Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовая работа по курсу «Компьютерная графика»
Каркасная визуализация заданной поверхности

Студент: Черемисинов М. Л.
Преподаватель: Филиппов Г. С.
Группа: М8О-308Б-18
Вариант: 10
Дата:
Оценка:
Подпись:

Москва, 2020

Условие

Составить и отладить программу, обеспечивающую каркасную визуализацию порции поверхности заданного типа. Должна быть предусмотрена возможность изменения точности аппроксимации и параметров поверхности.

Вариант: 10

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

Основные определения

Кинематическая поверхность. Образующая и направляющая.

Существует множество способов описания поверхностей: явное представление в виде функции $z = f(x, y)$, уравнения $F(x, y, z) = 0$, параметрического представления радиусом-вектором $r(u, v)$, а также другие.

Существует также способ построения поверхности, при котором она представляется как совокупность положений некоторой линии, перемещающейся в пространстве по определенным правилам. Поверхность, построенную подобным способом, принято называть "кинематической". При её построении используются два понятия: *образующая* и *направляющая*.

Образующей называют линию, движущуюся в пространстве по определенным правилам (возможно вдоль какой-либо кривой).
Направляющей называют линию, вдоль которой происходит движение образующей кривой.

В результате подобного описания поверхности через две кривые появляется возможность строить тела достаточно сложной формы.

Кривая Безье и кардиоида.

В данной работе в качестве направляющей будет рассматриваться кривая Безье 3 порядка, а в качестве образующей - кардиоида.

Кривые Безье начали использоваться в 60-х годах XX века при проектировании автомобилей.

Сама кривая описывается следующей формулой:

$$B(t) = \sum_{k=0}^{n} P_k b_{k,n}(t), 0 \leq t \leq 1$$

где P_k - ключевые точки, для которых строится кривая, а $b_{k,n}(t)$ - базисные функции кривой, определяющиеся по следующему правилу:

$$b_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

В случае двух точек кривая Безье представляет собой соединяющий их отрезок, в случае трех и более - кривую, плавно перетекающую из первой точки в последнюю.

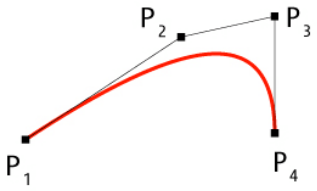


Рис. 1: Кривая Безье для 4-х контрольных точек

Кардиоида представляет собой кривую, полученную при отслеживании положения заранее заданной точки на одной окружности, вращающейся по поверхности другой.

В математическом виде кардиоиду можно описать через уравнение:

$$(x^2 + y^2 + 2ax)^2 - 4a^2(x^2 + y^2) = 0$$

или в полярных координатах

$$r = 2a(1 - cos\varphi)$$

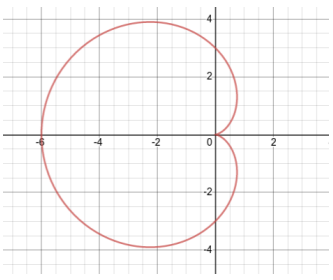


Рис. 2: Кардиоида при $a = 1.5$

Описание проекта

Для реализации программы был выбран язык $C++$ и opensource-библиотек $OpenGL$, $glad$, $glfw$, glm и $ImGui$ для работы с графикой, окнами и математикой.

Составляющие проекта можно условно разделить на две части: пользовательский интерфейс и отображение фигуры.

В первом окне пользователь может изменять различные параметры поверхности: положение ключевых точек, цвет, стиль отрисовки, параметры освещения и прочее. Во втором окне происходит непосредственная демонстрация поверхности.

Структура проекта

Проект имеет следующую структуру:

- *main.cpp* - создание окон и работа с ними, обработка пользовательского ввода
- *curves_math.hpp* - расчёт координат точек кривой Безье
- *math_help.hpp* - функции и константы для генерации фигуры
- *shaders/* - вершинные и фрагментные шейдеры
  - *shader.vert*
  - *shader.frag*
  - *lightShader.vert*
  - *lightShader.frag*
- *imgui/* - библиотека для работы с пользовательским интерфейсом
  - *fonts/*
    - *ProggyClean.ttf* - шрифт для пользовательского интерфейса
  - $*.cpp$
  - $*.h$
- *CMakeLists.txt* - инструкции для сборки проекта

Описание работы

Выделим основные этапы работы программы:

1. Создание окон при помощи библиотеки $glfw$, инициализация их свойств: размера, положения и др.
2. Загрузка функций $OpenGL$ при помощи библиотеки $glad$
3. Загрузка и инициализация шейдеров
4. Инициализация пользовательского интерфейса
5. Расчёт начальных значений фигуры и источника света
6. Работа основного цикла:
   (a) Получение действий от пользователя (изменение параметров фигуры, нажатие клавиш)
   (b) Расчёт новых координат точек поверхности и источника света (если требуется)
   (c) Передача uniform-переменных из основной программы в шейдеры
   (d) Отрисовка поверхности и источника света
   (e) Отрисовка пользовательского интерфейса

Приведем демонстрацию кода программы для наиболее значимых её частей.

Нахождение точек поверхности начинается с вычисления направляющей кривой Безье.

```
1  //calculates bezier curve with O(n^2*t) time and O(n*t) memory
2  void calculateCurve(const std::vector<float> &t) {
3      std::vector<std::vector<point3>> vec(keyPoints.size() - 1,
4                                                  std::vector<point3>(t.size()));
5      for (size_t i = 0; i < vec.size(); i++) {
6          calculateBezierTwoPoints(keyPoints[i], keyPoints[i + 1], vec[i], t);
7      }
8
9      for (size_t last = vec.size() - 1; last > 0; last--) {
10         for (size_t i = 0; i < last; i++) {
11             for (size_t j = 0; j < t.size(); j++) {
12                 vec[i][j] = (1 - t[j]) * vec[i][j] + t[j] * vec[i + 1][j];
13             }
14         }
15     }
16
17     this->points = vec[0];
```

```
}

//calculates bezier curve for two points
static void calculateBezierTwoPoints(const point3 &left, const point3 &right,
                                     std::vector<point3> &dest, const std::vector<float> &
                                         t) {
    for (size_t i = 0; i < t.size(); i++) {
        dest[i] = (1 - t[i]) * left + t[i] * right;
    }
}
```

Вычисление точек кардиоиды происходит следующим образом:

1. Вычисляем параметр φ в пределах $[0; 2\pi]$
2. Вычисляем значение радиус-вектора r
3. Переводим полярные координаты в декартовы

```
std::vector<crv::point3> cardioid(float a, int precision) {
    std::vector<crv::point3> res(precision);

    std::vector<float> phi = math::linspace(0, 2.0f * math::pi, precision);
    for (int i = 0; i < phi.size(); i++) {
        float r = 2.0f * a * (1.0f - std::cos(phi[i]));
        res[i] = {0.0f, r * std::sin(phi[i]), r * std::cos(phi[i])};
    }

    return res;
}
```

Вычисление точек поверхности можно разбить на данные этапы:

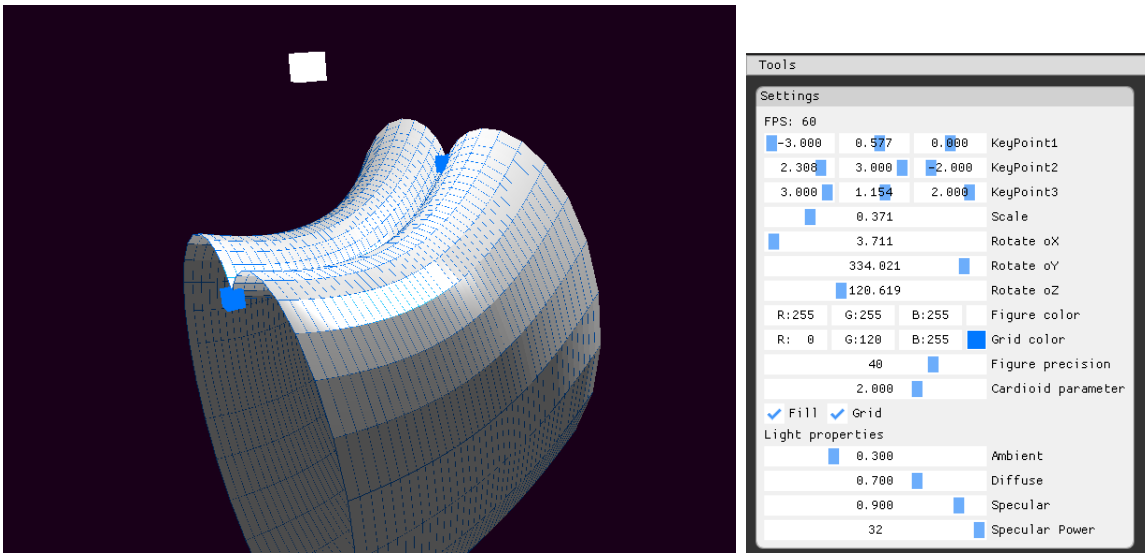
1. Вычисление направляющей кривой Безье
2. Вычисление стандартной кардиоиды
3. Проход по точкам кривой Безье и добавление их значений к каждой точке кривой Безье

```
//returns vector of triangles and vector of indices
std::pair<std::vector<float>, std::vector<unsigned>> customFigure(const std::vector<
    crv::point3> &keyPoints, int precision) {

    std::vector<float> triangles;

    // calculate bezier curve
    crv::Bezier bezier;
```

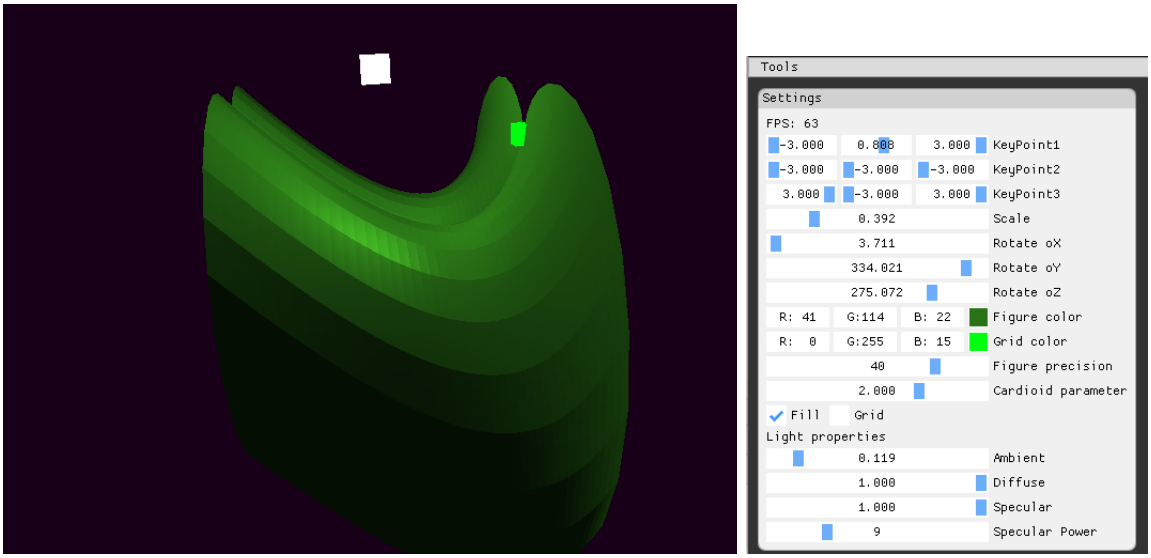
```
    bezier.setPrecision(precision);
    bezier.setKeyPoints(keyPoints);
    bezier.calculateCurve();

    // calculate base cardioid
    std::vector<crv::point3> cardi = cardioid(cardioidMainValue, precision);


    std::vector<crv::point3> prev = cardi;
    for (crv::point3 &p : prev) {
        p += bezier.points[0];
    }

    for (size_t i = 1; i < bezier.points.size(); i++) {
        // move cardioid points to i-th bezier curve point
        std::vector<crv::point3> next = cardi;
        for (crv::point3 &p : next) {
            p += bezier.points[i];
        }

        for (size_t j = 0; j < prev.size(); j++) {
            /* calculate triangle coordinates and normals */
        }

        prev = next;
    }

    // default indices
    // ToDo: may be improved
    std::vector<unsigned> indices(triangles.size() / 6);
    for (size_t i = 0; i < indices.size(); i++) {
        indices[i] = i;
    }

    return {triangles, indices};
}
```

Демонстрация работы

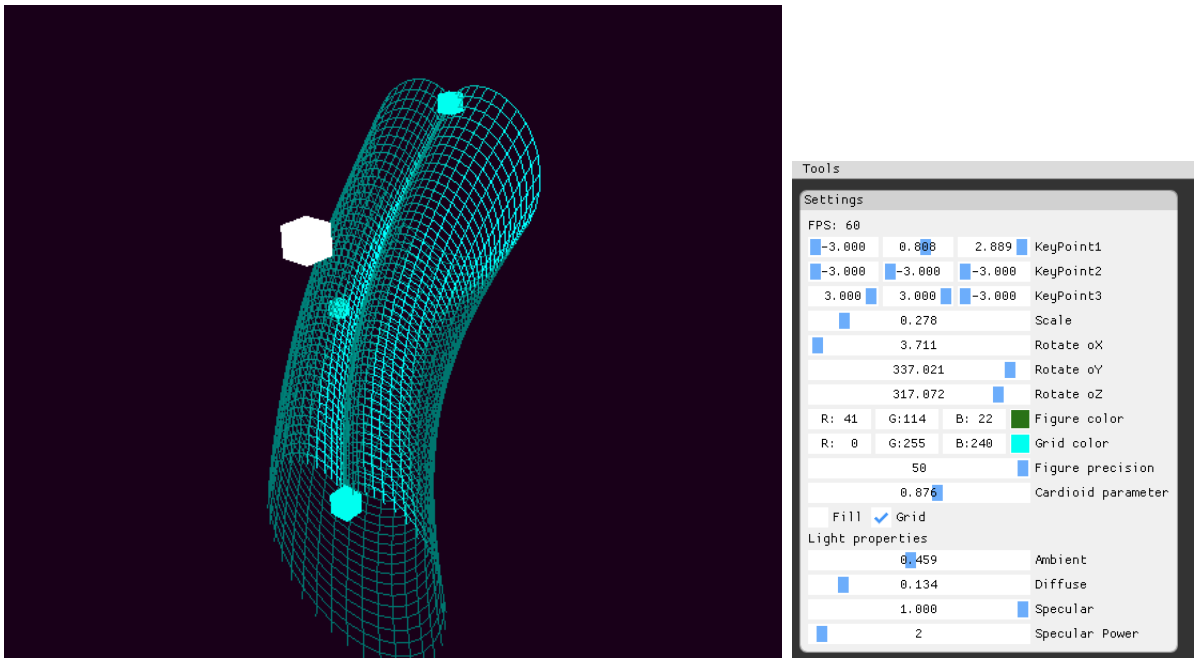
Программа состоит из двух частей: окно с изображением поверхности и окно с пользовательским интерфейсом для настроек. Ключевые точки изображаются кубами. Изменение положения источника света осуществляется при помощи клавиш W, A, S, D, Q, E.



Реализована возможность изменять взаимодействие поверхности со светом, её цвет, положение ключевых точек.



Также возможно изменять параметр кардиоиды и отрисовывать только каркас фигуры.



Возможные улучшения

Для рассчёта координат итоговых треугольников, которые будут передаваться в шейдеры, есть возможность исключить повторяющиеся вершины путём модификации расчёта вектора вершин и индексов, что поспособствует более экономному расходу памяти.

Для хранения одной точки необходимо использовать 6 чисел с плавающей запятой: 3 для её координат и 3 для вектора нормали.

В последней реализации программы для поддержания фигуры с точностью 40 (в кривой Безье и каждой кардиоиде по 40 точек) используется 56160 чисел типа $float$, среди которых уникальных всего лишь $40 * 40 * 6 = 9600$ (17.09%).

Выводы

При помощи средств *OpenGL* была реализована программа, позволяющая моделировать кинематическую поверхность с направляющей в виде кубической кривой Безье

и образующей кардиоидой. Стоит отметить, что предусмотрена возможность строить абсолютно любые поверхности - необходимо лишь изменить функцию генерации треугольников и индексов.

Одним из вариантов дальнейшего развития проекта является добавление координатных осей, сетки, задания направляющей и образующей в диалоговых окнах.