МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по компьютерной графике

Учебно-методическое пособие

Часть 2. Двухмерные кривые и сплайны

2018 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по компьютерной графике. Часть 2. Двухмерные кривые и сплайны. Учебно-методическое пособие. Озерск: ОТИ НИЯУ МИФИ, 2018. — 20 с.

В пособии подробно излагается, как выполнять практические работы по дисциплине «Инженерная и компьютерная графика». Работы второй части включают в себя рисование плоских сплайновых кривых.

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ предлагается изучить основы рисования плоских кривых (графиков), а также сплайновых кривых.

В этой части работ рассматриваются следующие темы:

1) рисование кривых (графиков);
2) непараметрический сплайновый сегмент;
3) параметрический сплайновый сегмент;
4) многосегментная сплайновая кривая;
5) кривая Безье;
6) B-сплайновые кривые.

Основные сведения по рисованию кривых и сплайнам приведены в учебно-методическом пособии автора «Сплайновые кривые и поверхности». Наиболее актуальные версии этих документов доступны в Интернет по адресу http://revol.ponocom.ru.

На выполнение каждой работы предположительно отводится 2 академических часа, однако некоторые, наиболее сложные работы могут потребовать большего времени.

Поэтому, в зависимости от количества учебных часов, выделенных на проведение практических работ, сложности работ и навыков обучающегося, преподаватель может выбирать индивидуальные траектории работ.

Для защиты знаний и навыков, полученных в ходе выполнения практических работ студент должен иметь тетрадь (12-18 листов). Для каждой выполненной работы в тетрадь записывается отчет.

Отчет по работе начинается с заголовка:

1. Фамилия Имя Отчество
2. Группа
3. Дата начала выполнения работы
4. Код работы
5. Название работы
6. Цели работы
7. Задачи работы

При необходимости при выполнении работы в отчет записываются контрольные значения. Во время защиты тетрадь используется для вопросов преподавателя и ответов студента.

Работа КГ-201. Рисование кривых (2 часа)

Цели:
- изучение методов рисования кривых линий (графиков функций).

Задачи:
- выявление размеров области рисования;
- вычисление масштаба;
- вспомогательные функции рисования;
- рисование кривых точками;
- рисование кривых линиями.

1.1. Рабочее пространство

Для выполнения работы используется проект curves, подготовленный преподавателем. Скачайте архив проекта, извлеките из него каталог curves в корневой каталог диска C:. Откройте проект. Установите платформу x86 или Win32, в зависимости от того, что есть. Убедитесь, что проект не содержит ошибок.

В проекте два основных модуля.

Модуль curves.h содержит константы, переменные, и основные функции проекта. Вносить в него изменения не требуется.

Модуль curves.cpp является рабочим.

В модуле определена функция draw, она вызывается автоматически:

```
/* контекст устройства GDC */
void draw() {
    // пример вывода
    MoveToEx(GDC, 300, 0, 0);
    LineTo(GDC, 0, 0);
    LineTo(GDC, 0, 300);
}
```

Комментарий показывает контекст, который используем для вывода. Код рисует в контексте пару прямых, чтобы убедиться в том, что рисовать можно.

1.2. Размеры области вывода

В этой части нужно вычислить размеры области вывода, масштаб изображения и центр системы координат.

Сначала определим константы модуля:

```
/* 2018 ReVoL Primer Template */
/* curves.cpp */
#include "stdafx.h"
#include "curves.h"
// размер графика
#define GX (double)7
#define GY (double)2
// шаг по X
#define DX 5
```

Первые две константы задают размер области вывода в единицах графика. Рисовать будем полную синусоиду, размер которой вдоль оси X равен двум пи, примерно 7 единиц, а полная высота синусоиды составляет две единицы. Шаг по X — это константа, задающая шаг точек вдоль оси X, в которых будет вычисляться значение функции графика при рисовании линиями. Далее описываем переменные, которые нужно вычислить в первую очередь:

```
// размер устройства вывода
int WX = 0, WY = 0;
// масштаб осей
double SX = 0, SY = 0;
// центр системы координат
int CX = 0, CY = 0;
```

Размер устройства вывода — это размер окна для рисования, дескриптор которого hWndGDE, а контекст устройства GDC. Масштаб осей задает преобразования, которые нужно выполнить, чтобы вместить график размером GX и GY в окно для рисования. Переменные CX и CY — это координаты центра системы координат в системе координат устройства вывода. Ниже описываем функцию, которая вычислит эти переменные:

```
/* параметры вывода */
void setview() {
}

/* контекст устройства GDC */
void draw() {
    setview();
}
```

Перейдем в функцию setview. Сначала нужно вычислить размеры клиентской части окна графического вывода (окна для рисования). Для этого нужно описать переменную rc типа RECT и получить в нее размеры при помощи функции GetClientRect.

Далее вычисляем WX и WY (Window X и Y), используя разность полей rc.right и rc.left, и rc.bottom и rc.top.

Затем вычисляем центр системы координат. Мы хотим, чтобы ось X проходила по центру окна по высоте, а ось Y — по левому краю окна, так как мы собираемся рисовать синусоиду. Поэтому принимаем CX равным нулю, а CY — половине высоты окна.

Теперь вычислим масштаб. Это отношение размеров области вывода в единицах графика к размерам окна для рисования или наоборот. Если мы делим GX на WX, то получим единицу измерения «единица графика на пиксель». Если же мы делим WX на GX, то получим единицу измерения «пиксель на единицу графика». Как делить — неважно, главное, чтобы потом правильно использовать эти масштабные коэффициенты. Естественно, точно также нужно поделить GY на WY или наоборот.

1.3. Оси координат

Следующий шаг — нарисовать оси координат. Хотелось бы, чтобы оси координат были нарисованы не основным цветом. Для этого нужно бы создать перья, но не хочется отвлекаться от главной задачи, поэтому будем рисовать тем пером, которое есть.

Чтобы нарисовать одну линию, нужно вызвать две функции. Сначала устанавливаем текущую точку при помощи MoveToEx, затем рисуем линию от текущей точки до нужной точки при помощи LineTo. Учитывая, что координаты центра системы координат заданы CX и CY, проще всего нарисовать координатные оси, используя систему координат устройства вывода. Тогда горизонтальная ось рисуется от (0, CY) до (WX, CY), а вертикальная — от (CX, 0) до (CX, WY).

Эти действия программируем в draw после вызова функции setview.

1.4. Вспомогательные функции

Вспомогательные функции нужны для того, чтобы развернуть ось Y, которая в устройстве вывода направлена вниз. Если этого не сделать, то либо придется выполнять этот разворот все время при рисовании, либо график будет перевернутым.

Рисовать мы будем точками и линиями, соответственно, определим функцию pixel, которая нарисует точку, и функции для рисования линий, аналогичные функциям MoveToEx и LineTo.

Функция pixel:

```
// разворачивает ось Y
void pixel(HDC hdc, int X, int Y) {
    SetPixel(hdc, X, CY - Y, RGB(255, 0, 0));
}
```

Здесь мы используем системную функцию для рисования пикселя, которая требует указания цвета. Мы выбираем красный.

Функция, которая установит текущую точку:

```
// разворачивает ось Y
void moveTo(HDC hdc, int X, int Y) {
    MoveToEx(hdc, X, CY - Y, 0);
}
```

Функция, которая нарисует линию:

```
// разворачивает ось Y
void lineTo(HDC hdc, int X, int Y) {
    LineTo(hdc, X, CY - Y);
}
```

Реализуем функции перед функцией draw и после функции setview.

1.5. Функция графика

Определим функцию, которая будет вычислять точку графика для произвольной функции. Для примера, точку графика функции sin *x*.

Функцию графика разместим над функцией draw:

```
// функция графика
double funca(double x) {
    return sin(x);
}
```

Впоследствии в нее можно вписать любую другую функцию.

1.6. Рисование точками

Первый опыт заключается в том, чтобы нарисовать график функции при помощи точек. Описываем функцию byPoints перед функцией draw:

```
// рисует график точками
void byPoints(HDC hdc) {
    double x, y;
  }
}
```

Есть два подхода к рисованию графика, либо в системе координат графика, либо в системе координат устройства вывода. В любом случае мы рисуем график, вычисляя значение *y* для последовательности значений *x* через некоторый интервал *dx*. Если использовать систему координат графика, то нужно ввести переменную, которая будет соответствовать смещению вдоль оси X на один пиксель, получится некоторая погрешность.

Поэтому проще рисовать в системе координат устройства вывода, то есть в пикселях. Тогда приращение *x* вдоль оси X нужно принять равным единице (одному пикселю). Будем рисовать именно так.

Тогда функция byPoints рисует график следующим образом.

Для каждого *x* от нуля до WX исключительно, изменяя *x* в каждой итерации на единицу, вычислить значение *y* для данного *x*, переведенного в систему координат графика, и нарисовать точку при помощи функции pixel, переведя полученное значение *y* в систему координат устройства вывода.

Проще говоря, *x* и *y* нужно умножать или делить на масштабные коэффициенты SX и SY (ScaleX и ScaleY) при вычислении funca(*x*).

Таким образом, в функции byPoints нужно организовать цикл по переменной *x*, которая изменяется от нуля до WX исключительно, в цикле вычислить *y* при помощи funca(*x*), и нарисовать точку pixel(GDC, *x*, *y*), не забывая при этом умножать (или делить) *x* на SX, а вычисленное значение *y* на SY.

Разбираемся с масштабными коэффициентами и получаем график синусоиды во всю длину устройства вывода. Помним, что *x* и *y* у нас в пикселях.

После того, как синусоида будет нарисована, нужно изменить размеры области графика, например, задать GX равным 30, а GY равным 1. Поэкспериментируйте с этими значениями. Цель — убедиться, что рисование точками не всегда дает хороший результат.

1.7. Рисование линиями

Теперь перейдем к рисованию графика при помощи линий. В большинстве случаев это лучший вариант. Описываем функцию byLines:

```
// рисует график линиями
void byLines(HDC hdc) {
    double x, y;
}
/* контекст устройства GDC */
void draw() {
    . . .
    byPoints(GDC);
    byLines(GDC);
}
```

Рисовать линиями будем поверх графика, нарисованного точками. В какой-то момент графики сольются, а в какой-то момент мы увидим разницу.

Рисование линиями происходит не так, как рисование точками. Сначала нужно вычислить начальную точку кривой графика, и установить ее как текущую. Затем в цикле вычислять следующие точки, меняя x на некоторое значение, заданное константой DX, и рисовать линии до. Нет смысла задавать DX, равное единице, нужно подобрать оптимальное значение.

Вернем первоначальные значения GX и GY.

Сначала в функции byLines вычислим y для первой точки, при $x = 0$. После этого установим текущую точку при помощи функции moveTo.

Затем формируем цикл по x от DX до WX, изменяя в каждой итерации x на значение DX. В итерации вычисляем y = funca(x), и рисуем линию при помощи функции lineTo. При этом точно также, как и при рисовании точками, используем масштабирующие коэффициенты SX и SY.

После того, как синусоида получится, нужно пробовать изменять параметры GX, GY и DX. В какой-то момент изменение DX сделает график ломаным, а не кривым, и мы увидим нарисованную ранее кривую при помощи точек.

1.8. Контрольные вопросы и упражнения

1. Как задаются явные и неявные функции?
2. Как задаются непараметрические и параметрические функции?
3. Что означает параметр функции в параметрической форме?
4. Что называется нормализованным параметром?
5. Отобразите график функции $y = x^2$.
6. Отобразите график функции $y = x^{0,5}$.

Для выполнения пунктов 5 и 6 определите дополнительные функции funcb, funcc, setviewb, setviewc.

2. Работа КГ-202. Сегмент кубического сплайна (2 часа)

Цели:
- изучение кубических сплайнов.
Задачи:
- сегмент непараметрического кубического сплайна;
- сегмент параметрического кубического сплайна.
Опорные документы:
[4, с.125-128]
[2, с.161]

2.1. Рабочее пространство

Для выполнения работ по сплайновым кривым используется проект spline, подготовленный преподавателем. Скачайте архив проекта, извлеките из него каталог spline в корневой каталог диска C:. Откройте проект. Установите платформу x86 или Win32, в зависимости от того, что есть. Убедитесь, что проект не содержит ошибок.

В проекте много модулей.

Модуль spline.h содержит константы, переменные, и прототипы. Вносить в него изменения не требуется. Модуль spline.cpp содержит основные функции проекта. Вносить в него изменения не требуется.

Модули 1-nonpa.h, 2-param.h, 3-multis.h, 4-bezier.h и 5-bspline.h являются рабочими. Здесь описываются функции, которые рисуют сплайновые кривые.

Точки кривой задаются перемещением схватов (рисунок 1).

Рисунок 1 — Схваты и направляющие векторы

Схваты пронумерованы от нуля до некоторого числа. Направляющие векторы имеют другой цвет, они задают кривизну в концевых точках. Два схвата на концах направляющих векторов обозначены точками.

Координаты центров схватов передаются в функции рисования как массивы px и py. Схват с номером *n* имеет координаты px[*n*] и py[*n*].

Концевой схват левого направляющего вектора имеет координаты px[GRIP_L] и py[GRIP_L], концевой схват правого направляющего вектора имеет координаты px[GRIP_R] и py[GRIP_R].

Максимальное количество точек определяет константа MAX_POINT, значение которой равно 8. Направляющие векторы и задающие многоугольники рисуются автоматически.

2.2. Непараметрический сплайновый сегмент

Рабочий модуль 1-nonpa.h. Для рисования непараметрического сплайнового сегмента предназначена функция draw_nonparam.

```
void draw_nonparam(HDC hdc, const int * px, const int * py, int ln) {}
```

Параметрами функции являются контекст устройства, массивы координат точек px и py, число линий *ln*, которыми рисуется кривая. Сегмент рисуется вдоль оси X, и $y(x)$ вычисляется при изменении x от 0 до длины кривой t. При рисовании к x прибавляется x_0.

Расчетные формулы следующие.

Точка кривой вычисляется по формуле (1).

$$y = y_0 + k_0 x + a_3 x^2 + a_4 x^3 \quad (1)$$

Здесь:

y_0 = py[0],

k_0 — тангенс угла наклона левого направляющего вектора,

a_3 и a_4 — коэффициенты, которые нужно вычислить,

x — текущее значение расчетной координаты вдоль оси X.

Перед тем, как рисовать сегмент кривой сплайна, нужно вычислить тангенсы углов наклона направляющих векторов и коэффициенты a_3 и a_4.

Начинать расчеты нужно с k_0 и k_1.

1. Вычисляем разности координат dx и dy конечных точек направляющих векторов. Тогда $k = dy/dx$. Сначала вычисляем dx и dy конечных точек левого направляющего вектора и k_0, затем используем те же переменные для вычисления k_1.

2. Вычисляем длину кривой t вдоль оси X.

3. Вычисляем коэффициент a_3 по формуле (2).

$$a_3 = 3(y_1 - y_0) / t^2 - (k_0 + k_0 + k_1) / t \quad (2)$$

4. Вычисляем коэффициент a_4 по формуле (3).

$$a_4 = 2(y_0 - y_1) / t^3 + (k_0 + k_1) / t^2 \quad (3)$$

5. Вычисляем шаг приращения x по длине кривой. Результат записываем в переменную ax, деля длину кривой t на число отрезков *ln*.

6. Устанавливаем начальную точку рисования (x_0, y_0) при помощи функции MoveToEx.

7. Программируем цикл из *ln* итераций. В каждой итерации:

- вычисляем новое значение x, прибавляя к нему приращение ax;
- вычисляем y по формуле (1), преобразуя ее в формулу (4)

$$y = y_0 + (k_0 + (a_3 + a_4 * x) * x) * x \quad (4)$$

- рисуем линию до точки $(x_0 + x, y)$ при помощи функции LineTo.

Далее исследуем поведение рисуемого сегмента.

2.3. Параметрический сплайновый сегмент

Рабочий модуль 2-param.h. Параметрический сплайновый сегмент рисует функция draw_segment. Эта же функция используется при рисовании многосегментной кривой:

```
void draw_segment(HDC hdc, int x0, int y0, int x1, int y1, . . .) {
}
```

Один сегмент сплайна вычисляется в функции draw_param:

```
void draw_param(HDC hdc, const int * px, const int * py, int ln) {
    // дифференциалы в концевых точках
    double k0x = 0, k0y = 0, k1x = 0, k1y = 0;
    // длина кривой (хорда)
    double t = 0;
    // выводим сегмент
    draw_segment(hdc, px[0], py[0], px[1], py[1], k0x, k0y, . . .);
}
```

Начнем с функции draw_param.

Функции передаются контекст устройства hdc, массивы координат px и py, и количество линий *ln*, которыми рисуется сегмент. В фукции нужно вычислить дифференциалы в конечных точках и длину кривой. Эти параметры затем используются при вызове функции рисования сегмента.

Дифференциалы в концевых точках вычисляем по формулам (5) и (6).

$$k_x = dx/d \quad (5)$$

$$k_y = dy/d \quad (6)$$

Здесь:

dx — разность координат *x* концевых точек направляющего вектора;

dy — разность координат *y* концевых точек направляющего вектора;

d — длина направляющего вектора.

Длину направляющего вектора вычисляем по формуле (7).

$$d = (dx^2 + dy^2)^{1/2} \quad (7)$$

Сначала вычисляем дифференциалы *k*0*x* и *k*0*y* в начальной точке, а затем дифференциалы *k*1*x* и *k*1*y* в начальной точке.

Наконец, вычисляем длину кривой *t*, равную длине хорды между начальной и конечной точками кривой. Начальной точкой является в этом случае точка с индексом 0, а конечной — точка с индексом 1.

Переходим к функции draw_segment.

Кроме контекста устройства, в эту функцию передаются координаты начальной и конечной точек сегмента *x*0, *y*0, *x*1, *y*1, дифференциалы в начальной и конечной точках *k*0*x*, *k*0*y*, *k*1*x*, *k*1*y*, длина кривой *t*2 и количество линий *ln*, которыми рисуется сегмент. Остается только вычислить коэффициенты кривой и собственно *ln* точек кривой.

Точка кривой для значения параметра *t* вычисляется по формуле (8).

$$P(t) = P_0 + P'_0 t + a_3 t^2 + a_4 t^3 \quad (8)$$

Здесь:

P_0 — начальная точка кривой,

P'_0 — дифференциал в начальной точке,

a_3 и a_4 — коэффициенты, требующие вычисления.

Эту формулу для плоского случая можно разложить на составляющие ее координаты x и y, которые тогда вычисляются по формулам (9) и (10).

$$x(t) = x_0 + x'_0 t + a_{3x} t^2 + a_{4x} t^3 \quad (9)$$

$$y(t) = y_0 + y'_0 t + a_{3y} t^2 + a_{4y} t^3 \quad (10)$$

Здесь x_0, y_0, x'_0 и y'_0 передаются функции как параметры $x0$, $y0$, $k0x$ и $k0y$, и остается только вычислить коэффициенты a_{3x}, a_{3y}, a_{4x}, a_{4y} по формулам (11), (12), (13) и (14).

$$a_{3x} = 3(x_1 - x_0) / t_2^2 - (x'_0 + x'_0 + x'_1) / t_2 \quad (11)$$

$$a_{3y} = 3(y_1 - y_0) / t_2^2 - (y'_0 + y'_0 + y'_1) / t_2 \quad (12)$$

$$a_{4x} = 2(x_0 - x_1) / t_2^3 + (x'_0 + x'_1) / t_2^2 \quad (13)$$

$$a_{4y} = 2(y_0 - y_1) / t_2^3 + (y'_0 + y'_1) / t_2^2 \quad (14)$$

Здесь x'_1 и y'_1 передаются функции как параметры $k1x$ и $k1y$, а длина сегмента t_2 передается как параметр $t2$.

После того, как коэффициенты будут определены, нужно вычислить приращение параметра dt, поделив длину кривой t_2 на число линий ln.

Приступаем к рисованию. Сначала нужно установить текущую точку рисования (x_0, y_0) при помощи функции MoveToEx.

Затем формируем цикл из ln итераций. В итерации:

- вычисляем новое значение t, прибавляя к нему приращение dt;
- вычисляем x по формуле (9), преобразуя ее в формулу (15)

$$x = x_0 + (x'_0 + (a_{3x} + a_{4x} * t) * t) * t \quad (15)$$

- вычисляем y по формуле (10), преобразуя ее в формулу (16)

$$y = y_0 + (y'_0 + (a_{3y} + a_{4y} * t) * t) * t \quad (16)$$

- рисуем линию до точки (x, y) при помощи функции LineTo.

Далее исследуем поведение рисуемого сегмента, для чего нажимаем переключатель «Параметрический».

2.4. Контрольные вопросы и упражнения

1. Напишите формулу сегмента непараметрического сплайна.
2. Напишите формулу сегмента параметрического сплайна.
3. Напишите формулу коэффициента a_3 параметрического сплайна.
4. Напишите формулу коэффициента a_4 параметрического сплайна.
5. Сравните непараметрический и параметрический сегменты.
6. Измените формулу вычисления дифференциала для непараметрического сплайна на $k = dy/d$, где d — длина направляющего вектора.
7. Сравните непараметрический и параметрический сегменты.

3. Работа КГ-203. Многосегментный кубический сплайн (2 часа)

Цели:
- построение многосегментного кубического сплайна.
Задачи:
- вычисление элементов тригональной системы уравнений;
- решение тригональной системы уравнений.
Опорные документы:
[4. с.128-140]

3.1. Рабочее пространство

Для выполнения данной работы используется проект spline, полученный в результате выполнения предыдущей работы.

Рисование многосегментного кубического сплайна кривой выполняем в функции draw_segments, модуль 3-multis.h:

```
void draw_segments(HDC hdc, int * px, int * py, int m, int ln) {
}
```

Параметрами функции являются контекст устройства, массивы координат точек сплайна, число точек *m*, число линий *ln*, которыми рисуется один сегмент сплайна.

3.2. Вычисление элементов тригональной системы

Многосегментный сплайн рисуется как последовательность сегментов, соединенных своими концами. Для обеспечения гладкости в точках соединения первые и вторые производные в этих точках должны быть совпадать. Это условие приводит к тригональной системе линейных уравнений, которую можно решить методом прогонки и вычислить дифференциалы в точках соединения.

Метод прогонки реализован в программе в функции solve_tds, параметрами которой являются массив дифференциалов K, массив свободных элементов B, массив длин сегментов T, и количество сегментов *n*. Все эти значения должны быть вычислены в функции draw_segments. Массивы K, B и T объявлены в начале модуля. В этом случае они изначально будут содержать нулевые значения, а это удобнее для отладки.

Проще всего вычислить значение *n*. Оно на единицу меньше значения *m*, передаваемого в функцию draw_segments.

Массив T вычисляется также относительно просто. Длина сегмента равна длине хорды, соединяющей концевые точки сегмента. Вычисляются длины всех сегментов, при этом длина первого сегмента записывается в элемент массива с индексом 1, длина второго сегмента записывается в элемент массива с индексом 2 и так далее до элемента с индексом *n*.

В массиве K должны быть заполнены два элемента, дифференциалы в начальной точке всей сплайновой кривой, и дифференциалы в конечной точке всей кривой. Дифференциалы вычисляются по формулам (5) и (6).

Дифференциалы в начальной точке записываются в элемент массива K с нулевым индексом. Дифференциалы в конечной точке записываются в элемент массива K с индексом n.

Свободные элементы вычисляются для элементов массива B с индексами от 1 до n–1. Если, например, в кривой три сегмента, то должны быть вычислены свободные элементы с индексами 1 и 2.

Элементы B вычисляются по формулам (17) и (18).

$$B.X_i = 3[(x_{i+1} - x_i)T_i / T_{i+1} + (x_i - x_{i-1})T_{i+1} / T_i] \quad (17)$$

$$B.Y_i = 3[(y_{i+1} - y_i)T_i / T_{i+1} + (y_i - y_{i-1})T_{i+1} / T_i] \quad (18)$$

В формулах (1) и (2) индекс i изменяется от 1 до n–1.

Заметим, что длины сегментов T вычисляются при помощи разностей координат dx и dy, которые используются также в формулах (17) и (18). Это дает возможность вычислить их один раз, а затем использовать в разных формулах. После вычисления всех массивов вызываем функцию solve_tds и убеждаемся, что вычисляются промежуточные дифференциалы.

3.3. Отладка

Для отладки выполните следующие действия.

Откройте FAR, зайдите в каталог C:\spline\Debug, и удалите в нем файл spline.ini при помощи F8 или Shift+Delete.

Установите точку остановки на вызове функции solve_tds. Запустите программу F5 и выберите многосегментный сплайн.

Тогда массивы должны содержать следующие значения:

K[0] = (x= 0.832 y= 0.554)

K[3] = (x= 0.832 y= 0.554)

T[1] = T[2] = T[3] = 36.055

B[1] = B[2] = (x= 180 y= 120)

Нажмите F10 и выполните функцию solve_tds. Результат:

K[1] = (x= 0.832 y= 0.554)

K[2] = (x= 0.832 y= 0.554)

3.4. Вывод сегментов

Выводим сегменты при помощи цикла из n итераций. Если в первой итерации принять i = 0, j = 1, тогда вызываем функцию draw_segments с параметрами hdc, px[i], py[i], px[j], py[j], K[i].x, K[i].y, K[j].x, K[j].y, T[j], *ln*.

3.5. Контрольные вопросы и упражнения

1. Напишите условия гладкости кривой в точках соединения.
2. Напишите элементы матриц A и B тригональной системы.

4. Работа КГ-204. Кривая Безье (2 часа)

Цели:
- изучение базиса кривой Безье.
Задачи:
- вычисление базиса кривой Безье;
- вычисление точек кривой Безье.
Опорные документы
[4, с.147]

4.1. Рабочее пространство

Для выполнения данной работы используется проект spline. Рисование кривой Безье выполняем в функции draw_bezier, модуль 4-bezier.h:

```
// рисует кривую Безье
void draw_bezier(HDC hdc, int * px, int * py, int m, int ln) {
}
```

Параметрами функции являются контекст устройства, массивы координат точек задающего многоугольника, число точек *m*, число линий *ln*, которыми рисуется один сегмент сплайна.

Кроме этой функции, в модуле определены функции для вычисления базисной функции кривой Безье:

```
// вычисляет базисную функцию кривой Безье
double jnit(int n, int i, double t) {
}

// вычисляет полиномиальный коэффициент
double cni(int n, int i) {
}

// факториал n
int fact[] = { 0, 0, 0, 0, 0, 0, 0, 0 };
```

4.2. Базисная функция

Сначала определим массив, вычисляющий факториал. Максимальное значение *n* факториала не превысит 7, так как максимальное количество точек равно 8, а *n* равно *m* – 1, *m* — параметр функции draw_bezier. Значения факториала для *n* от 0 до 7 следующие: 1, 1, 2, 6, 24, 120, 720, 5040.

Далее нужно определить функцию cni, которая вычисляет полиномиальный коэффициент (*n*, *i*) по формуле (19).

$$(n, i) = n! / i! / (n–i)! \quad (19)$$

Затем определяем базисную функцию jnit. Параметрами этой функции являются степень полинома *n*, номер вершины *i*, параметр *t*. Она вычисляется по формуле (20).

$$J_{n,i}(t) = (n, i)\, t^i (1–t)^{n–i} \quad (20)$$

Здесь (*n*, *i*) обозначает полиномиальный коэффициент (19).

4.3. Вывод кривой

Точки кривой вычисляются по формулам (21), (22) и (23).

$$w = J_{n,i}(t) \quad (21)$$

$$x = сумма[x_i \cdot w],\ 0 \le i \le n \quad (22)$$

$$y = сумма[y_i \cdot w],\ 0 \le i \le n \quad (23)$$

Порядок вычислений следующий.
Вычисляем приращение параметра *dt*, деля единицу на *ln*.
Устанавливаем начальную точку рисования (x_0, y_0).
Формируем цикл из *ln* итераций по переменной *k*. В итерации:
- вычисляем новое значение *t*, прибавляя к нему приращение *dt*;
- принимаем текущие значения *x* и *y* равными нулю;
- формируем цикл из *m* итераций по переменной *i*; в цикле:
- вычисляем базисную функцию в переменную *w*;
- вычисляем сумму *x*, увеличивая значение *x* на $x_i \cdot w$;
- вычисляем сумму *y*, увеличивая значение *y* на $y_i \cdot w$;
- после вычисления сумм *x* и *y* рисуем линию до точки (*x*, *y*).

4.4. Сравнение с системной функцией

Установите количество точек равным 4.
В конце функции draw_bezier напишите следующий код, рисующий кривую Безье при помощи системной функции:

```
POINT points[MAX_POINT];
for (i = 0; i < m; i++) {
    points[i].x = px[i];
    points[i].y = py[i];
}
PolyBezier(hdc, (PPOINT)points, m);
```

Сравните, насколько отличаются две кривые. Попробуйте изменять количество линий *ln*. Функция PolyBezier рисует сегменты 3-й степени, поэтому число точек может быть только 4, 7, 10 … Попробуйте установить число точек 7 и сравните результаты.

4.5. Контрольные вопросы и упражнения

1. Напишите базисную функцию кривой Безье.
2. Напишите формулу вычисления точки кривой Безье.
3. Исследуйте поведение кривой при разных значениях *n*.
4. Объясните, почему системная функция PolyBezier рисует кривую, не совпадающую с вашей, при числе точек, равном 7.

5. Работа КГ-205. B-сплайн (2 часа)

Цели:
- исследование B-сплайн кривых.
Задачи:
- вычисление узлового вектора B-сплайна;
- вычисление базисной функции B-сплайна;
- вычисление точек кривой B-сплайна.
Опорные документы:
[4, с.152]

5.1. Рабочее пространство

Для выполнения данной работы используется проект spline. Рисование B-сплайна выполняем в функции draw_bspline, модуль 5-bspline.h:

```
void draw_bspline(HDC hdc, const int * px, const int * py, int ... ) {}
```

Параметрами функции являются контекст устройства hdc, массивы координат точек задающего многоугольника px и py, число точек *m*, число линий *ln*, которыми рисуется сплайн, порядок B-сплайна *k*.

Кроме того, в модуле есть функция для вычисления узлового вектора:

```
// строит узловой вектор B-сплайна
void build_knot(const int * px, const int * py, int m, int k, . . . {
}
```

Параметрами функции build_knot являются массивы координат точек задающего многоугольника px и py, число точек *m*, порядок B-сплайна *k*, и массив значений узлового вектора X. Сам узловой вектор задан в начале модуля:

```
// узловой вектор
double V[MAX_POINT + MAX_POINT + 1];
```

Еще одна функция вычисляет весовую функцию B-сплайна:

```
// вычисляет базисную функцию B-сплайна
double nikt(int i, int k, double t, int ln) {
    return 0;
}
```

5.2. Узловой вектор

Начинаем программирование с вычисления узлового вектора. Будем реализовывать упрощенную версию кривой, без кратных вершин. Тогда узловой вектор строится следующим образом.

Если количество сегментов задающего многоугольника равно *n*, порядок B-сплайна равен *k*, то максимальное значение параметра $t_{max} = n - k + 2$.

Формируем значения X[*i*] такие, чтобы получилось *k* нулей в начале вектора, *k* значений t_{max} в конце вектора, а промежуточные значения увеличиваются на единицу.

Например, если $m = 8$, $n = 7$, то узловые векторы равны:
для $k = 2$: 0012345677
для $k = 3$: 00012345666
для $k = 4$: 000012345555
для $k = 5$: 0000012344444
для $k = 6$: 00000012333333
для $k = 7$: 000000012222222
для $k = 8$: 0000000011111111
при этом длина узлового вектора равна $m + k$.

Программируем функцию build_knot, и убеждаемся, что выводятся правильные узловые векторы в нижней части окна программы.

5.3. Базисная функция B-сплайна

Базисная функция nikt рекурсивная. Если k равно 1, функция возвращает 1, если $V[i] \cdot ln \le t \cdot ln \le V[i + 1] \cdot ln$, иначе возвращает 0, при этом $t \cdot ln$ приводится к целому значению, иначе возможны провалы весовой функции. Если $k > 1$, функция возвращает сумму двух дробей: $a + b$, каждая из которых состоит из числителя *num* и знаменателя *den*. Для дроби a используем формулы (24), (25) и (26).

$$nk = nikt(i, k - 1, t, ln) \quad (24)$$
$$num = (t - V[i]) \cdot nk \quad (25)$$
$$den = V[i + k - 1] - V[i] \quad (26)$$

Для дроби b используем формулы (27), (28) и (29).

$$nk = nikt(i + 1, k - 1, t, ln) \quad (27)$$
$$num = (V[i + k] - t) \cdot nk \quad (28)$$
$$den = V[i + k] - V[i + 1] \quad (29)$$

Если абсолютная величина *den* меньше 0,001, или абсолютная величина *nk* меньше 0,001, значение дроби a или b принимаем равным нулю.

5.4. Вычисление точек кривой

Вычисление точек кривой не отличается от вычисления точек кривой Безье. Приращение dt параметра t вычисляем делением t_{max} на число линий *ln*. Весовая функция вычисляется по формуле (30).

$$w = nikt(i, k, t, ln) \quad (30)$$

5.5. Контрольные вопросы и упражнения

1. Опишите базисную функцию B-сплайна.
2. Как порядок B-сплайна влияет на вид кривой?
3. При каких условиях кривая совпадает с многоугольником?
4. При каких условиях кривая касается всех ребер многоугольника?
5. При каких условиях кривая совпадает с кривой Безье?

Литература

1. Волков Е. А. Численные методы. Учеб. пособие для вузов. — 2-е изд., испр. — М.: Наука. Гл. ред. физ.-мат. лит., 1987. — 248 с.

2. Е. В. Шишкин, А. В. Боресков. Компьютерная графика. Динамика, реалистические изображения. М.: «Диалог-МИФИ», 1995. — 288 с.

3. Л. Аммерал. Принципы программирования в машинной графике. Пер. с англ. — М.: «Сол Систем», 1992. — 224 с.: ил.

4. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. / Пер. Ю. П. Кулябичев, В. Г. Иваненко; ред. Ю. И. Топчеев. — М.: Машиностроение, 1980. — 240 с., ил.

5. Самарский А. А., Гулин А. В. Численные методы: Учеб. пособие для вузов. — М.: Наука. Гл. ред. физ.-мат. лит., 1989. — 432 с.