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

ВВЕДЕНИЕ

Для моделирование построения поверхностных и объемных геометрий необходимо использовать методы геометрического моделирования.

Поэтому, прежде, чем мы приступим к описанию построения геометрий с помощью операции движения, рассмотрим основные методы построения поверхностей и кривых, а именно кривые и поверхности Безье, рациональные кривые и поверхности Безье, B-Spline и NURBS.

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 Неявные 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:

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

и также будем пользоваться обозначением

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

Для поверхности:

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

и также будем пользоваться обозначением

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

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:

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

где $B_{i,n}$ - полиномы Бернштейна.

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

Для вычисления точек кривой Безье удобно использовать алгоритм де Кастельжо:

Листинг 1.1 – Псевдокод алгоритма де Кастельжо

```
deCasteljau(P, n, u, C)
{
    /*Вычисление точки на кривой Безье*/
    /*[in]: P, n, u*/
    /*[out]: C (точка)*/
    for(i=0; i<=n; i++) /* Используем локальный массив, */
        Q[i] = P[i]      /* чтобы не изменить исходный массив
            контрольных точек */
    for(k=1; k<=n; k++)
        for(i=0; i<=n-k; i++)
            Q[i] = (1.0-u)*Q[i] + u*Q[i+1]
    C = Q[0]
}
```

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

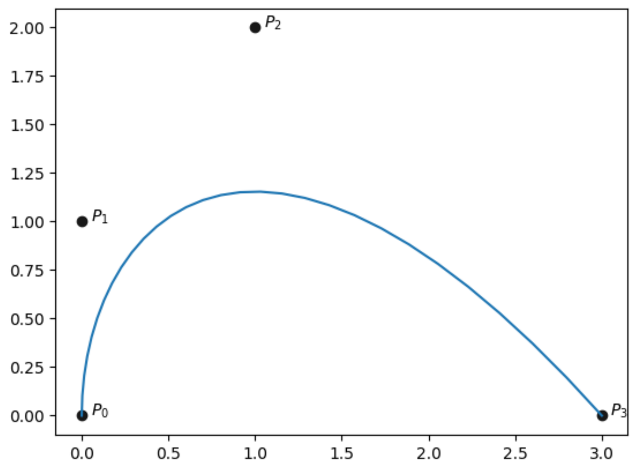

Рисунок 1.1 – Пример кривой Безье

1.2.2 Поверхности Безье

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

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

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

```
deCasteljauForSurface(P, n, m, u0, v0, S)
{
    /*Вычисление точки на повехности Безье*/
    /*[in]: P, n, m, u0, v0*/
    /*[out]: S (точка)*/
    if (n <= m)
    {
        for(j=0; j<=m; j++) /* P[j][] - j-ая строка */
            deCasteljau(P[j][], n, u0, Q[j]);
        deCasteljau(Q, m, v0, S);
    }
    else
    {
        for(i=0; i<=n; i++)
            deCasteljau(P[][i], m, v0, Q[i]);
        deCasteljau(Q, n, u0, S);
    }
}
```

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

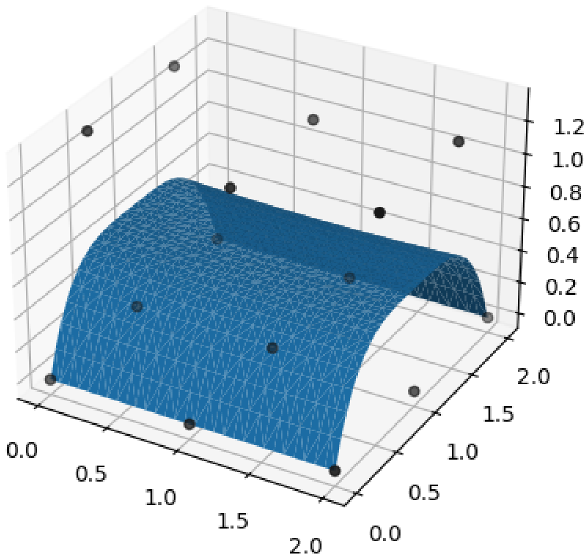

Рисунок 1.2 – Пример поверхности Безье

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.

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

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^ω на гиперплоскость $W = 1$

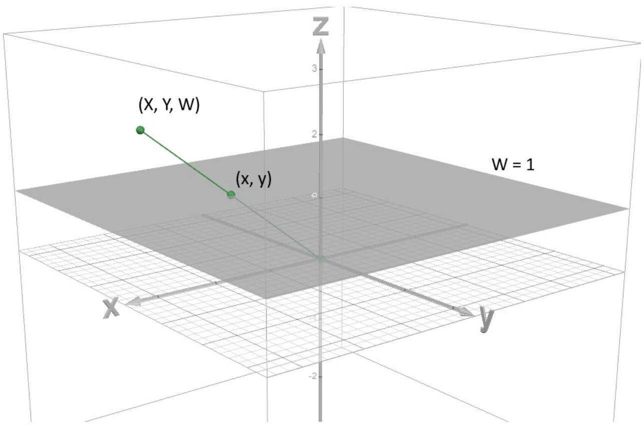

Рисунок 1.3 – Представление точки евклидова пространства в однородной форме для двумерного случая

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

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

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

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

Уравнение (1.10) в координатном виде:

$$X(u) = \sum_{i=0}^{n} B_{i,n}(u)\omega_i x_i \qquad Y(u) = \sum_{i=0}^{n} B_{i,n}(u)\omega_i y_i$$

$$Z(u) = \sum_{i=0}^{n} B_{i,n}(u)\omega_i z_i \qquad W(u) = \sum_{i=0}^{n} B_{i,n}(u)\omega_i$$

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

$$x(u) = \frac{X(u)}{W(u)} = \frac{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i x_i}{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i}$$

$$y(u) = \frac{Y(u)}{W(u)} = \frac{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i y_i}{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i}$$

$$z(u) = \frac{Z(u)}{W(u)} = \frac{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i z_i}{\sum\limits_{i=0}^{n} B_{i,n}(u)\omega_i}$$

или в векторной записи

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

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).

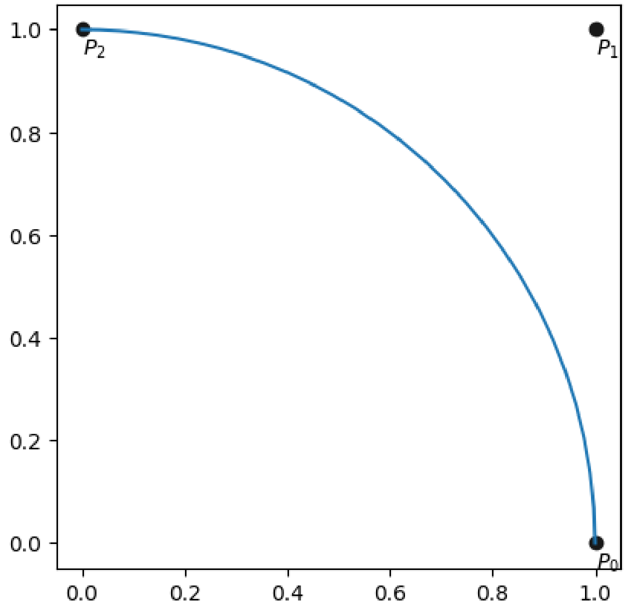

Рисунок 1.4 – Дуга окружности, построенная с помощью рациональной кривой Безье

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

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

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

где

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

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.

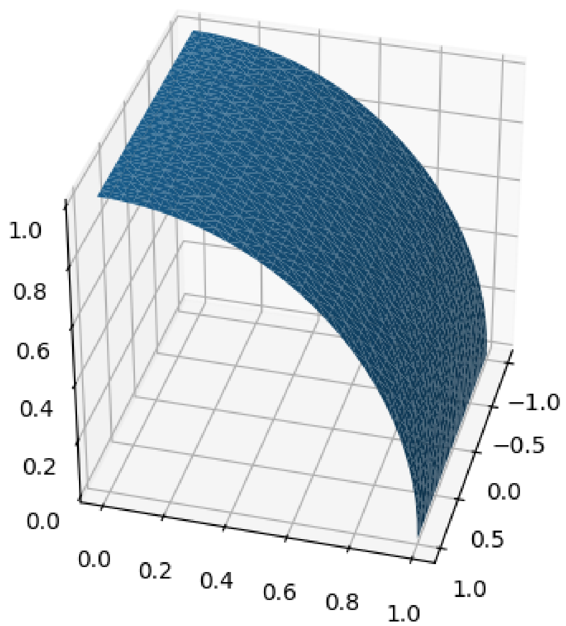

Рисунок 1.5 – Цилиндрическая поверхность, построенная с помощью рациональной поверхности Безье

1.4 B-сплайны

1.4.1 B-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

$$C(u) = \sum_{i=0}^{n} N_i(u)\mathbf{P_i}, \qquad u_{min} \le u \le u_{max} \tag{1.14}$$

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. $\sum\limits_{i=0}^{n} N_i(u) = 1$ 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:

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

В 1972 году Кокс и де Бур предложили использовать функции $N_{i,p}$, определяемые рекурсивно. Пусть $U = \{u_0, \ldots, u_m\}$ - неубывающая последовательно вещественных чисел, т.е. $u_i \le u_{i+1}, i = 0, \ldots, m-1$. u_i называют *узлами* (knot), а U - *вектором узлов* (knot vector). Тогда *i-тая базисная функция B-сплайна p-ой степени*, обозначаемая $N_{i,p}(u)$, выражается следующим образом:

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

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

В силу свойств базисных функций B-сплайна в любом заданном промежутке $[u_i, u_{i+1}]$ могут быть отличны от нуля только $p+1$ функций: $N_{i-p,p}, \ldots, N_{i,p}$. Например, единственные кубические функции, отличные от нуля на $[u_3, u_4]$ - это функции $N_0^3, \ldots, N_3^3$. 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

```
int FindSpan(n, p, u, U)
{
    /* Поиск i-го индекса в векторе узлов*/
    /*[in]: n, p, u, U*/
    /*[out]: i*/
    if(u == U[n+1]) return n; /*Специальный случай*/
    low = p; high = n+1; /*Бинарный поиск*/
    mid = (low+high)/2;
    while(u < U[mid] || u >= U[mid+1])
    {
        if(u < U[mid])  high = mid;
        else            low = mid;
        mid = (low+high)/2;
    }
    return mid;
}
```

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 $N_{i-p,p}, \ldots, N_{i,p}$ можно использовать следующий алгоритм:

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

```
BasicFunc(i, u, p, U, N)
{
    /*Вычисление ненулевых базисных функций*/
    /*[in]: i, u, p, U*/
    /*[out]: N*/
    N[0] = 1.0;
    for(j=1; j<=p; j++){
        left[j] = u-U[i+1-j];
        right[j] = U[i+j]-u;
        saved = 0.0;
        for(r=0; r<j; r++)
        {
            temp = N[r]/(right[r+1]+left[j-r]);
            N[r] = saved+right[r+1]*temp;
            saved = left[j-r]*temp;
        }
        N[j] = saved;
    }
}
```

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 диапозон 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:

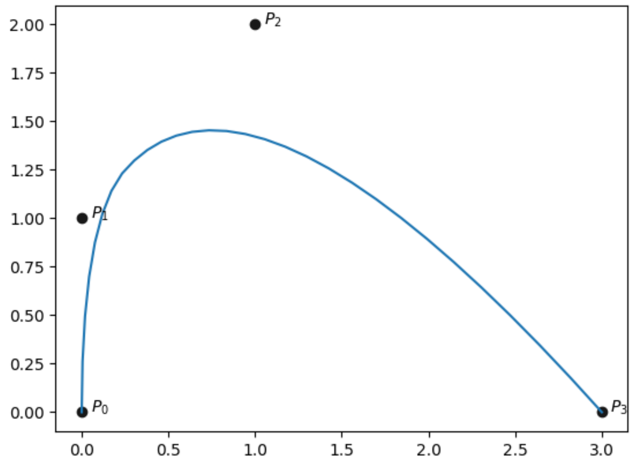

Рисунок 1.6 – Пример B-сплайн кривой

1.4.2 Свойства B-сплайна

B-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 $\{u_0, u_1, \ldots, u_{n+p+1}\}$, получающийся B-сплайн определяется только в промежутке $[u_p, u_{n+1})$, т.к. только в этом промежутке $\sum\limits_{i=0}^{n} N_{i,p} = 1$;

— 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 B-сплайны позволяют варьировать число контрольных точек, используемых в разработке кривой, без изменения степени полинома;

— кривые на базе B-сплайнов аффинно инвариантны. Для преобразования B-сплайн кривой мы просто преобразуем каждую контрольную точку и генерируем новую кривую;

— B-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 B-сплайна «активны» (то есть отличны от нуля). В этом случае кривая должна лежать внутри выпуклой оболочки не более $p+1$ последовательных активных контрольных точек;

— B-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;

— B-сплайн кривые уменьшают колебания: B-сплайн кривая не пересекает никакую линию чаще, чем её контрольный полигон.

1.4.3 Призводные B-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:

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

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

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

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

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

которую можно вычислить с помощью следующего алгоритма:

Листинг 1.5 – Алгоритм вычисления производных кривой

```
CurveDerivs(n, p, U, P, u, d, CK)
{
  /*Вычисление производных кривой*/
  /*[in]: n, p, U, P, u, d*/
  /*[out]: CK*/
  du = min(d,p);
  for(k=p+1; k<=d; k++) CK[k]=0.0;
  spand = FindSpan(n,p,u,U);
  dersBasisFunc(span, u, p, du, U, nders); // nders -
     результат функции dersBasisFunc, т.е. производные
     базисных функций
  for(k=0; k<=du; k++)
  {
      CK[k] = 0.0;
      for(j=0; j<=p; j++){
          CK[k] = CK[k] + nders[k][j]*P[span-p+j];
      }
  }
}
```

1.4.4 B-сплайн поверхности

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

$$S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) \mathbf{P_{i,j}} \tag{1.21}$$

Например, для набора контрольных точек

$$\begin{array}{ccc}
\mathbf{P_{0,0}} = (0,0,0), & \mathbf{P_{0,1}} = (1,0,0), & \mathbf{P_{0,2}} = (2,0,0), \\
\mathbf{P_{1,0}} = (0,0.5,1.3), & \mathbf{P_{1,1}} = (1,0.5,1.2), & \mathbf{P_{1,2}} = (2,0.5,1.3), \\
\mathbf{P_{2,0}} = (0,1,0), & \mathbf{P_{2,1}} = (1,1,0), & \mathbf{P_{2,2}} = (2,1,0), \\
\mathbf{P_{3,0}} = (0,1.5,1.3), & \mathbf{P_{3,1}} = (1,1.5,1.2), & \mathbf{P_{3,2}} = (2,1.5,1.3), \\
\mathbf{P_{4,0}} = (0,2,0), & \mathbf{P_{4,1}} = (1,2,0), & \mathbf{P_{4,2}} = (2,2,0),
\end{array}$$

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 B-сплайн поверхность:

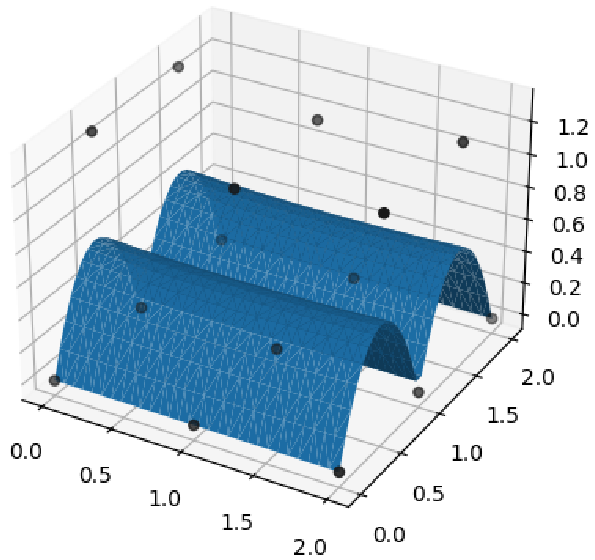

Рисунок 1.7 – Пример B-сплайн поверхности

1.5 NURBS

1.5.1 Рациональный B-сплайн и NURBS

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

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

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

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

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

где

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

— 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 $\mathbf{P}_{i-p}, \ldots, \mathbf{P}_i$;

— 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 $R_{i-p,p}, \ldots, R_{i,p}$ 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 B-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.

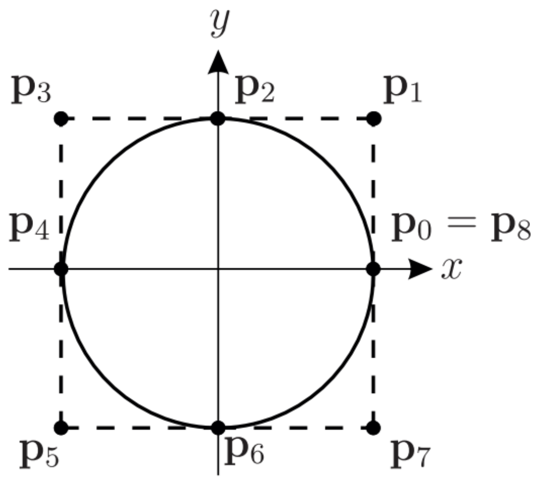

Рисунок 1.8 – Способ задания контрольных точек для окружности

Обычно в пакетах графической разработки для построения рациональных B-сплайнов используются неравномерные представления вектора узлов. Данные сплайны называются *«NURBS»* (Nonuniform Rational B-splines — неравномерные рациональные B-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 B-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

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

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

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

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:

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

откуда получаем

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

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 B-сплайн кривой

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

```
{
    /*Вычисление точки на повехности Безье*/
    /*[in]: Aders, wders, d, CK*/
    /*[out]: CK*/
    for(k=0; k<=d; k++)
    {
        v = Aders[k];
        for(i=1; i<=k; i++)
            v = v - Bin[k][i]*wders[i]*CK[k-i];
        CK[k] = v/wders[0];
    }
}
```

1.5.3 NURBS-поверхности

Для задания NURBS-поверхности будем использовать следующую формулу:

$$\mathbf{S}(u,v) = \frac{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{m}\omega_{ij}N_{i,p}(u)N_{j,q}(v)\mathbf{P_{i,j}}}{\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{m}\omega_{ij}N_{i,p}(u)N_{j,q}(v)}, \tag{1.27}$$

где $\mathbf{P_{ij}}$ — контрольные точки, ω_{ij} — их веса.

Благодаря общности и гибкости NURBS-поверхности стали пользоваться популярностью. Поскольку B-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 B-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 вектора $\mathbf{d}$. Если направляющей движения контура служит отрезок прямой $\mathbf{g}(v) = \mathbf{P} + vh\mathbf{d}$, $0 \leq v \leq 1$, то мы получим поверхность выдавливания по направлению $\mathbf{d}$. Эта поверхность будет описываться следующей формулой:

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

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

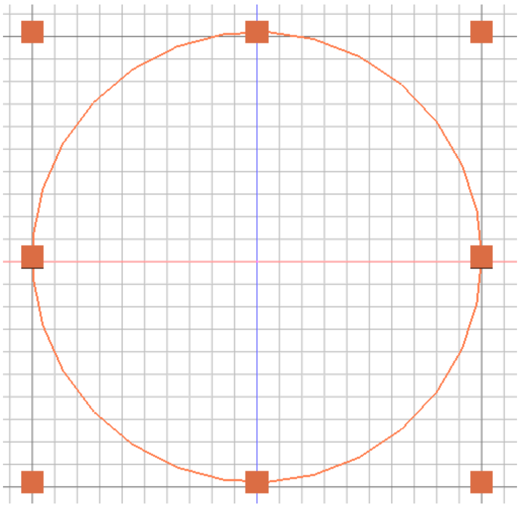

Рисунок 2.1 – Окружность, выполненная с помощью NURBS

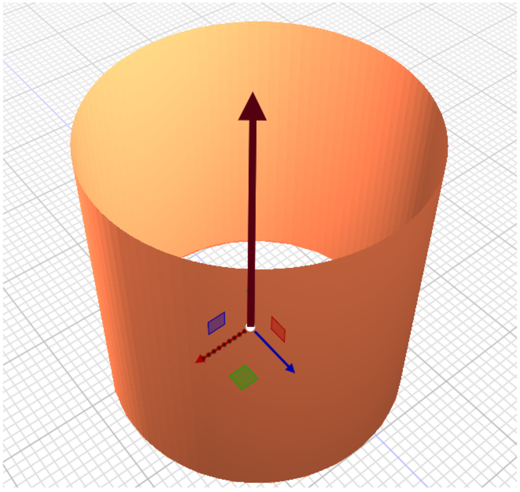

Рисунок 2.2 – Поверхность выдавливания

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:

$$\mathbf{S}(u, v) = \mathbf{P_s} + \mathbf{M}(v)(\mathbf{C}(u) - \mathbf{P_s}), \tag{2.2}$$

где

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

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

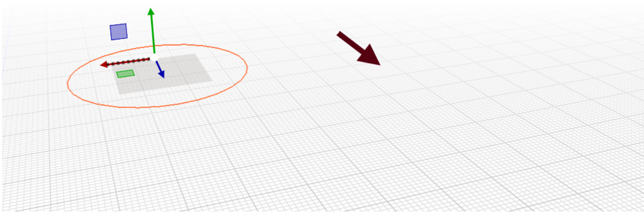

Рисунок 2.3 – Пример построения поверхности вращения

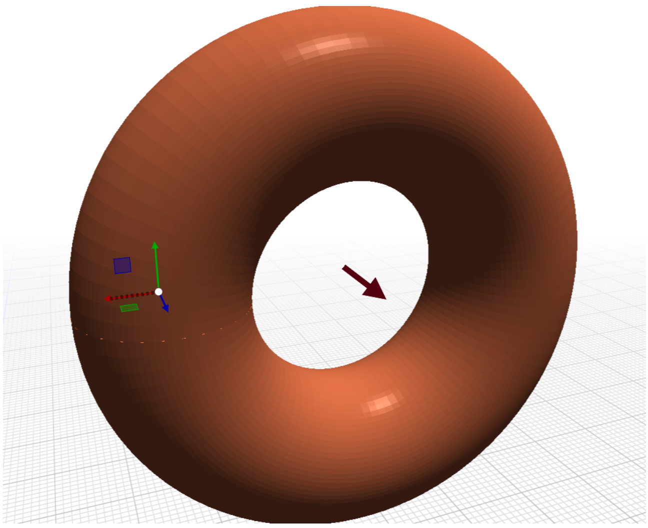

Рисунок 2.4 – Поверхность вращения

2.3 Поверхность сдвига

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

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

можно получить поверхность сдвига.

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

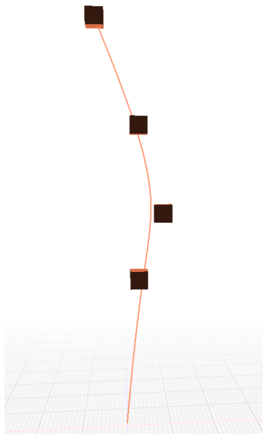

Рисунок 2.5 – Направляющий сплайн

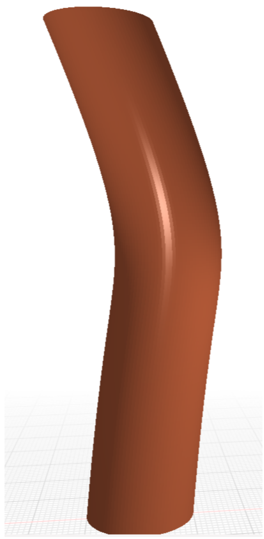

Рисунок 2.6 – Поверхность сдвига

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. Второй $\mathbf{i}_2$ направим ортогонально первому, а $\mathbf{i}_3 = \mathbf{i}_1 \times \mathbf{i}_2$.

Тогда для вычисления радиус-вектора точки кинематической поверхности построим матрицу

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

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 напраляющей. Тогда радиус-вектор точки образующей в подвижной СК при $v = v_{\min}$ равен

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

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 навравляющей 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

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

Таким образом, радиус вектор кинематической поверхности опишем следующей функцией

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

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

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

Так, например, для направляющей, изображенной на Рисунке 2.7, и образующей NURBS окружности получим кинематическую поверхность, изображенную на Рисунке 2.8.

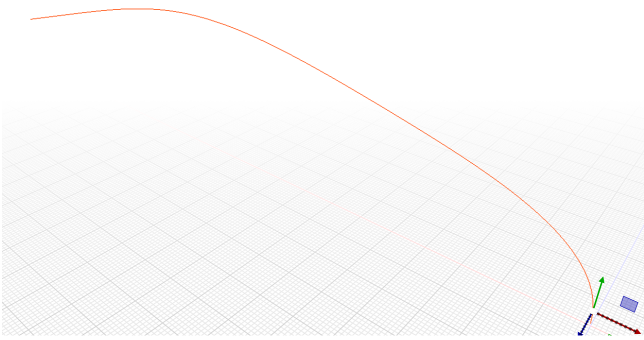

Рисунок 2.7 – Направляющий сплайн

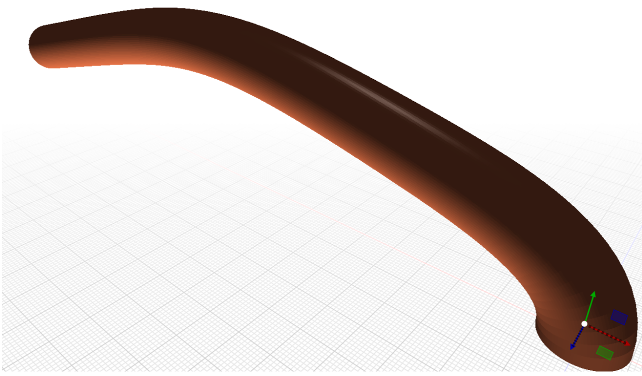

Рисунок 2.8 – Поверхность сдвига

ПРИЛОЖЕНИЕ А
Алгоритмы

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

```
Matrix<float> dersBasisFunc(int i, float u, int p, int n,
                            std::vector<float> U) {
  Matrix<float> ders(n + 1, p + 1);
  Matrix<float> ndu(p + 1, p + 1);
  std::vector<float> left(p + 1, 0);
  std::vector<float> right(p + 1, 0);
  Matrix<float> a(2, p + 1);
  ndu[0][0] = 1.0f;
  for (auto j = 1; j <= p; j++) {
    left[j] = u - U[i + 1 - j];
    right[j] = U[i + j] - u;
    float saved = 0.0f;
    for (auto r = 0; r < j; r++) {
      ndu[j][r] = right[r + 1] + left[j - r];
      float temp = ndu[r][j - 1] / ndu[j][r];
      ndu[r][j] = saved + right[r + 1] * temp;
      saved = left[j - r] * temp;
    }
    ndu[j][j] = saved;
  }
  for (auto j = 0; j <= p; j++) {
    ders[0][j] = ndu[j][p];
  }
  for (auto r = 0; r <= p; r++) {
    int s1 = 0;
    int s2 = 1;
    a[0][0] = 1.0f;
    for (auto k = 1; k <= n; k++) {
      float d = 0.0f;
      int rk = r - k;
      int pk = p - k;
      if (r >= k) {
        a[s2][0] = a[s1][0] / ndu[pk + 1][rk];
        d = a[s2][0] * ndu[rk][pk];
      }
      int j1 = rk >= -1 ? 1 : -rk;
```

```
      int j2 = r - 1 <= pk ? k - 1 : p - r;
      for (auto j = j1; j <= j2; j++) {
        a[s2][j] = (a[s1][j] - a[s1][j - 1]) / ndu[pk + 1][rk + j];
        d += a[s2][j] * ndu[rk + j][pk];
      }
      if (r <= pk) {
        a[s2][k] = -a[s1][k - 1] / ndu[pk + 1][r];
        d += a[s2][k] * ndu[r][pk];
      }
      ders[k][r] = d;
      int j = s1;
      s1 = s2;
      s2 = j;
    }
  }
  int r = p;
  for (auto k = 1; k <= n; k++) {
    for (auto j = 0; j < p; j++) {
      ders[k][j] *= r;
    }
    r *= p - k;
  }
  return ders;
}
```