

**Московский авиационный институт
(Национальный исследовательский университет)**

Факультет: «Информационные технологии и прикладная математика»

Кафедра: 806 «Вычислительная математика и программирование»

Курсовая работа
по «Компьютерной графике»
Тема: «Линейчатая поверхность Кунса»

Студент: Мариничев И. А.

Группа: М8О-308Б-19

Преподаватель: Филиппов Г. С.

Оценка:

Москва
2021

1. Постановка задачи.

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Исходные данные готовятся самостоятельно и вводятся из файла или в панели ввода данных. Должна быть обеспечена возможность тестирования программы на различных наборах исходных данных. Программа должна обеспечивать выполнение аффинных преобразований для заданной порции поверхности, а также возможность управлять количеством изображаемых параметрических линий. Для визуализации параметрических линий поверхности разрешается использовать только функции отрисовки отрезков в экранных координатах.

Вариант №8: Линейчатая поверхность Кунса (границы – кривые Безье 3D 2-й степени)

2. Описание программы.

Рассмотрим способ построения поверхности $Q(u, w)$. Пусть известны четыре граничные кривые $B(u, 0), B(u, 1), B(0, w), B(1, w)$. Тогда билинейная смешивающая функция равна:

$$Q(u, w) = B(u, 0)(1 - w) + B(u, 1)w + B(0, w)(1 - u) + B(1, w)u.$$

Однако, проверив этот результат в угловых точках куска поверхности, например:

$$Q(0, 0) = B(0, 0) + B(0, 0) = 2B(0, 0)$$

и на границах:

$$Q(0, w) = B(0, 0)(1 - w) + B(0, 1)w + B(0, w)$$

Так как угловые точки учитываются дважды, правильный результат можно получить только с помощью вычитания дополнительных членов, возникающих из-за удвоения угловых точек:

$$\begin{aligned} Q(u, w) = & B(u, 0)(1 - w) + B(u, 1)w + B(0, w)(1 - u) + B(1, w)u \\ & - B(0, 0)(1 - u)(1 - w) - B(0, 1)(1 - u)w - B(1, 0)u(1 - w) \\ & - B(1, 1)uw. \end{aligned}$$

Граничными кривыми в моём варианте являются кривые Безье 2-й степени, они строятся в соответствии со следующей формулой:

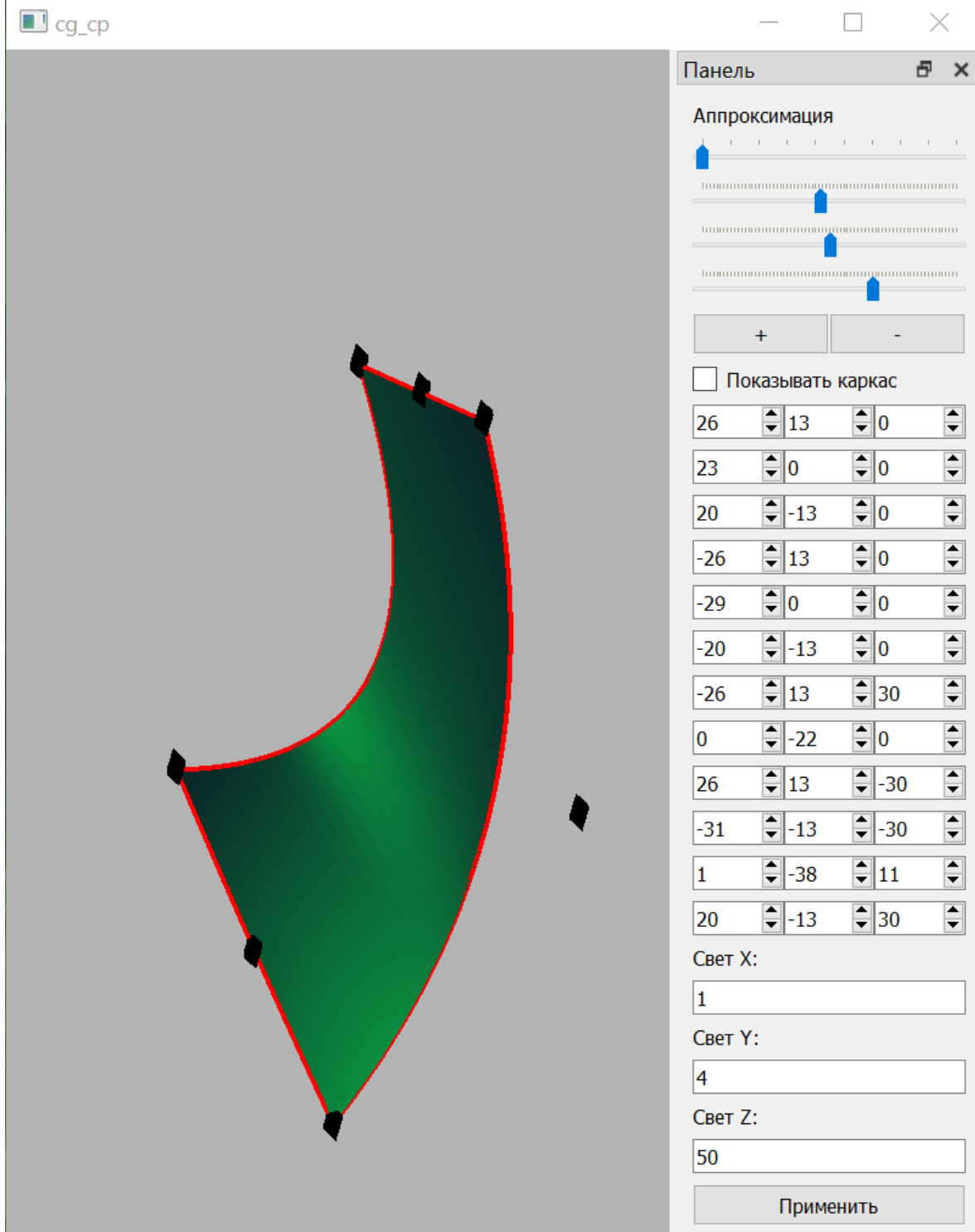
$$B(t) = (1 - t)^2 P_0 + 2t(1 - t)P_1 + t^2 P_2,$$

где P_0, P_1, P_2 — опорные точки, $0 \leq t \leq 1$.

Для решения задачи я использовал C++, фреймворк Qt и OpenGL.

3. Демонстрация работы программы.

1) Вид поверхности при высокой аппроксимации, заданными материалами и наличии света на сцене.



The screenshot displays the 'cg_cp' software interface. The main window shows a 3D perspective view of a rectangular frame structure. The frame is composed of a red outline and several horizontal black lines. The structure is tilted, and its vertices are marked with black cubes. The background is a light gray.

On the right side, there is a control panel titled 'Панель'. It includes a section for 'Аппроксимация' (Approximation) with four horizontal sliders and two buttons labeled '+' and '-'. Below this is a checkbox labeled 'Показывать каркас' (Show skeleton), which is checked. Underneath the checkbox is a table of numerical values for the skeleton, organized in four rows and three columns. Each cell contains a number and a small up/down arrow icon. The values are: Row 1: 26, 13, 0; Row 2: 23, 0, 0; Row 3: 20, -13, 0; Row 4: -26, 13, 0; Row 5: -23, 0, 0; Row 6: -20, -13, 0; Row 7: -26, 13, 0; Row 8: 0, 13, 0; Row 9: 26, 13, 0; Row 10: -20, -13, 0; Row 11: 0, -13, 0; Row 12: 20, -13, 0. Below the table are three input fields labeled 'Свет X:', 'Свет Y:', and 'Свет Z:'. At the bottom of the panel is a button labeled 'Применить' (Apply).

4. Основной код программы.

1) Метод для отрисовки кривой Безье.

```
void display::drawBezierCurve(const std::vector<QVector4D> &points) {
    glColor3f(1.f, 0.f, 0.f);

    float prevX = points[0].x() + SQUARE_SIZE / 2, prevY = points[0].y() -
    SQUARE_SIZE / 2;
    float prevZ = points[0].z();
    glLineWidth(4.f);
    glBegin(GL_LINE_STRIP);
    glVertex3f(prevX, prevY, prevZ);
    for (double t = static_cast<double>(step); t < 1.; t +=
    static_cast<double>(step)) {
        float x = static_cast<float>(std::pow((1. - t), 2.)) * (points[0].x()
+ SQUARE_SIZE / 2) +
            2.f * static_cast<float>(t) * static_cast<float>(1. - t) *
            (points[1].x() + SQUARE_SIZE / 2) +
            static_cast<float>(std::pow(t, 2.)) * (points[2].x() +
SQUARE_SIZE / 2);
        float y = static_cast<float>(std::pow((1. - t), 2.)) * (points[0].y()
- SQUARE_SIZE / 2) +
            2.f * static_cast<float>(t) * static_cast<float>(1. - t) *
            (points[1].y() - SQUARE_SIZE / 2) +
            static_cast<float>(std::pow(t, 2.)) * (points[2].y() -
SQUARE_SIZE / 2);
        float z = static_cast<float>(std::pow((1. - t), 2.)) *
            (points[0].z()) +
            2.f * static_cast<float>(t) * static_cast<float>(1. - t) *
            (points[1].z()) +
            static_cast<float>(std::pow(t, 2.)) * (points[2].z());
        glVertex3f(x, y, z);
        prevX = x;
        prevY = y;
        prevZ = z;
        if (t + static_cast<double>(step) >= 1.) {
            x = points[2].x() + SQUARE_SIZE / 2;
            y = points[2].y() - SQUARE_SIZE / 2;
            z = points[2].z();
            glVertex3f(x, y, z);
        }
    }
    glEnd();
}
```

2) Методы для отрисовки точек линейчатой поверхности.

```
float display::calcX(double t, double v) {
    double x = (1 - v) * (std::pow(1 - t, 2) * static_cast<double>(points1[0].x()
+ SQUARE_SIZE / 2) +
        2 * t * (1 - t) * static_cast<double>(points1[1].x() +
SQUARE_SIZE / 2) +
        std::pow(t, 2) * static_cast<double>(points1[2].x() + SQUARE_SIZE
/ 2)) +
        v * (std::pow(1 - t, 2) * static_cast<double>(points2[0].x() +
SQUARE_SIZE / 2) +
        2 * t * (1 - t) * static_cast<double>(points2[1].x() +
SQUARE_SIZE / 2) +
        std::pow(t, 2) * static_cast<double>(points2[2].x() + SQUARE_SIZE
/ 2));
    return static_cast<float>(x);
}

float display::calcY(double t, double v) {
```

```

        double y = (1 - v) * (std::pow(1 - t, 2) * static_cast<double>(points1[0].y()
- SQUARE_SIZE / 2) +
        2 * t * (1 - t) * static_cast<double>(points1[1].y() -
SQUARE_SIZE / 2) +
        std::pow(t, 2) * static_cast<double>(points1[2].y() - SQUARE_SIZE
/ 2)) +
        v * (std::pow(1 - t, 2) * static_cast<double>(points2[0].y() -
SQUARE_SIZE / 2) +
        2 * t * (1 - t) * static_cast<double>(points2[1].y() -
SQUARE_SIZE / 2) +
        std::pow(t, 2) * static_cast<double>(points2[2].y() - SQUARE_SIZE
/ 2));
    return static_cast<float>(y);
}

float display::calcZ(double t, double v) {
    double z = (1 - v) * (std::pow(1 - t, 2) *
static_cast<double>(points1[0].z()) +
    2 * t * (1 - t) * static_cast<double>(points1[1].z()) +
    std::pow(t, 2) * static_cast<double>(points1[2].z())) +
    v * (std::pow(1 - t, 2) * static_cast<double>(points2[0].z()) +
    2 * t * (1 - t) * static_cast<double>(points2[1].z()) +
    std::pow(t, 2) * static_cast<double>(points2[2].z()));
    return static_cast<float>(z);
}

```

5. Выводы.

В ходе выполнения данной курсовой работы была реализована программа, позволяющая моделировать линейчатую поверхность Кунса, в качестве границ которой выступают кривые Безье 2-й степени. Для выполнения данного задания мнегодились, навыки, полученные в 4, 5 и 7 лабораторных работах.

6. Список литературы.

1. Д. Роджерс, Дж. Адамс. Математические основы машинной графики, – М.: из-во “Мир”, 2001.
2. Голованов Н.Н. Геометрическое моделирование. – М.: Издательство физико-математической литературы, 2002.
3. Шишкин Е.В., Плис А.И. Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользователя.