Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа по курсу «Компьютерная графика»
Тема: Составная Бета-сплайновая поверхность

Студент:	К. М. Воронов
Преподаватель:	А. В. Морозов
Группа:	М8О-307Б-19
Дата:	
Оценка:	
Подпись:	

Москва, 2022

Составная Бета-сплайновая поверхность

Задача: Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и переключаются из графического интерфейса.

Должна быть обеспечена возможность тестирования программы на различных наборах подготовленных исходных данных и их изменение. 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 или буффер вершин Open GL. Реализовать возможность отображения опорных точек, направляющих и других данных по которым формируется порция поверхности и отключения каркасной визуализации

Описание

Бета-сплайновые поверхности

Элементарная Бета-сплайновая поверхность по заданному массиву точек

$$\begin{matrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{matrix}$$

Определяется при помощи векторного уравнения:

$$R^{(i,j)}(u,v) = \sum_{i=0}^{3}\sum_{j=0}^{3} b_i(u)b_j(v)P_{i,j}, 0 \le u, v \le 1,$$

Фунцкиональные коэффициенты $b_i(u)$ и $b_j(v)$ в котором задаются следующими формулами:

$$b_0(u) = \tfrac{2\beta_1^3}{\delta}(1-u)^3, \quad b_3(u) = \tfrac{2u^3}{\delta}$$

$$b_1(u) = \tfrac{1}{\delta}(2\beta_1^3 u(u^2-3u+3) + 2\beta_1^2(u^3-3u^2+2) + 2\beta_1(u^3-3u+2) + \beta_2(2u^3-3u^2+1))$$

$$b_2(u) = \tfrac{1}{\delta}(2\beta_1^2 u^2(-u+3) + 2\beta_1 u(-u^2+3) + \beta_2 u^2(-2u+3) + 2(-u^3+1))$$

(формулы для многочленов $b_j(v)$ отличаются от приведенных только тем, что всюду вместо переменной u стоит переменная v), где $\beta_1 > 0$ и $\beta_2 \ge 0$ и $\delta = 2\beta_1^3 + 4\beta_1^2 + 4\beta_1 + \beta_2 + 2$. Числовые параметры β_1 и β_2 называются **параметрами формы Бета-сплайновой поверхности**.

Составной Бета-сплайновой поверхностью, определяемой массиовом из $(m+1)(n+1)$ вершин

$$P = \{P_{ij}, i = 0,1,....,m,\ j = 0,1,...,n\} \quad m \ge 3, n \ge 3,$$

называется поверхность S, которую можно представить в виде объединения $(m-2)(n-2)$ элементарных Бета-сплайновых поверхностей $S^{(1,1)}, ..., S^{(m-2,n-2)}$,

$$S = \cup_{i=1}^{m-2} \cup_{j=1}^{n-2} S^{(i,j)};$$

Освещение

Освещение состоит из трех составляющих: фоновой, рассеивающей и зеркальной:

$$I = I_a + I_d + I_s$$

Фоновая составляющая
Фоновое освещение определяется как постоянная в каждой точке величина - надбавка к освещению (как бы влияние окружащией обстановки). Расчитывается оно по следующей формуле:

$$I_a = i_a * k_a,$$

где i_a - мощность фонового освещения, а k_a - свойство материала воспринимать фоновое освещение. Как видно, фоновая составляющая не зависит от пространственных координат освещаемой точки и источника.

Рассеивающая составляющая
Диффузное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный свет рассеивается равномерно по всем направлениям.
Свет точечного источника отражается от идеального рассеивателя по закону косинусов Ламберта: интенсивность отраженного света пропорциональна косинусу угла между направлением света и нормалью к поверхности. Для удобства все векторы, описанные ниже, берутся единичными. В этом случае косинус угла между ними совпадает со скалярным произведением:

$$I_d = \frac{k_d * i_l}{K + d} * cos(\vec{L}, \vec{N}),$$

где k_d - свойство материала воспринимать рассеянное освещение, i_l - интенсивность точеченого источника, L - направление из точки на источник света, N - вектор нормали в точке, K и d - коэффициенты пропорциональности интенсивности и расстояния до источника света.

Зеркальная составляющая Интенсивность зеркально отраженного света зависит от угла падения, длины волны падающего света и свойств вещества отражающей поверхности. Зеркальное отражение света является направленным.

Так как физические свойства зеркального отражения очень сложны, будем пользоваться эмпирической моделью Буи-Туонга Фонга:

$$I_s = i_l * w(\vec{L} \hat{} \vec{N}, l) * cos^p(\vec{R}, \vec{S}),$$

Заменив функцию w угла падения $\vec{L}\hat{}\vec{N}$ и длины волны l константой k_s и добавив те же коэффициенты пропорциональности интенсивности и расстояния до источника света. В итоге составляющая рассчитывается следующим образом:

$$I_s = \frac{i_l * k_s}{d+K} * cos^p(\vec{R}, \vec{S}),$$

где $\vec{R}$ - отражённый луч, а $\vec{S}$ - вектор наблюдения.

Исходный код

Основные фрагменты кода.

Функция генерации элементарной Бета-сплайновой поверхности

```
void Elementary_beta_spline(float du, float dv, float delta, int k1, int k2, int p)
{
    float u = 0;
    float v = 0;

    for (int t1 = 0; t1 <= (int)_u.Value; ++t1)
    {
        for (int t2 = 0; t2 <= (int)_v.Value; ++t2)
        {
            Vector3 res = new Vector3(0, 0, 0);
            for (int i = k1; i < k1 + 4; ++i)
            {
                float b_i = coeffB(i - k1, u, delta);
                for (int j = k2; j < k2 + 4; ++j)
                {
                    float b_j = coeffB(j - k2, v, delta);

                    res.X += b_i * b_j * Verticies[i][j].Point.X;
                    res.Y += b_i * b_j * Verticies[i][j].Point.Y;
                    res.Z += b_i * b_j * Verticies[i][j].Point.Z;
                }
            }
            dr.Add(new Vertex(res.X, res.Y, res.Z));

            dr[dr.Count - 1].Id = (uint) dr.Count - 1;

             v += dv;
        }

        v = 0;
        u += du;
    }

    p = p * ((int)_u.Value + 1) * ((int)_v.Value + 1);

    for (int i = 0; i < (int) _u.Value; ++i)
    {
        for (int j = 0; j < (int) _v.Value; ++j)
        {
            Polygons.Add(new Polygon());
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + p
                ]);
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + 1
                + p]);
```

```
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + (int)_v.Value + 1 + p]);

            dr[i * ((int)_v.Value + 1) + j + p].polygons.Add(Polygons[Polygons.Count - 1]);
            dr[i * ((int)_v.Value + 1) + j + 1 + p].polygons.Add(Polygons[Polygons.Count - 1]);
            dr[i * ((int)_v.Value + 1) + j + ((int)_v.Value + 1) + p].polygons.Add(Polygons[Polygons.Count - 1]);

            Polygons.Add(new Polygon());
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + 1 + p]);
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + 1 + (int)_v.Value + 1 + p]);
            Polygons[Polygons.Count - 1].points.Add(dr[i * ((int)_v.Value + 1) + j + (int)_v.Value + 1 + p]);

            dr[i * ((int)_v.Value + 1) + j + 1 + p].polygons.Add(Polygons[Polygons.Count - 1]);
            dr[i * ((int)_v.Value + 1) + 1 + j +((int)_v.Value + 1) + p].polygons.Add(Polygons[Polygons.Count - 1]);
            dr[i * ((int)_v.Value + 1) + j + ((int)_v.Value + 1) + p].polygons.Add(Polygons[Polygons.Count - 1]);
        }
    }
}
```

Генерация Составной Бета-сплайновой поверхности, подсчёт нормалей, подготовка к загрузке в буффер.

```
void Figure()
{
    dr = new List<Vertex>();

    float du = (float) (1f / _u.Value);
    float dv = (float) (1f / _v.Value);

    float delta = 2 * (float) _beta1.Value * (float) _beta1.Value * (float) _beta1.Value + 4 * (float) _beta1.Value * (float) _beta1.Value + 4 * (float) _beta1.Value + (float) _beta2.Value + 2;

    Polygons = new List<Polygon>();

    int p = 0;
    for (int i = 0; i < Verticies.Count - 3; ++i)
    {
        for (int j = 0; j < Verticies[i].Count - 3; ++j)
        { Elementary_beta_spline(du, dv, delta, i, j, p);
            p += 1;
```

```
        }
    }


    foreach (var pol in Polygons)
    {
        Normals(pol);
    }

    for (int i = 0; i < dr.Count; ++i)
    {
        Vector3 n = new Vector3(0, 0, 0);
        for (int l = 0; l < dr[i].polygons.Count; ++l)
        {
            n += dr[i].polygons[l].NormalInWorldSpace;
        }

        n.X = n.X / dr[i].polygons.Count;
        n.Y = n.Y / dr[i].polygons.Count;
        n.Z = n.Z / dr[i].polygons.Count;
        n = Vector3.Normalize(n);
        dr[i].NormalInWorldSpace = n;
    }

    masver = new float[dr.Count * 6];
    int k = 0;
    for (int i = 0; i < dr.Count; ++i)
    {
        masver[k] = dr[i].Point.X;
        k += 1;
        masver[k] = dr[i].Point.Y;
        k += 1;
        masver[k] = dr[i].Point.Z;
        k += 1;
    }

    for (int i = 0; i < dr.Count; ++i)
    {
        masver[k] = dr[i].NormalInWorldSpace.X;
        k += 1;
        masver[k] = dr[i].NormalInWorldSpace.Y;
        k += 1;
        masver[k] = dr[i].NormalInWorldSpace.Z;
        k += 1;
    }

    masid = new uint[Polygons.Count * 3];
    k = 0;
```

```
    foreach (var pol in Polygons)
    {
        for (int i = 0; i < pol.points.Count; ++i)
        {
            masid[k] = pol.points[i].Id;
            k += 1;
        }
    }
}
```

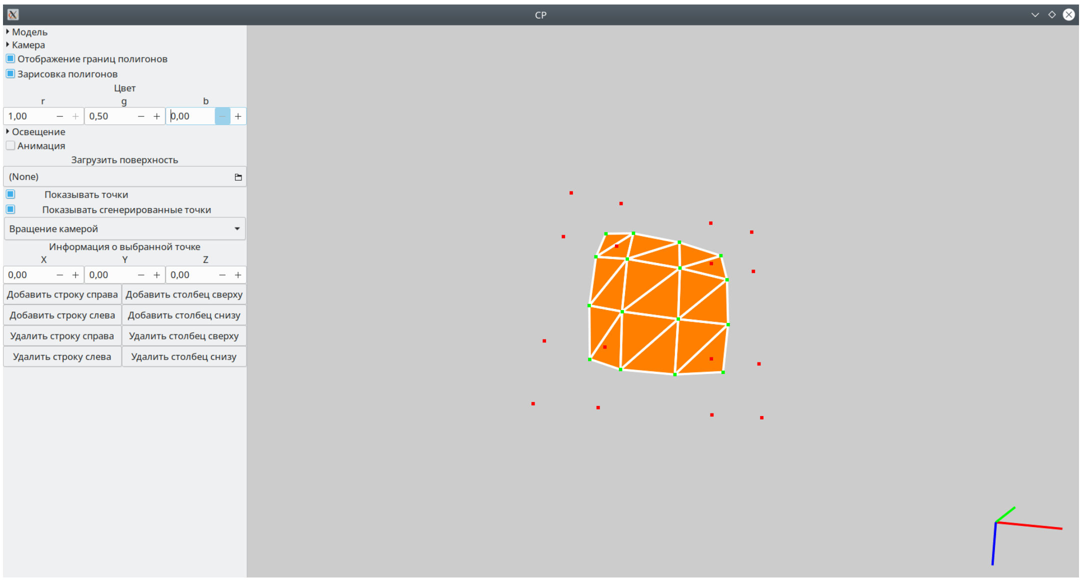
Фрагментный шейдер для освещения

```
#version 330 core

struct Material {
    vec3 ka;
    vec3 kd;
    vec3 ks;
    float p;
};

struct Light {
    vec3 ia;
    vec3 il;
    vec3 position;
};

in vec3 normalnf;
in vec3 fragCoord;

out vec4 color;

uniform mat4 view4f;

uniform float k;
uniform vec3 c;
uniform vec3 camera;
uniform Material m;
uniform Light l;

uniform mat4 proj4f;

void main() {
    vec3 position = vec3(view4f * vec4(l.position, 1) );

    vec3 background = m.ka * l.ia;
    vec3 diffuse = m.kd * l.il;

    vec3 toLight = normalize(position - fragCoord);
    diffuse *= max(dot(toLight, normalnf), 0);
```

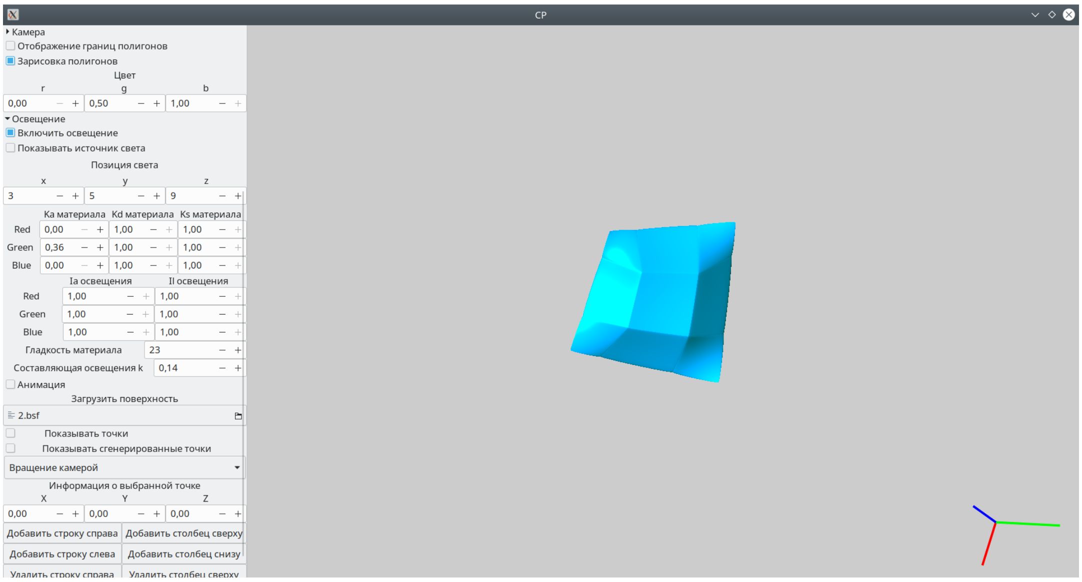
```

    vec3 specular = m.ks * l.il;

    if (dot(toLight, normalnf) > 1e-3) {
        vec3 r = reflect(-toLight, normalnf);
        specular *= pow(max(dot(r, normalize(camera - fragCoord)), 0), m.p);
    } else {
        specular *= 0;
    }

   color = vec4(c * background + (c * diffuse + c * specular) / (k + length(position -
       fragCoord) / 20), 1);
}
```

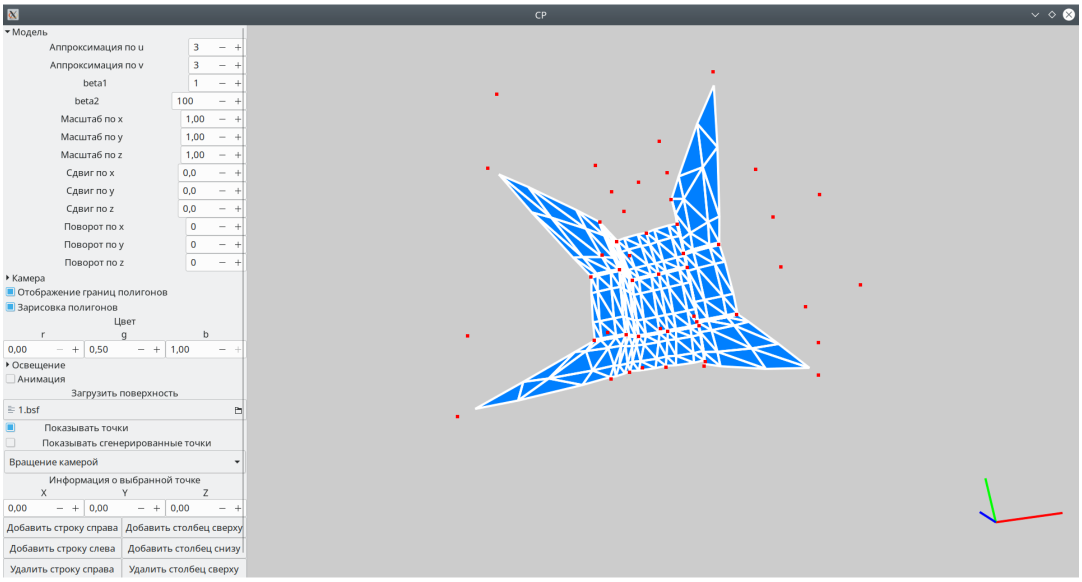
Скриншоты работы программы

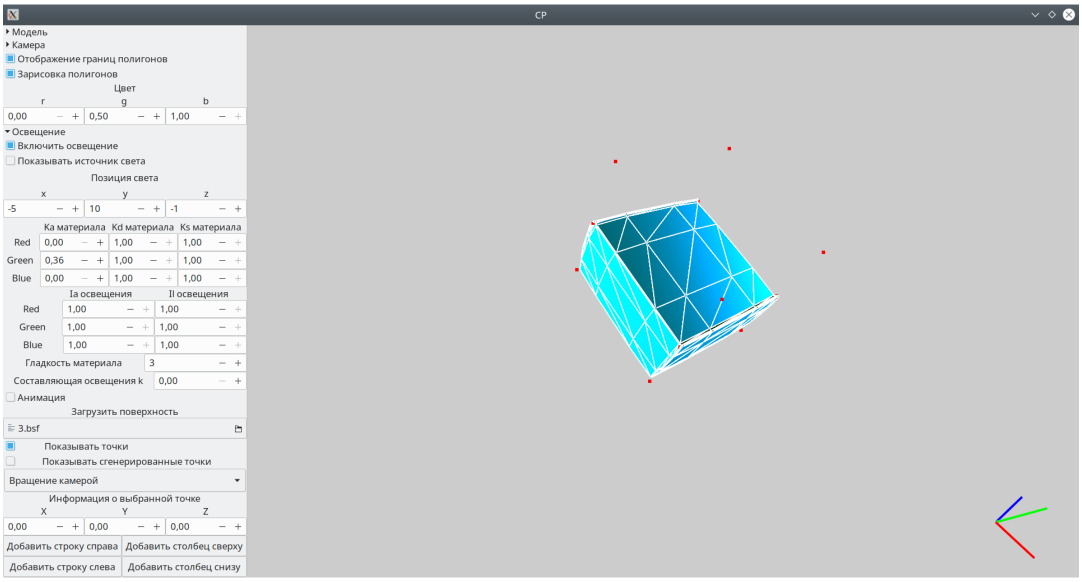
Элементарная Бета-сплайновая поверхность



Составные Бета-сплайновые поверхности




CP
Модель
Аппроксимация по u 3
Аппроксимация по v 3
beta1 1
beta2 100
Масштаб по x 1,00
Масштаб по y 1,00
Масштаб по z 1,00
Сдвиг по x 0,0
Сдвиг по y 0,0
Сдвиг по z 0,0
Поворот по x 0
Поворот по y 0
Поворот по z 0
Камера
Отображение границ полигонов
Зарисовка полигонов
Цвет
r g b
0,00 0,50 1,00
Освещение
Анимация
Загрузить поверхность
1.bsf
Показывать точки
Показывать сгенерированные точки
Вращение камерой
Информация о выбранной точке
X Y Z
0,00 0,00 0,00
Добавить строку справа
Добавить столбец сверху
Добавить строку слева
Добавить столбец снизу
Удалить строку справа
Удалить столбец сверху


CP
Модель
Камера
Отображение границ полигонов
Зарисовка полигонов
Цвет
r g b
0,00 0,50 1,00
Освещение
Включить освещение
Показывать источник света
Позиция света
x y z
-5 10 -1
Ka материала Kd материала Ks материала
Red 0,00 1,00 1,00
Green 0,36 1,00 1,00
Blue 0,00 1,00 1,00
Ia освещения Il освещения
Red 1,00 1,00
Green 1,00 1,00
Blue 1,00 1,00
Гладкость материала 3
Составляющая освещения k 0,00
Анимация
Загрузить поверхность
3.bsf
Показывать точки
Показывать сгенерированные точки
Вращение камерой
Информация о выбранной точке
X Y Z
0,00 0,00 0,00
Добавить строку справа
Добавить столбец сверху
Добавить строку слева
Добавить столбец снизу

Список литературы

[1] Шикин Е.В., Плис Л.И.Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользователей. М.: ДИАЛОГ-МИФИ, 1996 г. страницы 191, 194.

[2] Материалы Морозова Александра Валерьевича, Московский авиационный институт.