уравнение (1.22) - это просто сумма полиномов Бернштейна, которая равна 1. Таким образом, рациональные кривые Безье являются обобщением полиномиальных кривых Безье.

### 1.3.2 Рациональные поверхности Безье

Аналогично рациональным кривым Безье, рациональные поверхности Безье можно представить как перспективную проекцию 4-х мерной полиномиальной поверхности Безье

$$\mathbf{S}^\omega(u, v) = \sum_{i=0}^n \sum_{j=0}^m B_{i,n}(u) B_{j,m}(v) \mathbf{P}_{i,j}^\omega$$

$$\begin{aligned} \mathbf{S}(u, v) = H\{\mathbf{S}^\omega(u, v)\} &= \frac{\sum_{i=0}^n \sum_{j=0}^m B_{i,n}(u) B_{j,m}(v) \omega_{i,j} \mathbf{P}_{i,j}}{\sum_{i=0}^n \sum_{j=0}^m B_{i,n}(u) B_{j,m}(v) \omega_{i,j}} = \\ &= \sum_{i=0}^n \sum_{j=0}^m R_{i,j}(u, v) \mathbf{P}_{i,j}, \quad (1.12) \end{aligned}$$

где

$$R_{i,j}(u, v) = \frac{B_{i,n}(u) B_{j,m}(v)}{\sum_{r=0}^n \sum_{s=0}^m B_{r,n}(u) B_{s,m}(v) \omega_{r,s}} \quad (1.13)$$

На Рисунке 1.5 изображена цилиндрическая поверхность, построенная с помощью рациональной поверхности Безье. Она представляет собой поверхность, полученную движением дуги окружности из Рисунка 1.4.

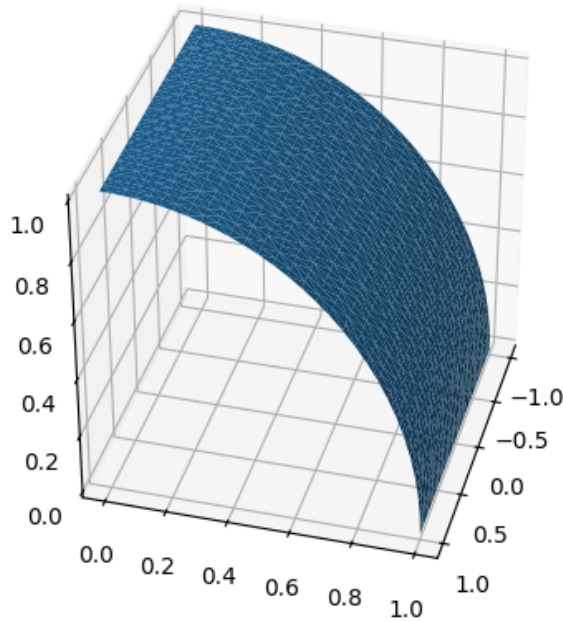


Рисунок 1.5 – Цилиндрическая поверхность, построенная с помощью рациональной поверхности Безье

## 1.4 В-сплайны

### 1.4.1 В-сплайн кривая

У кривых заданных полиномами или рациональными функциями есть несколько минусов.

— Для большого числа точек требуется полиномы большой степени. Так для того, чтобы построить кривую через  $n$  точек, требуется полином  $n - 1$  степени. Кривые, заданные полиномами с большими степенями, тяжело обрабатывать, а также они численно неустойчивы.

— Для сложных кривых также требуется большая степень полинома.

— Полиномиальные кривые не очень подходят для проектирования кривой. Хотя в кривых Безье и можно менять форму кривой, изменяя контрольные точки и значения весов в них, кривая меняется нелокально, т.е. изменение параметров одной точки меняет всю кривую.

В-сплайны лишены этих недостатков: степень полинома В-сплайна можно задать независимо от числа контрольных точек, а также они В-сплайны допускают локальный контроль над формой кривой.

Поставим задачу следующим образом. Пусть даны контрольные точки  $\mathbf{P}_i$ . Определим кривую по формуле

$$C(u) = \sum_{i=0}^n N_i(u) \mathbf{P}_i, \quad u_{min} \leq u \leq u_{max} \quad (1.14)$$

где  $N_i(u)$  - набор кусочно-полиномиальных функций, таких, что

1.  $N_i(u) = 0$  при  $u \notin [a_i, b_i] \subset [u_{min}, u_{max}]$ ;
2.  $N_i(u)$  линейно независимы и образуют базис;
3.  $\sum_{i=0}^n N_i(u) = 1$  для каждого  $u \in [u_{min}, u_{max}]$ .

Решение поставленной задачи даётся *В-сплайнами* (сокр. от basis).

Общее выражение для расчёта координат точек В-сплайна:

$$C(u) = \sum_{i=0}^n N_{i,p}(u) \mathbf{P}_i \quad (1.15)$$

В 1972 году Кокс и де Бур предложили использовать функции  $N_{i,p}$ , определяемые рекурсивно. Пусть  $U = \{u_0, \dots, u_m\}$  - неубывающая последовательно вещественных чисел, т.е.  $u_i \leq u_{i+1}, i = 0, \dots, m - 1$ .  $u_i$  называют

узлами (knot), а  $U$  - вектором узлов (knot vector). Тогда  $i$ -тая базисная функция В-сплайна  $p$ -ой степени, обозначаемая  $N_{i,p}(u)$ , выражается следующим образом:

$$N_{i,0}(u) = \begin{cases} 1, u \in [u_i, u_{i+1}] \\ 0, u \notin [u_i, u_{i+1}] \end{cases} \quad (1.16)$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (1.17)$$

В силу свойств базисных функций В-сплайна в любом заданном промежутке  $[u_i, u_{i+1}]$  могут быть отличны от нуля только  $p + 1$  функций:  $N_{i-p,p}, \dots, N_{i,p}$ . Например, единственные кубические функции, отличные от нуля на  $[u_3, u_4]$  - это функции  $N_0^3, \dots, N_3^3$ . Поэтому при вычислении базисных функций  $N^k$  в точке  $u$ , важно уметь находить индекс  $i$  в векторе узлов, при котором выполняется соотношение  $u_i \leq u \leq u_{i+1}$ . Для этого например можно использовать алгоритм бинарного поиска:

Листинг 1.3 – Алгоритм бинарного поиска индекса  $i$

```

1  int FindSpan(n, p, u, U)
2  {
3      /* Поиск i-го индекса в векторе узлов */
4      /*[in]: n, p, u, U*/
5      /*[out]: i*/
6      if(u == U[n+1]) return n; /*Специальный случай*/
7      low = p; high = n+1; /*Бинарный поиск*/
8      mid = (low+high)/2;
9      while(u < U[mid] || u >= U[mid+1])
10     {
11         if(u < U[mid])    high = mid;
12         else              low = mid;
13         mid = (low+high)/2;
14     }
15     return mid;
16 }
```

Также, по этой же причине следует, что чтобы находить значение сплайна для любого  $u$  из отрезка  $[u_i, u_{i+1}]$ , необходимо иметь не менее  $p$  дополнительных узлов до и после него. На практике этого обычно достигают, дублируя первый и последний узел нужное число раз. Например, если  $p = 3$  и даны узлы в точках  $\{0, 1, 2\}$ , то расширенный массив узлов будет иметь вид:

$\{0, 0, 0, 0, 1, 2, 2, 2, 2\}$ .

Для вычисления ненулевых базисных функций  $N_{i-p,p}, \dots, N_{i,p}$  можно использовать следующий алгоритм:

Листинг 1.4 – Алгоритм вычисления базисных функций  $N_{i-p,p}, \dots, N_{i,p}$

```
1 BasicFunc(i, u, p, U, N)
2 {
3     /*Вычисление ненулевых базисных функций*/
4     /*[in]: i, u, p, U*/
5     /*[out]: N*/
6     N[0] = 1.0;
7     for(j=1; j<=p; j++){
8         left[j] = u-U[i+1-j];
9         right[j] = U[i+j]-u;
10        saved = 0.0;
11        for(r=0; r<j; r++){
12            {
13                temp = N[r]/(right[r+1]+left[j-r]);
14                N[r] = saved+right[r+1]*temp;
15                saved = left[j-r]*temp;
16            }
17        N[j] = saved;
18    }
19 }
```

Также следует заметить, что существуют разные подходы к заданию узлового вектора. Разные методы задания узловых значений позволяют получить разные функции сопряжения и, соответственно, разные кривые. Если расстояние между значениями в узлах постоянно, получающаяся в результате кривая называется *равномерным В-сплайном*. Например, можно задать следующий равномерный вектор узлов:

$$\{-1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0\}$$

Часто значения узлов нормируются в диапазон от 0 до 1.

Аналогично, если допускается выбор одинаковых внутренних значений узлов и неравномерное размещение значений узлов, то такое В-сплайн называется *неравномерным*.

Так, например, для контрольных точек  $\mathbf{P}_0 = (0, 0)$ ,  $\mathbf{P}_1 = (0, 1)$ ,  $\mathbf{P}_2 = (1, 2)$ ,  $\mathbf{P}_3 = (3, 0)$  и узлового вектора  $U = \{0, 0, 0, \frac{1}{4}, 1, 1, 1\}$  В-сплайн 2-степени

будет иметь вид:

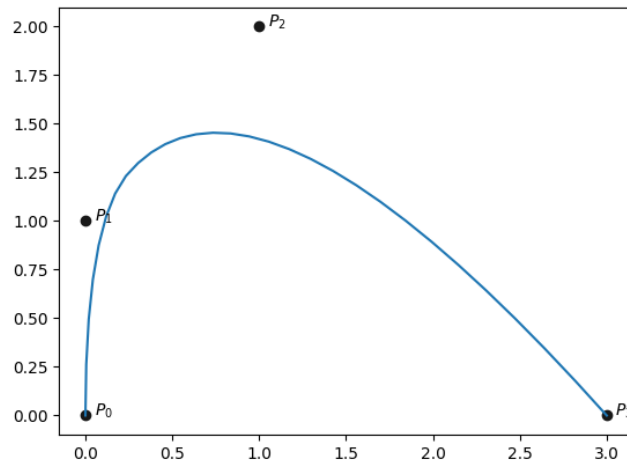


Рисунок 1.6 – Пример В-сплайн кривой

### 1.4.2 Свойства В-сплайна

В-сплайн имеет следующие свойства:

- полиномиальная кривая имеет степень  $p$  и непрерывность  $C^{p-1}$ ;
- диапазон параметра  $u$  делится на  $n + p + 1$  подынтервалов  $n + p + 2$  значениями, заданными в векторе узлов;
- если значения узлов обозначить  $\{u_0, u_1, \dots, u_{n+p+1}\}$ , получающийся В-сплайн определяется только в промежутке  $[u_p, u_{n+1})$ , т.к. только в этом промежутке  $\sum_{i=0}^n N_{i,p} = 1$ ;
- каждый участок сплайна определяется  $p + 1$  контрольными точками;
- локальная коррекция: любая контрольная точка  $\mathbf{P}_i$  может влиять на форму кривой  $\mathbf{C}(u)$  только на интервале  $[u_i, u_{i+p+1})$ ;
- при движении вдоль кривой, функции  $N_{i,p}$  действуют подобно переключателям. Когда  $u$  проходит мимо узла  $u_{i+p+1}$  в векторе узлов, функция  $N_{i,p}$  (и, соответственно, точка  $\mathbf{P}_i$ ) выключаются, поскольку становится равной нулю, и включаются следующие;
- чем меньше степень кривой, тем ближе она подходит к контрольным точкам. Кривые высоких порядков более гладкие;

- помимо локального контроля В-сплайны позволяют варьировать число контрольных точек, используемых в разработке кривой, без изменения степени полинома;
- кривые на базе В-сплайнов аффинно инвариантны. Для преобразования В-сплайн кривой мы просто преобразуем каждую контрольную точку и генерируем новую кривую;
- В-сплайн кривая является выпуклой комбинацией своих контрольных точек и поэтому лежит внутри их выпуклой оболочки. Возможно более сильное утверждение: при любом значении  $u \in [u_p, u_{n+1}]$  только  $p + 1$  функций В-сплайна «активны» (то есть отличны от нуля). В этом случае кривая должна лежать внутри выпуклой оболочки не более  $p + 1$  последовательных активных контрольных точек;
- В-сплайн кривые обеспечивают линейную точность: если  $p + 1$  последовательных контрольных точек коллинеарны, то их выпуклая оболочка будет прямой линией, и кривая будет захвачена внутрь её;
- В-сплайн кривые уменьшают колебания: В-сплайн кривая не пересекает никакую линию чаще, чем её контрольный полигон.

### 1.4.3 Производные В-сплайн кривой

Обозначим за  $\mathbf{C}^k(u)$   $k$ -ую производную кривой  $\mathbf{C}(u)$ . Если зафиксировать  $u$ , тогда мы можем получить  $\mathbf{C}^k(u)$  вычисляя  $k$ -ую производную базисных функций с помощью следующих формул:

$$N'_{i,p} = \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (1.18)$$

$$N^{(k)}_{i,p} = p \left( \frac{N^{(k-1)}_{i,p-1}}{u_{i+p} - u_i} - \frac{N^{(k-1)}_{i+1,p-1}}{u_{i+p+1} - u_{i+1}} \right) \quad (1.19)$$

А также можно использовать алгоритм, приведенный в Листинге А.1 в ПРИЛОЖЕНИИ А.



Тогда  $k$ -ая производная кривой  $\mathbf{C}(u)$

$$\mathbf{C}^k(u) = \sum_{i=0}^n N_{i,p}^{(k)} \mathbf{P}_i, \quad (1.20)$$

которую можно вычислить с помощью следующего алгоритма:

Листинг 1.5 – Алгоритм вычисления производных кривой

```

1  CurveDerivs(n, p, U, P, u, d, CK)
2  {
3      /*Вычисление производных кривой*/
4      /*[in]: n, p, U, P, u, d*/
5      /*[out]: CK*/
6      du = min(d,p);
7      for(k=p+1; k<=d; k++) CK[k]=0.0;
8      span = FindSpan(n,p,u,U);
9      dersBasisFunc(span, u, p, du, U, nders); // nders -
          результат функции dersBasisFunc, т.е. производные
          базисных функций
10     for(k=0; k<=du; k++)
11     {
12         CK[k] = 0.0;
13         for(j=0; j<=p; j++){
14             CK[k] = CK[k] + nders[k][j]*P[span-p+j];
15         }
16     }
17 }
```

#### 1.4.4 В-сплайн поверхности

В-сплайн поверхность задается с помощью контрольных точек и двух векторов узлов. Ее точки можно найти с помощью формулы:

$$S(u, v) = \sum_{i=0}^n \sum_{j=0}^m N_{i,p}(u) N_{j,q}(v) \mathbf{P}_{ij} \quad (1.21)$$

Например, для набора контрольных точек

$$\begin{aligned}
\mathbf{P}_{0,0} &= (0, 0, 0), & \mathbf{P}_{0,1} &= (1, 0, 0), & \mathbf{P}_{0,2} &= (2, 0, 0), \\
\mathbf{P}_{1,0} &= (0, 0.5, 1.3), & \mathbf{P}_{1,1} &= (1, 0.5, 1.2), & \mathbf{P}_{1,2} &= (2, 0.5, 1.3), \\
\mathbf{P}_{2,0} &= (0, 1, 0), & \mathbf{P}_{2,1} &= (1, 1, 0), & \mathbf{P}_{2,2} &= (2, 1, 0), \\
\mathbf{P}_{3,0} &= (0, 1.5, 1.3), & \mathbf{P}_{3,1} &= (1, 1.5, 1.2), & \mathbf{P}_{3,2} &= (2, 1.5, 1.3), \\
\mathbf{P}_{4,0} &= (0, 2, 0), & \mathbf{P}_{4,1} &= (1, 2, 0), & \mathbf{P}_{4,2} &= (2, 2, 0),
\end{aligned}$$

векторов узлов  $U = \{0, 0, 0, 1/2, 1/2, 1, 1, 1\}$ ,  $V = \{0, 0, 0, 1, 1, 1\}$  и  $p = q = 2$  получим следующую В-сплайн поверхность:

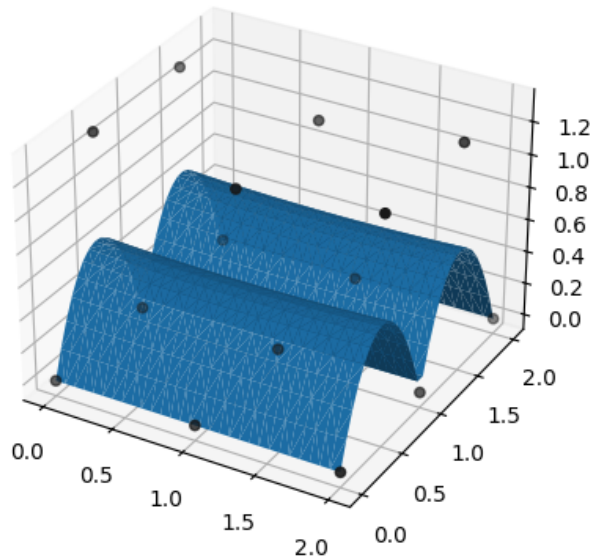


Рисунок 1.7 – Пример В-сплайн поверхности

## 1.5 NURBS

### 1.5.1 Рациональный В-сплайн и NURBS

Аналогично случаю рациональных кривых Безье, контрольные точки рационального В-сплайна указываются с использованием однородных координат. Функции сопряжения применяются именно к этим однородным координатам. Координаты точки рационального В-сплайна в однородном пространстве получаются по формулам:

$$X(u) = \sum_{i=0}^n N_{i,p}(u) \omega_i x_i \quad Y(u) = \sum_{i=0}^n N_{i,p}(u) \omega_i y_i$$

$$Z(u) = \sum_{i=0}^n N_{i,p}(u) \omega_i z_i \quad W(u) = \sum_{i=0}^n N_{i,p}(u) \omega_i$$

Тогда уравнение рационального В-сплайна в трехмерном пространстве в векторном виде примет вид:

$$\mathbf{C}(u) = \frac{\sum_{i=0}^n N_{i,p}(u) \omega_i \mathbf{P}_i}{\sum_{i=0}^n N_{i,p}(u) \omega_i} = \sum_{i=0}^n R_{i,p} \mathbf{P}_i, \quad (1.22)$$

где

$$R_{i,p} = \frac{N_{i,p}(u) \omega_i}{\sum_{j=0}^n N_{j,p}(u) \omega_j}$$

— базисные функции рационального В-сплайна.

Рациональные В-сплайны и их базисы это обобщение нерациональных В-сплайнов и базисов. При  $\omega_i \geq 0$  для всех  $i$  они наследуют почти все аналитические и геометрические свойства последних. В частности:

- каждая функция рационального базиса положительна или равна нулю для всех значений параметра, т.е.  $R_{i,p} \geq 0$ ;
- при  $p > 0$  каждая функция  $R_{i,p}(u)$  имеет ровно один максимум;
- рациональный В-сплайн степени  $p$  имеет непрерывность  $C^{p-1}$ ;
- максимальная степень рационального В-сплайна равна количеству контрольных точек минус 1;
- если  $u \in [u_i, u_{i+1})$ , то  $\mathbf{C}(u)$  находится в пределах выпуклой оболочки, составленной из контрольных точек  $\mathbf{P}_{i-p}, \dots, \mathbf{P}_i$ ;
- свойство локальности  $R_{i,p}(u) = 0$  для  $u \notin [u_i, u_{i+p+1})$ . Если  $u \in [u_i, u_{i+1})$ , только функции  $R_{i-p,p}, \dots, R_{i,p}$  являются ненулевыми;
- аффинная и перспективная инвариантность: применяемое к кривой преобразование можно свести к преобразованию только ее контрольных точек;
- свойство уменьшения вариации: прямая или плоскость пересекают сплайн не большее количество раз чем контрольный полигон сплайна.

Также с помощью квадратичных рациональных В-сплайнов можно целиком построить окружность или какое-либо другое коническое сечение. То есть с их помощью можно сшить отдельные дуги, представляемые с помощью рациональных сплайнов Безье.

Окружность можно задать множеством различных способов, например, задав в вершинах и на серединах сторон девять контрольных точек, причём начальная и конечная вершины должны совпадать в одной из середин (Рисунок 1.8). Узловой вектор можно задать в следующем виде:  $\{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\}$ . Вес контрольной точки  $\omega_i = 1$ , если  $i$  — четное и  $\omega_i = \frac{\sqrt{2}}{2}$ , если  $i$  — нечетное.

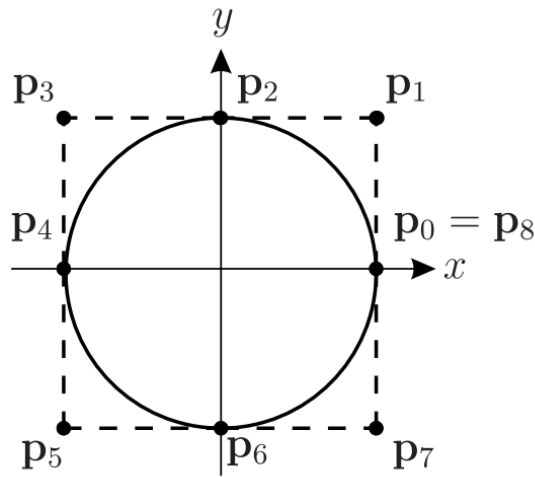


Рисунок 1.8 – Способ задания контрольных точек для окружности

Обычно в пакетах графической разработки для построения рациональных В-сплайнов используются неравномерные представления вектора узлов. Данные сплайны называются «*NURBS*» (Nonuniform Rational B-splines — неравномерные рациональные В-сплайны). NURBS с 1983 г. являются стандартом IGES. IGES — это стандарт обмена проектной информацией между системами автоматизированного проектирования, а также между ними и системами автоматизированного производства.

### 1.5.2 Производная NURBS-кривой

Выше были приведены формулы и алгоритмы для вычисления производных В-сплайн кривой. Эти формулы можно применять и к  $\mathbf{C}^\omega(u)$ , поскольку это нерациональная кривая в четырехмерном пространстве. Таким образом, производные кривой  $\mathbf{C}(u)$  можно выразить через производные кривой  $\mathbf{C}^\omega(u)$ .

Пусть

$$\mathbf{C}(u) = \frac{\omega(u)\mathbf{C}(u)}{\omega(u)} = \frac{\mathbf{A}(u)}{\omega(u)}, \quad (1.23)$$

где  $\mathbf{A}(u)$  - векторная функция, координаты которой являются первыми тремя координатами функции  $\mathbf{C}^\omega(u)$ . Тогда

$$\begin{aligned} \mathbf{C}'(u) &= \frac{\omega(u)\mathbf{A}'(u) - \omega'(u)\mathbf{A}(u)}{\omega(u)^2} = \\ &= \frac{\omega(u)\mathbf{A}'(u) - \omega'(u)\omega(u)\mathbf{C}(u)}{\omega(u)^2} = \frac{\mathbf{A}'(u) - \omega'(u)\mathbf{C}(u)}{\omega(u)} \end{aligned} \quad (1.24)$$

Так как  $\mathbf{A}(u)$  и  $\omega(u)$  представляют собой координаты  $\mathbf{C}^\omega(u)$ , можно получить первые производные используя уравнение (1.20). Последующие производные можно получить, дифференцируя функцию  $\mathbf{A}$ , используя правило Лейбница:

$$\begin{aligned} \mathbf{A}^{(k)}(u) &= (\omega(u)\mathbf{C}(u))^{(k)} = \sum_{i=0}^k \binom{k}{i} \omega^{(i)}(u) \mathbf{C}^{(k-i)}(u) = \\ &= \omega(u) \mathbf{C}^{(k)}(u) + \sum_{i=1}^k \binom{k}{i} \omega^{(i)}(u) \mathbf{C}^{(k-i)}(u) \end{aligned} \quad (1.25)$$

откуда получаем

$$\mathbf{C}^{(k)}(u) = \frac{\mathbf{A}^{(k)}(u) - \sum_{i=1}^k \binom{k}{i} \omega^{(i)}(u) \mathbf{C}^{(k-i)}(u)}{\omega(u)} \quad (1.26)$$

Производные  $\mathbf{A}^{(k)}(u)$  и  $\omega^{(i)}(u)$  могут быть получены с помощью алгоритма, приведенного в Листинге 1.5.

Теперь, если  $u$  зафиксировано, а производные от нулевой до  $d$  функций  $\mathbf{A}(u)$  и  $\omega(u)$  вычислены и загружены в массивы `Aders` и `wders` соответственно, т.е.  $\mathbf{C}^\omega(u)$  продифференцировано и его координаты разделены на массивы `Aders` и `wders`, то с помощью алгоритма, приведенного в Листинге 1.6, можно вычислить точку кривой и все производные в ней  $\mathbf{C}^{(k)}(u)$ ,  $1 \leq k \leq d$ .

Листинг 1.6 – Алгоритм вычисления производных рациональной В-сплайн кривой

1 | `RatCurveDerivs(Aders, wders, d, CK)`

```

2 {
3     /*Вычисление точки на поевности Безье*/
4     /*[in]: Aders, wders, d, CK*/
5     /*[out]: CK*/
6     for(k=0; k<=d; k++)
7     {
8         v = Aders[k];
9         for(i=1; i<=k; i++)
10             v = v - Bin[k][i]*wders[i]*CK[k-i];
11         CK[k] = v/wders[0];
12     }
13 }

```

### 1.5.3 NURBS-поверхности

Для задания NURBS-поверхности будем использовать следующую формулу:

$$\mathbf{S}(u, v) = \frac{\sum_{i=0}^n \sum_{j=0}^m \omega_{ij} N_{i,p}(u) N_{j,q}(v) \mathbf{P}_{ij}}{\sum_{i=0}^n \sum_{j=0}^m \omega_{ij} N_{i,p}(u) N_{j,q}(v)}, \quad (1.27)$$

где  $\mathbf{P}_{ij}$  — контрольные точки,  $\omega_{ij}$  — их веса.

Благодаря общности и гибкости NURBS-поверхности стали пользоваться популярностью. Поскольку В-сплайны являются частным случаем NURBS-поверхностей (при  $\omega_{ij} = 1$ ), можно использовать единый алгоритм для создания обширного семейства поверхностей. NURBS поверхности обладают большинством свойств, присущих В-сплайновым поверхностям и NURBS-кривым. NURBS поверхности позволяют точно описывать квадратичные поверхности, такие как цилиндр, конус, сфера, параболоид и гиперболоид. Поэтому дизайнеру вместо инструментария, состоящего из большого числа различных алгоритмов для создания поверхностей, потребуется всего один метод.

## 2 Поверхности движения

Многие модели или заготовки для них можно получить с помощью заметания (sweeping), т.е. путём движения кривой по заданной траектории. Такие объекты обладают трансляционной, вращательной или другой симметрией. Пусть траектория движения описывается кривой  $\mathbf{g}(v)$ , которую будем называть *направляющей*. Движущуюся по траектории кривую линию будем называть *образующей кривой*. Направляющая кривая и образующая кривая не должны иметь точек самопересечения. Набор таких двумерных примитивов, как окружности и прямоугольники, может предлагаться в качестве образующих как пункты меню. Существуют и другие методы получения двумерных фигур, например, построение замкнутых сплайновых кривых. Если образующая кривая не замкнута, то на её основе в общем случае нельзя построить тело. Обычно из незамкнутой кривой создаётся замкнутая составная кривая путём «придания ей толщины» с помощью эквидистантных кривых. В общем случае образующая представляет собой замкнутую составную фигуру. Если образующая является плоской кривой, то можно построить тело с плоскими торцами. В популярной системе трёхмерной графики 3Ds Max заметание называется Loft, в описаниях пакета на русском языке его именуют лофтингом.

### 2.1 Поверхность выдавливания

Пусть задана некоторая кривая  $\mathbf{C}(u)$  и единичный вектор  $\mathbf{d}$ . Если направляющей движения контура служит отрезок прямой  $\mathbf{g}(v) = \mathbf{P} + v\mathbf{h}\mathbf{d}$ ,  $0 \leq v \leq 1$ , то мы получим поверхность выдавливания по направлению  $\mathbf{d}$ . Эта поверхность будет описываться следующей формулой:

$$\mathbf{S}(u, v) = \mathbf{C}(u) + v\mathbf{h}\mathbf{d} \quad (2.1)$$

Тогда, например, если в качестве кривой  $\mathbf{C}(u)$  взять NURBS окружность (Рисунок 2.1), то можно получить поверхность выдавливания изображённую на Рисунке 2.2.

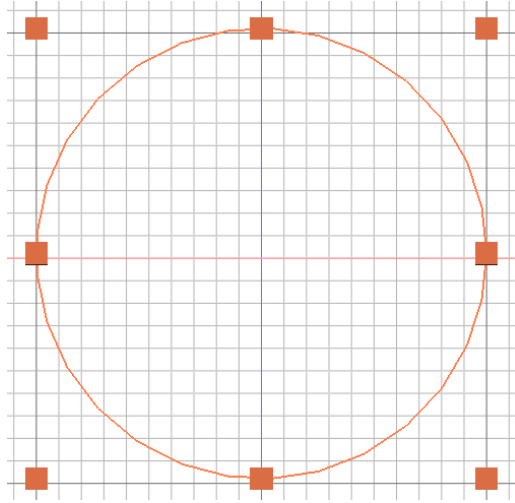


Рисунок 2.1 – Окружность, выполненная с помощью NURBS

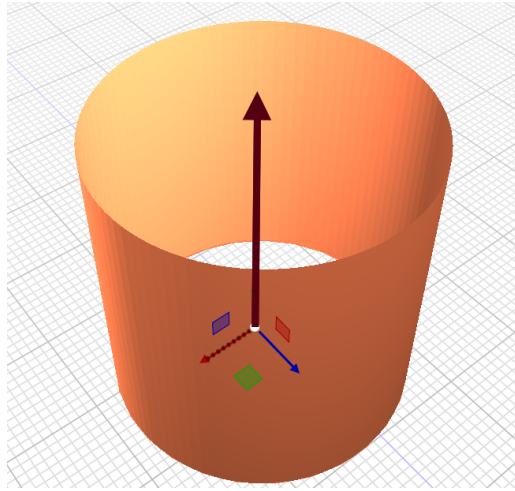


Рисунок 2.2 – Поверхность выдавливания

## 2.2 Поверхность вращения

Пусть кривая  $\mathbf{C}(u)$  задана в плоскости  $XZ$ . Для создания поверхности вращения повернём эту кривую вокруг оси  $z$ , изменяя параметр  $v$ , где  $v$  определяет угол, под которым каждая точка повернута относительно оси. И пусть точка  $\mathbf{P}_s = \{x_s, 0, 0\}$  - задает положения оси вращения в локальной системе координат. Тогда поверхность вращения можно получить с помощью следующей формулы:

$$\mathbf{S}(u, v) = \mathbf{P}_s + \mathbf{M}(v)(\mathbf{C}(u) - \mathbf{P}_s), \quad (2.2)$$



где

$$\mathbf{M}(v) = \begin{pmatrix} \cos v & 0 & 0 \\ \sin v & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

Так, например, для NURBS окружность построение поверхности вращения и результат будет выглядеть следующим образом:

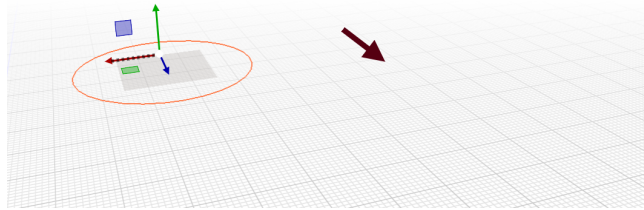


Рисунок 2.3 – Пример построения поверхности вращения

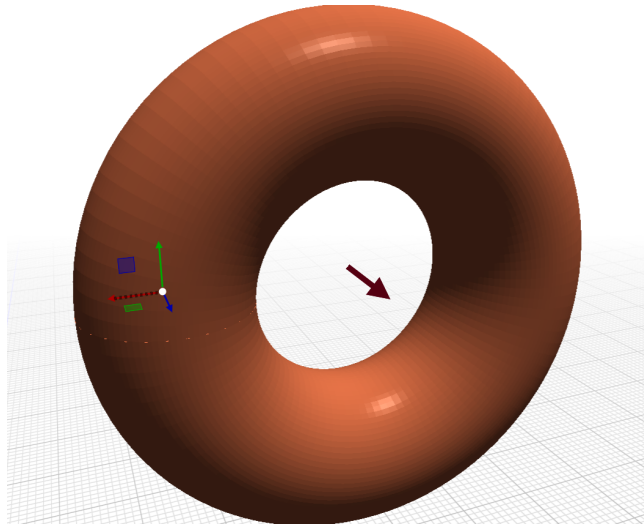


Рисунок 2.4 – Поверхность вращения

## 2.3 Поверхность сдвига

В общем случае кривая  $\mathbf{g}(v)$  не обязательно задает прямую. Тогда с помощью формулы

$$\mathbf{S}(u, v) = \mathbf{g}(v) + (\mathbf{C}(u) - \mathbf{g}(v_{\min})) \quad (2.4)$$

можно получить поверхность сдвига.

Для направляющей кривой, изображенной на Рисунке 2.5, и образующей NURBS окружности получим поверхность сдвига изображенную на Рисунке 2.6.

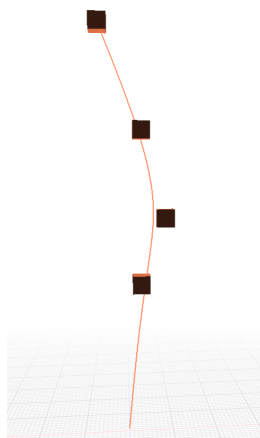


Рисунок 2.5 – Направляющий сплайн

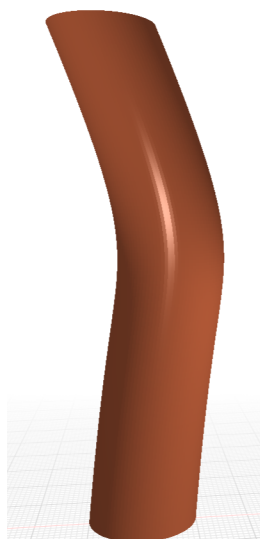


Рисунок 2.6 – Поверхность сдвига

## 2.4 Кинематические поверхности

Для построения кинематической поверхности необходимо вычислить подвижную декартову систему координат. Первый базисный вектор подвижной СК  $\mathbf{i}_1 = \frac{\mathbf{g}'}{|\mathbf{g}'|}$  направим по касательной к направляющей кривой. Вторым  $\mathbf{i}_2$  направим ортогонально первому, а  $\mathbf{i}_3 = \mathbf{i}_1 \times \mathbf{i}_2$ .

Тогда для вычисления радиус-вектора точки кинематической поверхно-

сти построим матрицу

$$\mathbf{A}(v) = [\mathbf{i}_1(v) \quad \mathbf{i}_2(v) \quad \mathbf{i}_3(v)]. \quad (2.5)$$

Матрица  $\mathbf{A}(v)$  является матрицей преобразования координат радиус-вектора точки из подвижной СК в глобальную и зависит от параметра направляющей кривой.

Запомним положение образующей кривой  $\mathbf{C}(u)$  в подвижном касательном базисе в начале направляющей и будем сохранять его при движении вдоль направляющей. Тогда радиус-вектор точки образующей в подвижной СК при  $v = v_{\min}$  равен

$$\mathbf{X}(u, v_{\min}) = \mathbf{A}^{-1}(v_{\min}) \cdot (\mathbf{C}(u) - \mathbf{g}(v_{\min})). \quad (2.6)$$

При движении вдоль направляющей кривой подвижный касательный базис меняет свое положение и ориентацию в пространстве и увлекает за собой жестко связанную с ним образующую кривую. Вектор  $\mathbf{X}(u, v_{\min})$  выражает положение точки образующей относительно точки на направляющей кривой в подвижном базисе, которое сохраняется для произвольного параметра  $v$ . Переходя из подвижной СК в глобальную при текущем параметре  $v$ , получим радиус-вектор точки на кинематической поверхности

$$\mathbf{S}(u, v) = \mathbf{g}(v) + \mathbf{A}(v) \cdot \mathbf{X}(u, v_{\min}). \quad (2.7)$$

Таким образом, радиус вектор кинематической поверхности опишем следующей функцией

$$\mathbf{S}(u, v) = \mathbf{g}(v) + \mathbf{M}(v) \cdot (\mathbf{C}(u) - \mathbf{g}(v_{\min})) \quad (2.8)$$

где  $\mathbf{M}(v)$  - матрица поворота текущего подвижного базиса относительно его начального положения. Эта матрица вычисляется по формуле

$$\mathbf{M}(v) = \mathbf{A}(v) \cdot \mathbf{A}^{-1}(v_{\min}) \quad (2.9)$$

Так, например, для направляющей, изображенной на Рисунке 2.7, и образующей NURBS окружности получим кинематическую поверхность, изоб-

раженную на Рисунке 2.8.

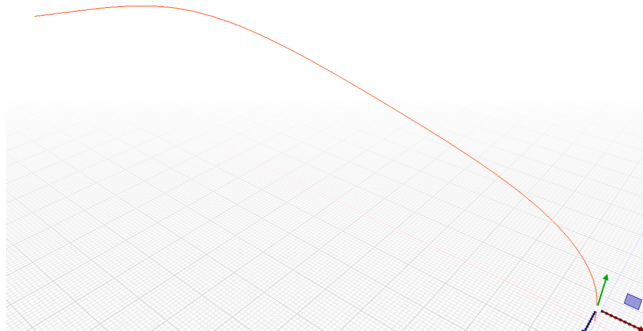


Рисунок 2.7 – Направляющий сплайн

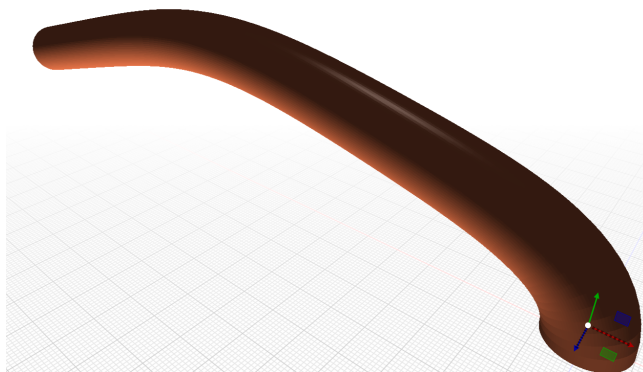


Рисунок 2.8 – Поверхность сдвига

## 3 Программная реализация

В рамках курсовой работы была выполнена программная реализация выше изложенных методов на языке C++. Она состоит из графической оболочки, поддерживающую визуализацию поверхностей и кривых, а также инструментов для создания кривых и поверхностей движения. Отрисовка двух и трехмерных объектов на экране осуществлена с помощью OpenGL.

Таким образом, программа разбита на классы отвечающие за отрисовку (Renderer) и классы осуществляющие математическую реализацию кривых и поверхностей.

### 3.1 Графическая оболочка

Графическая оболочка написана с использованием библиотеки с открытым исходным кодом для создания пользовательских интерфейсов Dear ImGui, предоставляющая отрисовку интерфейса в фреймбуффер OpenGL. Общий вид графической оболочки можно увидеть на Рисунке 3.1.

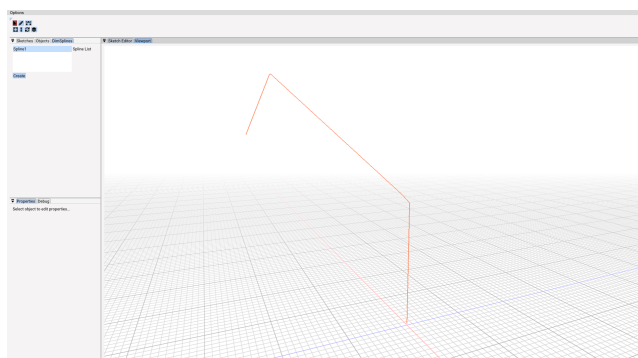


Рисунок 3.1 – Графическая оболочка

Пользовательский интерфейс состоит из окон: инструментов (в верхней части экрана), загруженных кривых (Sketches), кривых, определенных в рабочем пространстве (DimSplines), свойств выбранного объекта на сцене (Properties). А также окна рабочего пространства (Viewport) и окна редактирования кривых (Sketch Editor).

В окне редактирования кривых можно создавать или редактировать кривые (Рисунок 3.2). В свою очередь тип кривой можно выбрать в окне Properties, как показано на Рисунке 3.3.

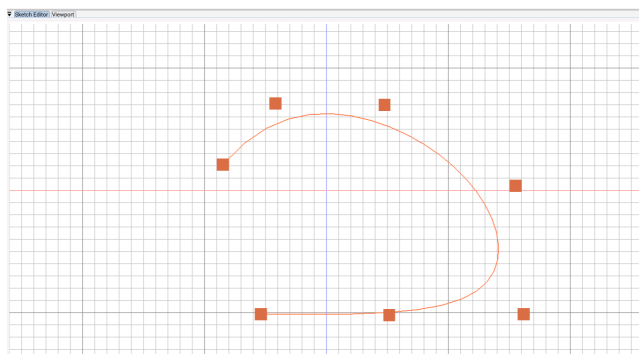


Рисунок 3.2 – Окно Sketch Editor

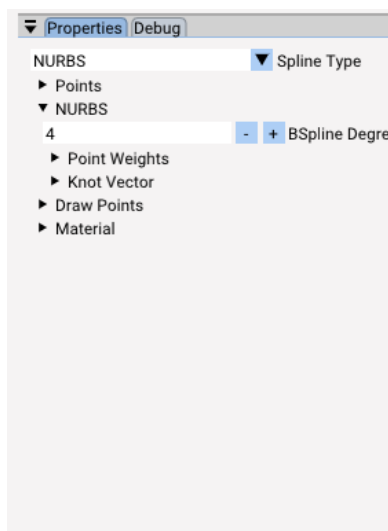


Рисунок 3.3 – Окно Properties

## 3.2 Инструменты создания поверхностей движения

Перед созданием поверхности движения нужно создать или загрузить готовую обрезающую кривую в окнах Sketch и Sketch Editor. Затем если необходимо создать опорную плоскость, с помощью соответствующего инструмента, а затем выбрать инструмент создания требуемой поверхности движения.

Рассмотрим пример работы с программой для создания поверхностей выдавливания и вращения.

### 3.2.1 Поверхность выдавливания

После создания опорной плоскости, прикрепления к ней нужной образующей кривой и выбрав инструмент создания поверхности выдавливания (Extrude), в окне Viewport программы отобразятся вспомогательные элементы для создания поверхности выдавливания (Рисунок 3.4). Также в окне Extrude

можно задать свойства поверхности выдавливания: направление выдавливания и длину (Рисунок 3.5). При нажатии кнопки Create будет создана поверхность с выбранными свойствами, как, например, на Рисунке (2.2).

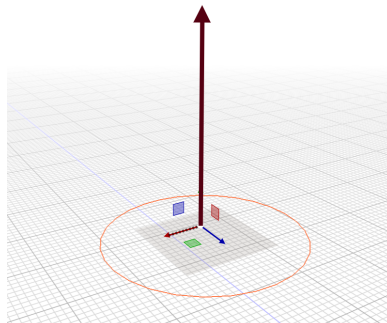


Рисунок 3.4 – Инструмент создания поверхности выдавливания

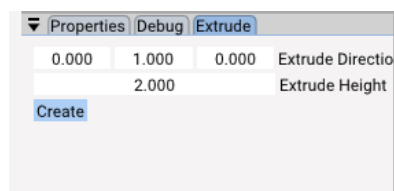


Рисунок 3.5 – Свойства поверхности выдавливания

### 3.2.2 Поверхность вращения

Аналогичным образом, после создания опорной плоскости, образующей кривой и выбора инструмента создания поверхности вращения отобразится вспомогательная графика - ось вращения (Рисунок 3.6), а также в окне Rotate отобразятся свойства создаваемой поверхности: радиус и угол вращения.

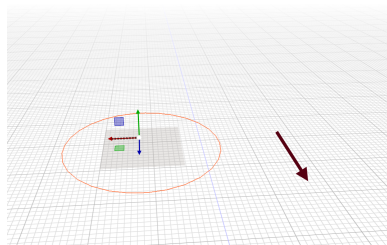


Рисунок 3.6 – Инструмент создания поверхности вращения

## ЗАКЛЮЧЕНИЕ

В рамках курсовой работы были рассмотрены методы построения поверхностей и кривых, а именно кривые и поверхности Безье, рациональные кривые и поверхности Безье, B-Spline и NURBS. А также алгоритмы для их построения на компьютере.

Затем были рассмотрены методы построения поверхностей движения, таких как поверхности выдавливания, сдвига, вращения и кинематические поверхности.

В рамках курсовой работы была выполнена программная реализация выше изложенных методов, которая поддерживает создание и редактирование кривых в реальном времени, проектирование поверхностей движения с различными параметрами, а также визуализацию полученных математических объектов на экране.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *L. Piegl, W. Tiller.* The NURBS Book. — Berlin : Springer, 1996.
2. *Н. Н. Голованов.* Геометрическое моделирование. — Москва : Издательство Физико-математической литературы, 2002.
3. *А. Н. Божко, Д. М. Жук, В. Б. Маничев.* Компьютерная графика : учеб. пособие для вузов. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2007.
4. *Д. Херн, М. П. Бейкер.* Компьютерная графика и стандарт OpenGL. — Москва : Издательский дом "Вильямс", 2005.
5. *Ю. Н. Косников.* Геометрические преобразования в компьютерной графике. — Пенза : Издательство ПГУ, 2011.

# ПРИЛОЖЕНИЕ А

## Алгоритмы

Листинг А.1 – Алгоритм вычисления производных базисных функций на языке C++

```
1 Matrix<float> dersBasisFunc(int i, float u, int p, int n,
2                             std::vector<float> U) {
3     Matrix<float> ders(n + 1, p + 1);
4     Matrix<float> ndu(p + 1, p + 1);
5     std::vector<float> left(p + 1, 0);
6     std::vector<float> right(p + 1, 0);
7     Matrix<float> a(2, p + 1);
8     ndu[0][0] = 1.0f;
9     for (auto j = 1; j <= p; j++) {
10         left[j] = u - U[i + 1 - j];
11         right[j] = U[i + j] - u;
12         float saved = 0.0f;
13         for (auto r = 0; r < j; r++) {
14             ndu[j][r] = right[r + 1] + left[j - r];
15             float temp = ndu[r][j - 1] / ndu[j][r];
16             ndu[r][j] = saved + right[r + 1] * temp;
17             saved = left[j - r] * temp;
18         }
19         ndu[j][j] = saved;
20     }
21     for (auto j = 0; j <= p; j++) {
22         ders[0][j] = ndu[j][p];
23     }
24     for (auto r = 0; r <= p; r++) {
25         int s1 = 0;
26         int s2 = 1;
27         a[0][0] = 1.0f;
28         for (auto k = 1; k <= n; k++) {
29             float d = 0.0f;
30             int rk = r - k;
31             int pk = p - k;
32             if (r >= k) {
33                 a[s2][0] = a[s1][0] / ndu[pk + 1][rk];
34                 d = a[s2][0] * ndu[rk][pk];
35             }
36             int j1 = rk >= -1 ? 1 : -rk;
```

```

37     int j2 = r - 1 <= pk ? k - 1 : p - r;
38     for (auto j = j1; j <= j2; j++) {
39         a[s2][j] = (a[s1][j] - a[s1][j - 1]) / ndu[pk + 1][rk +
40             j];
41         d += a[s2][j] * ndu[rk + j][pk];
42     }
43     if (r <= pk) {
44         a[s2][k] = -a[s1][k - 1] / ndu[pk + 1][r];
45         d += a[s2][k] * ndu[r][pk];
46     }
47     ders[k][r] = d;
48     int j = s1;
49     s1 = s2;
50     s2 = j;
51 }
52 int r = p;
53 for (auto k = 1; k <= n; k++) {
54     for (auto j = 0; j < p; j++) {
55         ders[k][j] *= r;
56     }
57     r *= p - k;
58 }
59 return ders;
60 }

```