Министерство образования Республики Беларусь
Учреждение образование
«Белорусский государственный университет
информатики и радиоэлектроники»
Кафедра интеллектуальных информационных технологий

С. А. САМОДУМКИН, М. Д. СТЕПАНОВА, Д. Г. КОЛБ

ПРАКТИКУМ ПО КОМПЬЮТЕРНОЙ ГРАФИКЕ

В 3-х частях

Часть 1

Рекомендовано Учебно-методическим объединением вузов Республики Беларусь по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для студентов учреждений, обеспечивающих получение высшего образования по специальности «Искусственный интеллект»

Под редакцией
профессора В. В. Голенкова

Минск БГУИР 2007

УДК 004.92 (075.8)
ББК 32.973.26 – 018.2 я 73
С 17

Рецензенты:

кафедра информатики и вычислительной техники
Высшего государственного колледжа связи
(заведующий кафедрой, кандидат технических наук, доцент Е. В. Новиков),

главный научный сотрудник Объединенного института проблем информатики
НАН Беларуси, доктор технических наук, профессор В. В. Старовойтов

Самодумкин, С. А.

С 17 Практикум по компьютерной графике : учеб.-метод. пособие. В 3 ч. Ч. 1 / С. А. Самодумкин, М. Д. Степанова, Д. Г. Колб; под ред. проф. В. В. Голенкова. – Минск : БГУИР, 2007. – 44 с. : ил.

ISBN 978-985-488-174-4

Практикум содержит теоретические сведения, упражнения для самостоятельной работы и задания для выполнения лабораторных занятий по курсу «Графический интерфейс интеллектуальных систем и когнитивная графика». В первую часть практикума включены растровые алгоритмы компьютерной графики.

Предназначен для студентов специальности «Искусственный интеллект» всех форм обучения.

УДК 004.92 (075.8)
ББК 32.973.26 – 018.2 я 73

ISBN 978-985-488-174-4 (ч. 1)
ISBN 978-985-488-173-7

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Обработка информации – одна из самых важных функций компьютера. В компьютерной среде информация может быть разделена на графические изображения, лингвистическую и звуковую информацию. Что касается обработки информации, связанной с изображениями, то здесь можно выделить три основных направления: компьютерная графика (КГ), обработка изображений (ОИ) и распознавание образов.

Задачей **компьютерной графики** является формирование изображений, или визуализация. Формирование изображения выполняется в три этапа: построение модели объекта (описание его геометрических элементов и их связей); подготовка модели к визуализации (выполнение геометрических преобразований, удаление невидимых линий); визуализация (отсечение окном, формирование растрового представления, вывод на терминал). Примером может служить область машиностроительного черчения, получившая свое развитие в системах автоматизированного проектирования (САПР). В таких системах часто необходимо выполнять сечения для произвольных тел.

Обработка изображений – это преобразование изображений. То есть и входными данными, и результатом является изображение. Примером обработки изображений могут служить: фильтрация, сглаживание, подавление шумов, повышение контраста, четкости, коррекция цветов и т.д. В качестве исходных материалов для обработки информации могут служить космические снимки, получаемые в цифровом виде; цифровые карты; отсканированные изображения и т. п. Задачей обработки изображений может быть как улучшение в зависимости от определенного критерия, так и специальное преобразование, кардинально изменяющее изображения. Например, для наложения космического снимка на карту необходимо его преобразовать в ортофотоплан, поскольку космоснимки выполняются в центральной проекции, а карта – в ортогональной. Оказывается, из снимка нельзя получить план аффинными преобразованиями, а только проективными, причем необходимы дополнительные данные о рельефе. В рассмотренном примере обработка изображений является промежуточным этапом для дальнейшего распознавания изображения. В данном случае перед распознаванием часто необходимо выделять контуры, создавать бинарное изображение, разделять по цветам.

Для **распознавания изображений** основная задача – получение описания изображенных объектов. Цель распознавания может формулироваться по-разному: выделение отдельных элементов (например условных знаков на карте), классификация изображения в целом (например проверка, изображен ли определенный объект на карте). Задача распознавания является обратной по отношению к визуализации.

Компьютерный практикум состоит из отдельных тем, соответствующих лабораторным работам. Каждая тема сопровождается необходимыми теоретическими сведениями и примерами работы алгоритмов компьютерной графики. В конце каждой темы приведены задание на выполнение лабораторной работы, а также упражнения, которые могут быть полезны студентам при подготовке к защите лабораторных работ и сдаче экзамена по дисциплине. Неотъемлемой частью практикума является виртуальная лаборатория, используемая при проведении лабораторных работ, целью которой является визуальное сопровождение пошаговой работы алгоритмов компьютерной графики. Библиотека алгоритмов размещена на сервере кафедры интеллектуальных информационных технологий БГУИР.

В процессе лабораторной работы студентам необходимо:

1. Изучить теоретическую часть (тема данного практикума, конспект лекций, дополнительная литература).

2. Осуществить программную реализацию задания средствами алгоритмического языка высокого уровня.

3. Подготовить и защитить отчет по лабораторной работе.

В отчете необходимо отразить цель лабораторной работы; привести формальную запись используемых в работе алгоритмов; распечатать тексты алгоритмов на языке программирования; дать результаты пошагового выполнения алгоритмов в виде таблицы; привести алгоритмическую оценку реализованных алгоритмов и сформулировать выводы по работе.

Авторы глубоко признательны рецензентам – коллективу кафедры информатики и вычислительной техники Высшего государственного колледжа связи, канд. техн. наук, доценту Е. В. Новикову; главному научному сотруднику Объединенного института проблем информатики д-ру техн. наук, профессору В. В. Старовойтову.

1. АЛГОРИТМЫ ПОСТРОЕНИЯ ОТРЕЗКОВ

В алгоритмах построения отрезков предполагается, что заданы координаты (x_1, y_1) начала и (x_2, y_2) конца отрезка. Экран монитора рассматривается как матрица дискретных элементов (пикселей), каждый из которых может быть подсвечен. Для дискретного устройства в общем случае нельзя непосредственно (однозначно) провести отрезок из одной точки в другую. Только для горизонтальных, вертикальных и наклоненных под углом 45° отрезков выбор пикселей очевиден. При любой другой ориентации выбрать нужные пиксели труднее.

Процесс определения пикселей, наилучшим образом аппроксимирующих заданный отрезок, называется ***разложением в растр***.

Сформулируем общие требования к алгоритмам рисования отрезков:

- отрезки должны выглядеть прямыми, начинаться и заканчиваться в заданных точках;
- яркость должна быть постоянной и не зависеть от длины и наклона;
- условие к быстродействию алгоритма.

Не все из перечисленных критериев могут быть полностью удовлетворены. Сама природа растрового дисплея исключает генерацию абсолютно прямых линий (кроме указанных выше случаев). Тем не менее при достаточно высоком разрешении монитора можно получить приемлемую аппроксимацию. Обычным компромиссом является нахождение приближенной длины отрезка, сведение вычислений к минимуму, предпочтительное использование целочисленной арифметики, а также реализация алгоритмов на аппаратном или микропрограммном уровне.

1.1. Цифровой дифференциальный анализатор

Один из методов разложения отрезка в растр (рис. 1.1) состоит в решении дифференциального уравнения отрезка прямой линии, записанного в форме

$$\frac{dy}{dx} = \frac{\Delta y}{\Delta x} = const, \text{ или } \frac{dy}{dx} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}, \qquad (1.1)$$

где $\Delta y = y_2 - y_1$ – разность y-координат концов отрезка (длина проекции отрезка на ось OY), $\Delta x = x_2 - x_1$ – разность x-координат (длина проекции отрезка на ось OX), $dy = y_{i+1} - y_i$, $dx = x_{i+1} - x_i$.

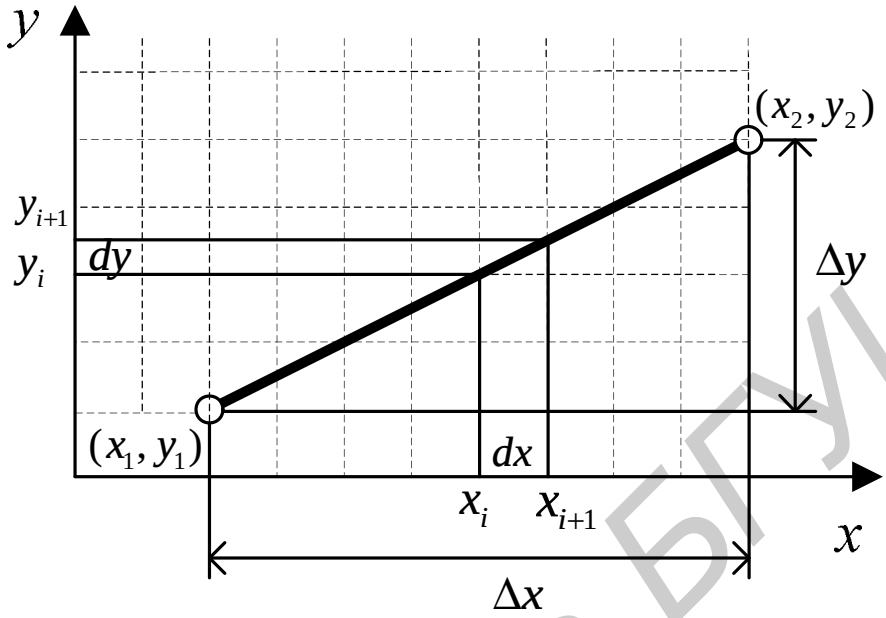


Рис. 1.1. Разложение отрезка в растр с использованием метода цифрового дифференциального анализатора

Если x и y изменяются вдоль прямой дискретно на dx и dy, то решение данного дифференциального уравнения представляется в виде

$$y_{i+1} = y_i + dy,\ x_{i+1} = x_i + dx. \qquad (1.2)$$

Выразим dy из уравнения (1.1):

$$dy = \frac{y_2 - y_1}{x_2 - x_1} dx.$$

Примем, что один шаг по целочисленной сетке на оси x равняется величине пикселя, т.е. $dx = x_{i+1} - x_i = 1$. Итерационная последовательность (1.2) будет иметь вид

$$y_{i+1} = y_i + \frac{y_2 - y_1}{x_2 - x_1},\ x_{i+1} = x_i + 1. \qquad (1.3)$$

Выражения (1.3) представляют собой рекуррентные соотношения для последовательных значений координат вдоль нужного отрезка. Когда $\frac{y_2 - y_1}{x_2 - x_1} > 1$,

то шаг по x будет приводить к шагу $y > 1$, поэтому x и y следует поменять ролями, придавая y единичное приращение ($dy = 1$).

Приведенный метод называется цифровым дифференциальным анализатором (ЦДА). В простом ЦДА либо dx, либо dy (большее из приращений Δx и Δy или большая длина проекции отрезка) выбирается в качестве единицы растра. В этом случае вычисления ведутся по соотношениям (1.2), где

$$dy = \frac{y_2 - y_1}{\mathbf{Max}(|x_2 - x_1|, |y_2 - y_1|)},\ dx = \frac{x_2 - x_1}{\mathbf{Max}(|x_2 - x_1|, |y_2 - y_1|)}.$$

Приведем алгоритм, работающий во всех квадрантах.

Предполагается, что концы отрезка (x_1, y_1) *и* (x_2, y_2) *не совпадают.* **Integer** – *функция преобразования вещественного числа в целое (взятие целой части),* **Sign** – *функция, возвращающая –1,0,1 для отрицательного, нулевого и положительного аргумента соответственно,* **Max** – *функция, возвращающая наибольшее значение двух аргументов,* **Plot** (a, b) – *отображение точки с координатами* (a, b).

Алгоритм разложения отрезка в растр методом ЦДА

Шаг 1. Аппроксимируем длину проекции отрезка

Длина = **Max** ($|x_2 - x_1)|$, $|y_2 - y_1)|$)

Шаг 2. Полагаем большее из приращений Δx или Δy равным единице растра

$dx = (x_2 - x_1)$ / Длина

$dy = (y_2 - y_1)$ / Длина

Шаг 3. Округляем величины, а не отбрасываем дробную часть (использование знаковой функции делает алгоритм пригодным для всех квадрантов) и отображаем концевую точку (x_1, y_1)

$x = x_1 + 0.5 \cdot$ **Sign** (dx)

$y = y_1 + 0.5 \cdot$ **Sign** (dy)

Plot (**Integer** (x),**Integer** (y))

Шаг 4. Основной цикл генерации отрезка

$i = 0$

while (i ≤ Длина)

{

$x = x + dx$

$y = y + dy$

Plot (**Integer** (x),**Integer** (y))

$i = i + 1$

}

Конец алгоритма.

Пример 1. Методом ЦДА разложить в растр отрезок из точки (0, 0) в точку (9, 4). Представить результаты пошагового выполнения алгоритма.

В данном случае $\Delta x = 9$, $\Delta y = 4$, поэтому разложение в растр будем проводить вдоль оси x.

Начальные установки

$x_1 = 0$, $y_1 = 0$, $x_2 = 9$, $y_2 = 4$,
Длина = 9, $dx = 1$, $dy = 4/9$.

Результаты итерационного процесса представим в виде следующей таблицы, а сгенерированный отрезок – на рис. 1.2.

i, шаг итерации	x	y	Plot (x, y), отображаемые координаты
0	0,5	0,5	(0, 0)
1	1,5	17/18	(1, 0)
2	2,5	25/18	(2, 1)
3	3,5	33/18	(3, 1)
4	4,5	41/18	(4, 2)
5	5,5	49/18	(5, 2)
6	6,5	57/18	(6, 3)
7	7,5	65/18	(7, 3)
8	8,5	73/18	(8, 4)
9	9,5	81/18	(9,4)

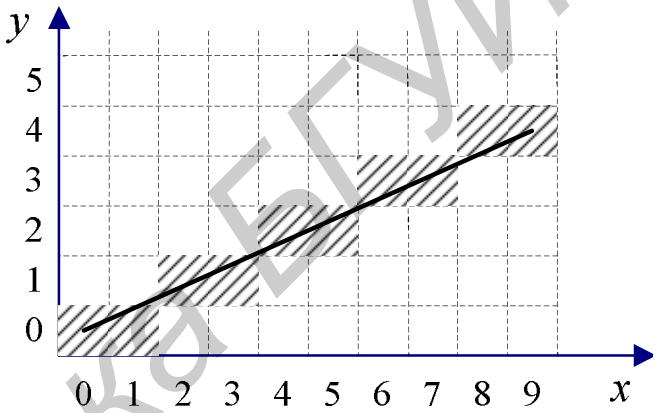


Рис. 1.2. Результат разложения отрезка в растр с использованием метода цифрового дифференциального анализатора

Приведенный алгоритм ЦДА не свободен от операций с вещественными числами. Исходя из чисто математических соображений, здесь все корректно, однако необходимо учесть, что в компьютере дробные числа представляются в формате с плавающей точкой не точно. Кроме погрешности представления чисел существует ошибка выполнения арифметических операций с плавающей точкой. С каждой итерацией четвертого шага алгоритма ЦДА ошибки накапливаются, и может так произойти, что $y \neq y_2$ на последнем шаге. Это необходимо учитывать при использовании алгоритма.

Недостатки алгоритма:

1. Использование операций с плавающей точкой обуславливает малую скорость. Однако это зависит от процессора и для различного типа компьюте-

ров может быть по-разному. В современных компьютерах, в которых процессоры используют эффективные способы аппаратного ускорения, время выполнения целочисленных операций не намного меньше. Для компьютеров предыдущих поколений разница была существенной, поэтому и старались разрабатывать алгоритмы только на основе целочисленных операций.

2. При вычислении координат путем добавления приращений может накапливаться ошибка вычислений координат.

1.2. Инкрементные алгоритмы. Алгоритм Брезенхема

В 1965 г. Брезенхем предложил подход, позволяющий разрабатывать так называемые инкрементные алгоритмы растеризации. Основной целью для разработки таких алгоритмов является построение циклов вычисления координат на основе только целочисленных операций сложения и вычитания без использования операций умножения и деления. Известны инкрементные алгоритмы не только для отрезков прямых, но и для кривых линий.

В алгоритме Брезенхема формирование растрового представления произвольного отрезка прямой осуществляется движением вдоль основной оси на один пиксель (в зависимости от углового коэффициента $\Delta y/\Delta x$). Изменение другой координаты (либо на нуль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние называется ошибкой.

Рассмотрим отрезок в первом октанте (рис. 1.3). В этом случае значение углового коэффициента лежит в диапазоне $0 \le \frac{\Delta y}{\Delta x} \le 1$. Из рисунка можно заметить, что если угловой коэффициент из точки (0, 0) больше, чем $1/2$, то точка растра (1, 1) лучше аппроксимирует ход отрезка, чем точка (1, 0). Если угловой коэффициент меньше $1/2$, то верно обратное. Для углового коэффициента, равного $1/2$, нет какого-либо предпочтительного выбора. В данном случае алгоритм выбирает точку (1, 1). Таким образом, выбор точки можно трактовать так: *рассматривается середина отрезка между возможными кандидатами и проверяется, где (выше или ниже этой середины) лежит точка пересечения отрезка прямой, после чего выбирается соответствующий пиксель.* Сравнивать с

нулем удобнее, чем с $1/2$, поэтому ошибку e на первом шаге с поправкой на половину пикселя можно вычислить следующим образом: $e = \frac{\Delta y}{\Delta x} - \frac{1}{2}$. В этом случае достаточно проверять только знак ошибки e. Приведем основные расчетные соотношения для построения отрезка в первом октанте.

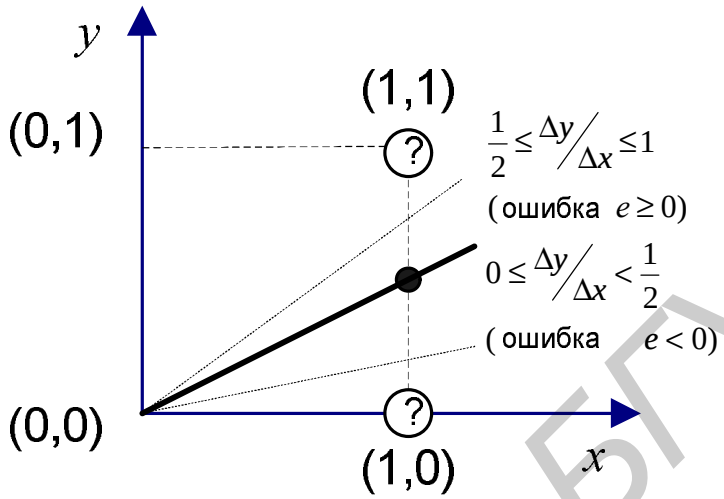


Рис. 1.3. Основная идея алгоритма Брезенхема

Основной выбирается ось абсцисс, т.к. $\Delta x > \Delta y$. Первоначальное значение ошибки $e = \frac{\Delta y}{\Delta x} - \frac{1}{2}$. Приращение принимается равным шагу растра, т.е. $x_{i+1} - x_i = 1$. Значение ошибки на каждом шаге определяется как $e = e + \frac{\Delta y}{\Delta x}$.

В зависимости от полученного значения ошибки возможны два варианта:

1. Ошибка $e < 0$. Отрезок пройдет ниже середины пикселя (см. рис. 1.3). В этом случае растровый элемент (x, y) лучше аппроксимирует положение отрезка.

2. Ошибка $e \geq 0$. Отрезок пройдет выше средней точки (см. рис. 1.3) и поэтому растровый элемент $(x, y + 1)$ лучше аппроксимирует положение отрезка. В этом случае необходимо скорректировать значение ошибки: $e = e - 1$, для того чтобы значение ошибки снова стало отрицательным (ошибка считается от y-координаты, которая теперь увеличилась на 1, поэтому из накопленной ошибки необходимо вычесть единицу).

Данные рассуждения применимы для первого октанта. Схема алгоритма Брезенхема генерации отрезка в первом октанте приведена на рис. 1.4.

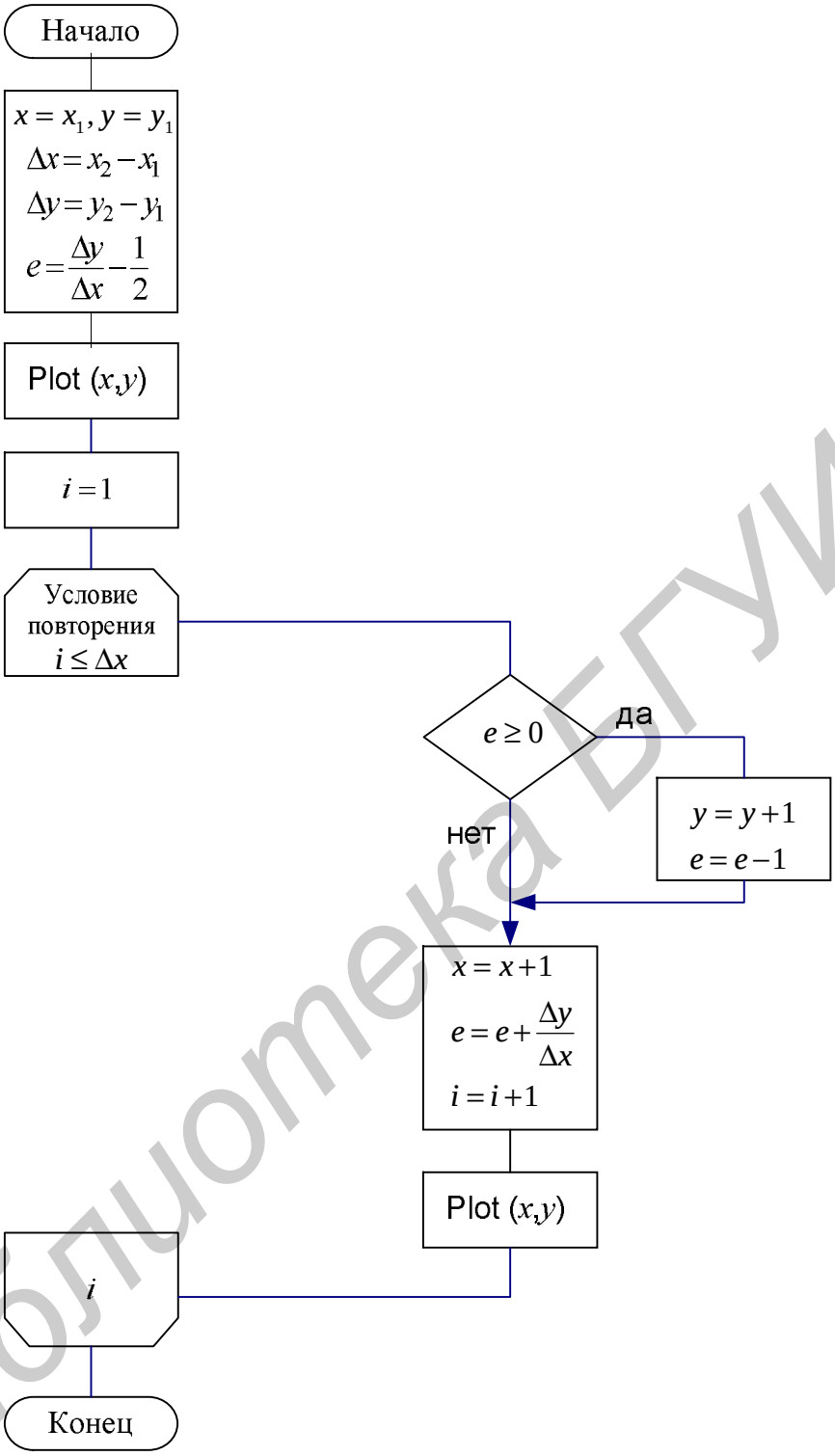


Рис. 1.4. Схема алгоритма Брезенхема генерации отрезка в первом октанте

Алгоритм Брезенхема в том виде, как он представлен на рис. 1.4, требует для вычисления углового коэффициента и оценки ошибки использования арифметики с плавающей точкой и операции деления. Так как в алгоритме Бре-

зенхема важен лишь знак ошибки e, то преобразование $e = 2 \cdot e \cdot \Delta x$ превратит предыдущий алгоритм в целочисленный.

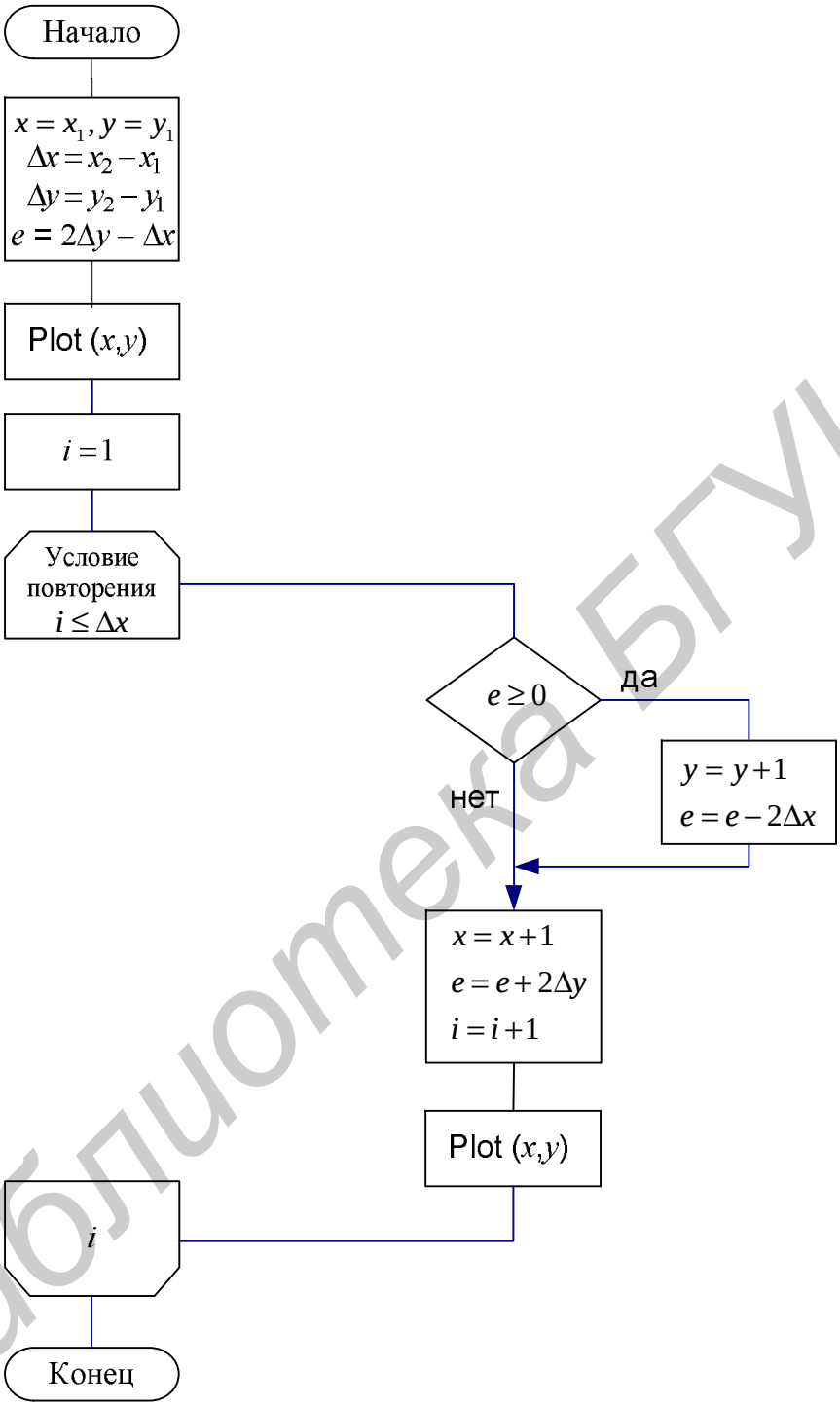


Рис. 1.5. Схема целочисленного алгоритма Брезенхема генерации отрезка

Чтобы реализация алгоритма Брезенхема была полной, нужно обрабатывать отрезок во всех октантах. Модификация заключается в том, чтобы учитывать в алгоритме номер квадранта, в котором лежит отрезок, и его угловой коэффициент. Разбор случаев для обобщенного алгоритма Брезенхема приведен

на рис. 1.6. Вне окружности показана постоянно изменяющаяся (ведущая) координата, а внутри приведено изменение координат.

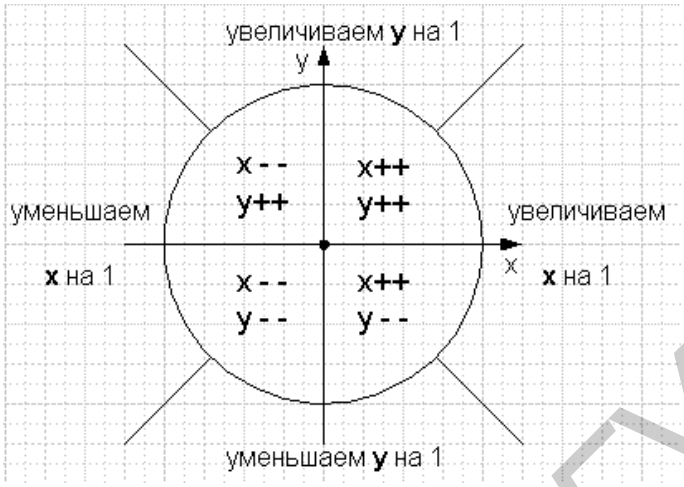


Рис. 1.6. Ведущие координаты и их изменения для всех октантов

Пример 2. Разложить в растр отрезок из точки (0, 0) в точку (9, 4), используя целочисленный алгоритм Брезенхема. Представить результаты пошагового выполнения алгоритма.

В данном случае $\Delta x = 9$, $\Delta y = 4$, поэтому разложение в растр будем проводить вдоль оси x.

Начальные установки

$x_1 = 0, y_1 = 0, x_2 = 9, y_2 = 4,$

$e = 2\Delta y - \Delta x = 2 \cdot 4 - 9 = -1.$

Результаты итерационного процесса представим в виде следующей таблицы, а сгенерированный отрезок – на рис. 1.7.

i, шаг итерации	e	x	y	Скорректированное значение ошибки e’	Plot (x, y), отображаемые координаты
0		0	0	-1	(0, 0)
1	-1	1	0	-7	(1, 0)
2	7	2	1	-3	(2, 1)
3	-3	3	1	5	(3, 1)
4	5	4	2	-5	(4, 2)
5	-5	5	2	3	(5, 2)
6	3	6	3	-7	(6,3)
7	-7	7	3	1	(7,3)
8	1	8	4	-9	(8,4)
9	-9	9	4	-1	(9,4)

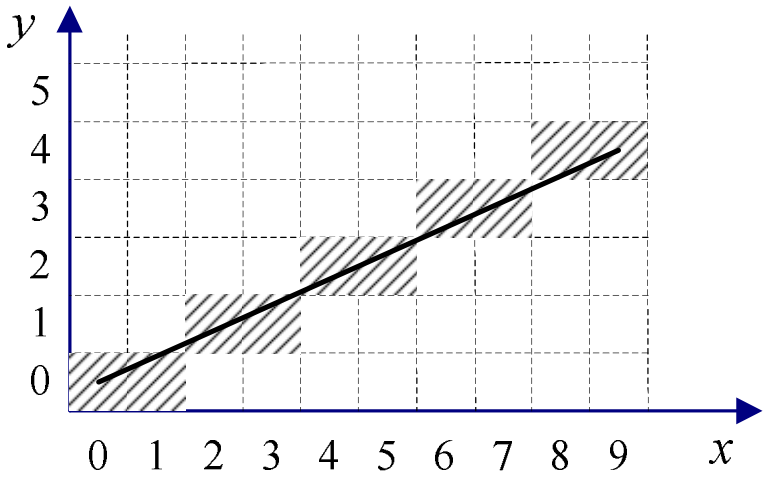


Рис. 1.2. Результат разложения отрезка из точки (0, 0) в точку (9, 4) в растр с использованием целочисленного алгоритма Брезенхема

1.3. Методы устранения ступенчатости

Все растровые алгоритмы построения геометрических примитивов подвержены одному общему серьезному недостатку – ступенчатости линий, или лестничному эффекту (по-английски называется ***aliasing***) (рис 1.7, а). Он возникает как следствие самой природы растрового дисплея, в котором изображение строится из дискретного набора точек. Эта ступенчатость хорошо заметна для глаза, в результате изображение теряет в реалистичности. Однако существует механизм ***anti-aliasing***, который призван бороться с этим дефектом.

Конечно, полностью устранить ступенчатость он не в силах, для этого необходимо отказаться от растровых мониторов. Однако с его помощью можно создать особую иллюзию, которая глазом будет восприниматься как гладкая линия (рис. 1.7, б). На рисунке линия размыта и кажется гладкой.

а) б)

Рис. 1.7. Лестничный эффект и способ его уменьшения

Первая группа методов устранения искажений связана с увеличением разрешения растра. Таким образом учитываются более мелкие детали. Однако все-

гда существует предел повышения разрешающей способности на уровне аппаратных средств компьютера.

Вторая группа методов устранения искажений изображения состоит в том, чтобы трактовать пиксель не как точку, а как конечную область. Так, в алгоритме Брезенхема генерации отрезков (см. подразд. 1.2) определение текущего отображаемого пикселя основывалось на расстоянии истинного положения отрезка и двух пикселей-претендентов. На основе значения такого расстояния активировался один из двух пикселей, но с максимальной интенсивностью. В результате получается характерный ступенчатый, или зазубренный отрезок (см. рис. 1.7, б). При наличии нескольких оттенков цвета, например полутонов серого, внешний вид отрезка улучшается путем размывания его краев. Другим способом сглаживания является изменение интенсивностей соседних пикселей. Для этого интенсивность двух пикселей устанавливается пропорционально расстоянию от центра точки до идеальной линии (рис. 1.8).

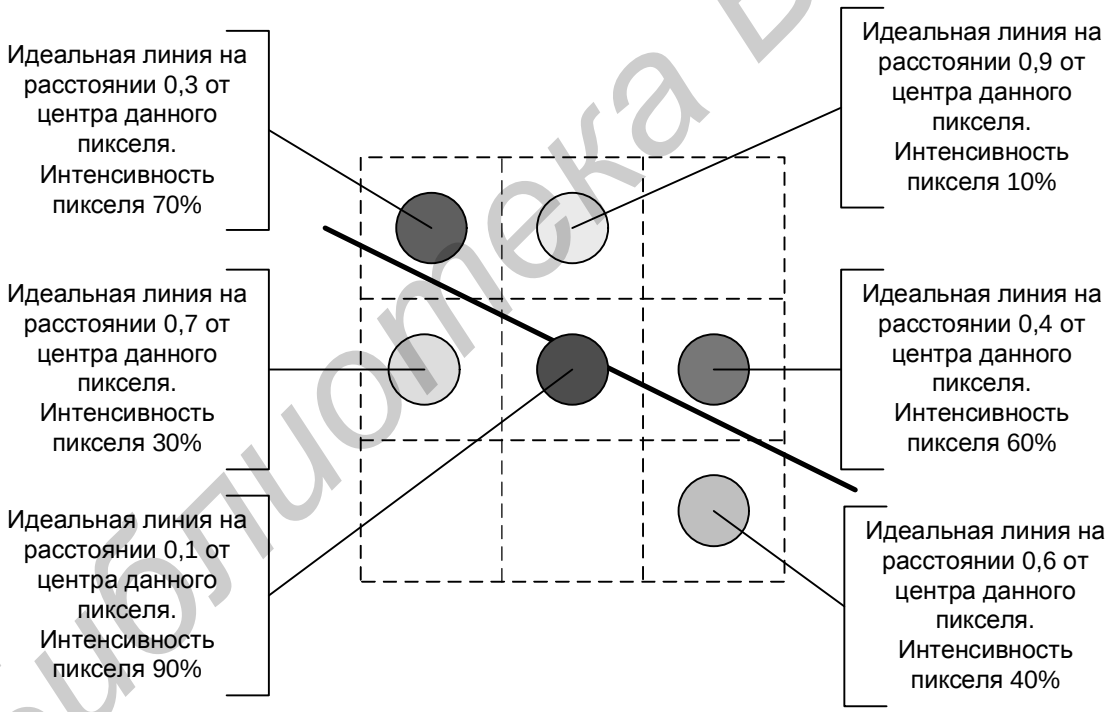


Рис. 1.8. Распределение интенсивности пикселя в зависимости от расстояния до идеальной линии (гипотетический случай)

Чем больше удалена точка от идеальной линии, тем меньше ее интенсивность. Значения интенсивности двух пикселей всегда дают в сумме единицу, т.е. это интенсивность одного пикселя, в точности попавшего на идеальную ли-

нию. Такое распределение придаст линии одинаковую интенсивность на всем ее протяжении, создавая при этом иллюзию, что точки расположены вдоль линии не по две, а по одной. Данный алгоритм известен как алгоритм Ву (Ксиаолин Ву в журнале Computer Graphics за июль 1991 года опубликовал статью об алгоритме сглаживания линий, сочетающем высококачественное устранение ступенчатости и скорость, близкую к скорости алгоритма Брезенхема без сглаживания).

Суть алгоритма Ву следующая. Горизонтальные, вертикальные и диагональные линии не требуют никакого сглаживания, поэтому их рисование выносится в отдельную функцию программы растровой развертки отрезков. Что касается других линий, то алгоритм Ву проходит их вдоль основной оси, подбирая координаты по неосновной оси аналогично алгоритму Брезенхема. Отличие состоит в том, что в алгоритме Ву на каждом шаге устанавливается не одна, а две точки. Например, если основной осью является X, то рассматриваются точки с координатами (x, y) и $(x, y + 1)$. В зависимости от величины ошибки, которая показывает, как далеко ушли пиксели от идеальной линии по неосновной оси, распределяется интенсивность между этими двумя точками (см. рис. 1.8).

Упражнения для самостоятельной работы

Упражнение 1.1. Методом ЦДА разложите в растр отрезок из точки (x_1, y_1) в точку (x_2, y_2). Представьте результаты пошагового выполнения алгоритма в виде следующей таблицы:

i	x	y	Plot (x,y)

Рассмотрите следующие случаи:

$x_1 = 0, x_2 = 8, y_1 = 0, y_2 = 5;$

$x_1 = 0, x_2 = -8, y_1 = 0, y_2 = 5;$

$x_1 = 0, x_2 = -8, y_1 = 0, y_2 = -5;$

$x_1 = 0, x_2 = 8, y_1 = 0, y_2 = -5.$

Приведите алгоритмическую оценку алгоритма.

Упражнение 1.2. На основе целочисленного алгоритма Брезенхема для первого октанта разработайте обобщенный целочисленный алгоритм Брезенхема генерации отрезков для всех квадрантов. Разложите в растр отрезок из точ-

ки (x_1, y_1) в точку (x_2, y_2). Представьте результаты пошагового выполнения алгоритма в виде следующей таблицы:

i, шаг итерации	*e*	*x*	*y*	Скорректированное значение ошибки *e'*	Plot (*x*, *y*), отображаемые координаты

Рассмотрите случаи из упражнения 1.1. Приведите алгоритмическую оценку алгоритма.

Упражнение 1.3. Составьте схему алгоритма Ву, обеспечивающего устранение лестничного эффекта. Сравните результаты работы алгоритма с алгоритмом Брезенхема. Рассмотрите случаи из упражнения 1.1.

Лабораторная работа № 1

Разработать элементарный графический редактор, реализующий построение отрезков с помощью алгоритма ЦДА, целочисленного алгоритма Брезенхема и алгоритма Ву. Вызов способа генерации отрезка задается из пункта меню и доступно через панель инструментов «Отрезки». В редакторе кроме режима генерации отрезков в пользовательском окне должен быть предусмотрен отладочный режим, где отображается пошаговое решение на дискретной сетке.

2. АЛГОРИТМЫ ПОСТРОЕНИЯ ЛИНИЙ ВТОРОГО ПОРЯДКА

В системах автоматизированного проектирования (САПР) широко используются конические сечения. Из курса аналитической геометрии известна теорема, утверждающая, что сечением любого круглого конуса плоскостью (не проходящей через его вершину) определяется кривая, которая может быть лишь эллипсом (частный случай – окружностью), гиперболой или параболой (рис. 2.1). Из рис. 2.1 легко усмотреть, что, поворачивая секущую плоскость вокруг прямой *PQ*, мы заставим изменяться кривую сечения. Будучи, например, первоначально эллипсом, она становится параболой, а затем превращается в гиперболу. Поэтому часто эллипсы, гиперболы и параболы называются коническими сечениями. Разложению конических сечений посвящено значительное число работ и естественно, что наибольшее внимание уделено окружности. Один из наиболее эффективных и простых для понимания алгоритмов генерации окружности принадлежит Брезенхему.

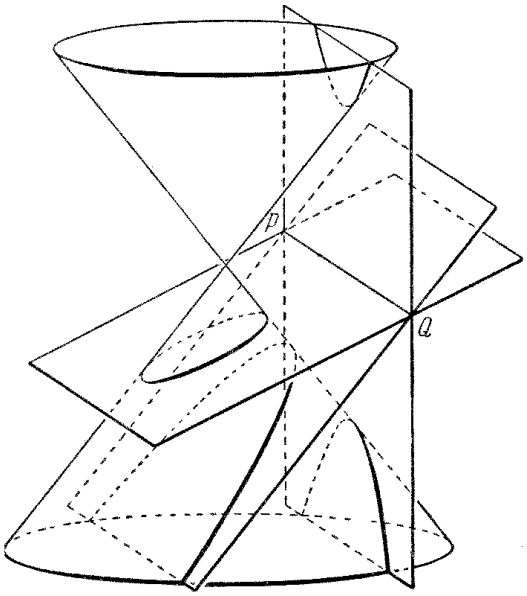


Рис. 2.1. Конические сечения

2.1. Алгоритм Брезенхема для генерации окружности

Алгоритм генерации окружности заключается в нахождении такого множества пикселей, которое наилучшим образом аппроксимирует кривую. Главное требование, предъявляемое к алгоритму, – эффективность.

Уравнение окружности задается следующим образом: $x^2 + y^2 = R^2$. При генерации окружности достаточно иметь алгоритм, генерирующий одну восьмую окружности, остальные фрагменты могут быть получены последовательными отражениями, как это показано на рис. 2.2.

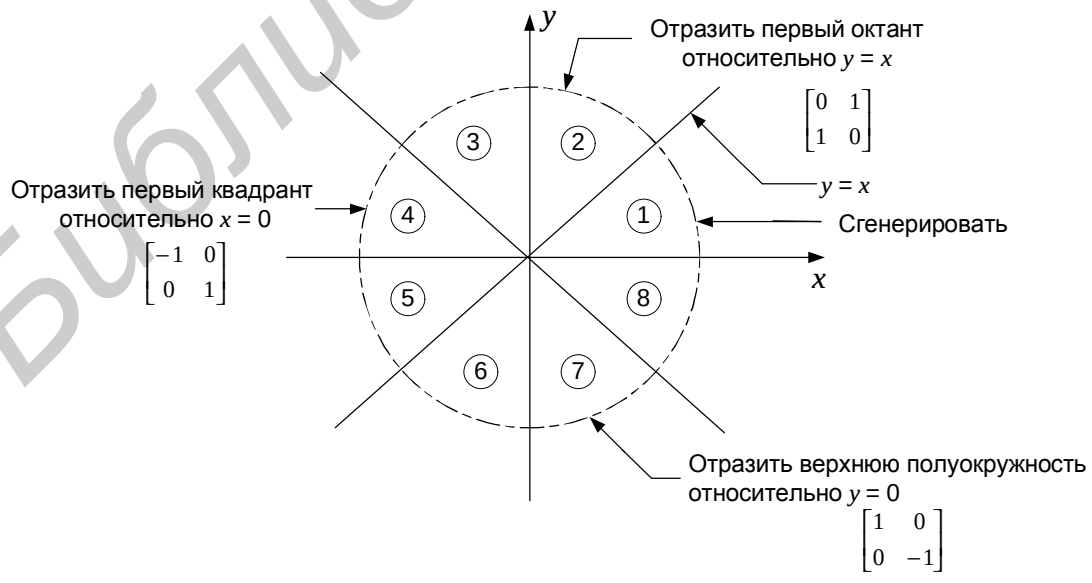


Рис. 2.2. Генерация полной окружности из дуги в первом октанте

Алгоритм Брезенхема начинает работать с точки (0, R) и рисует часть окружности по часовой стрелке, лежащую в первом квадранте. Предполагается, что центр окружности и начальная точка находятся точно в точках растра. Как и при рисовании линии, центральным понятием является ***ошибка*** – *разность между центром пикселя и действительным положением окружности*.

Вычисление координат очередного пикселя базируется на координатах ранее вычисленного пикселя. Пусть вычисленный пиксель имеет координаты (x_i, y_i), тогда при генерации окружности в первом квадранте по часовой стрелке пикселями-претендентами на прорисовку будут пиксели, изображенные на рис. 2.3, а ошибка будет вычисляться следующим образом:

$$\Delta = x_i^2 + y_i^2 - R^2 .$$

Значение ошибки обращается в нуль на самой окружности, она отрицательна внутри окружности и положительна вне ее (рис. 2.4). Таким образом, ошибку можно рассматривать как оценочную функцию.

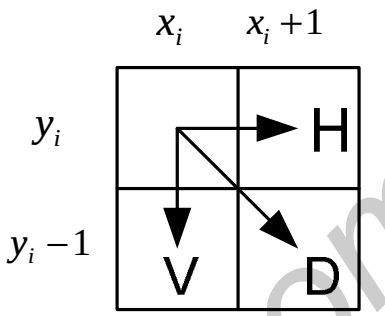


Рис. 2.3. Пиксели-претенденты

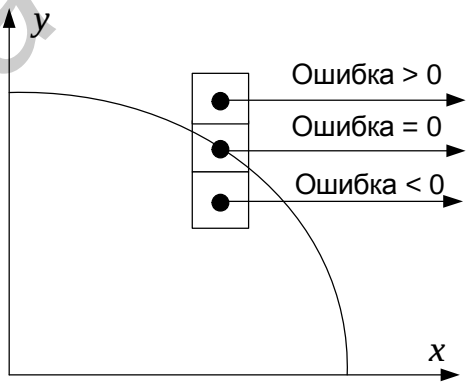


Рис. 2.4. Значения ошибки

Сущность метода состоит в следующем:

1) вычислить ошибки для пикселей претендентов;

2) выбрать среди них пиксель с минимальным абсолютным значением разности квадратов расстояний от центра окружности до пикселя-претендента и до окружности.

Имеется только три пикселя претендента: горизонтальный H, вертикальный V и диагональный D. Окружность при этом может принимать пять положений, которые показаны на рис. 2.5.

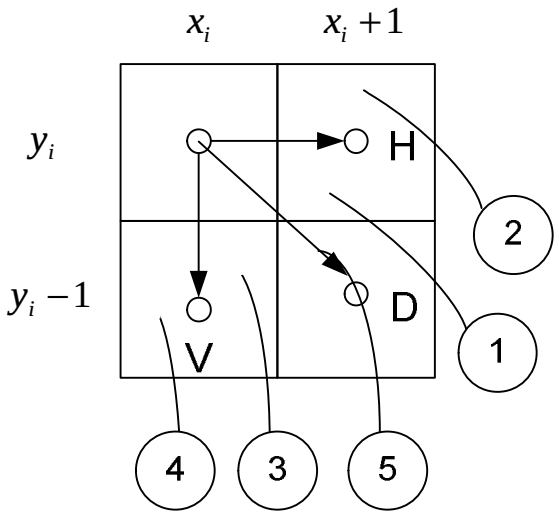


Рис. 2.5. Варианты прорисовки окружности

Вычислим значение ошибки для всех трех пикселей.

Для горизонтального пикселя:

$$\Delta_i = \Delta_H = (x_i+1)^2 + y_i^2 - R^2 = x_i^2 + 2x_i + 1 + y_i^2 - R^2 = \Delta_{i-1} + 2x_i + 1.$$

Для вертикального пикселя:

$$\Delta_i = \Delta_V = x_i^2 + (y_i-1)^2 - R^2 = x_i^2 + y_i^2 - 2y_i + 1 - R^2 = \Delta_{i-1} + (-2y_i + 1).$$

Для диагонального пикселя:

$$\Delta_i = \Delta_D = (x_i+1)^2 + (y_i-1)^2 - R^2 = x_i^2 + 2x_i + 1 + y_i^2 - 2y_i + 1 - R^2 =$$
$$= \Delta_{i-1} + 2(x_i - y_i + 1).$$

Выбор способа расчета определяется по значению Δ_i.

Случай А. Если $\Delta_i < 0$, то диагональная точка внутри окружности (необходимо нарисовать пиксели H или D) . На рис. 2.5 – варианты 1 и 2.

Случай Б. Если $\Delta_i > 0$, то диагональная точка вне окружности (необходимо нарисовать пиксели V или D). На рис. 2.5 – варианты 3 и 4.

Случай В. Если $\Delta_i = 0$, то диагональная точка лежит точно на окружности. На рис. 2.5 – вариант 5.

Рассмотрим случаи различных значений Δ_i в только что приведенной последовательности.

Случай А. Здесь в качестве следующего пикселя могут быть выбраны горизонтальный H или диагональный D.

Для определения того, какой пиксель выбрать – H или D, составим разность:

$$d = |\Delta_H| - |\Delta_D| = |(x_i + 1)^2 + y_i^2 - R^2| - |(x_i + 1)^2 + (y_i - 1)^2 - R^2|.$$

Для варианта 1:

При $d < 0$ расстояние от окружности до диагонального пикселя D больше, чем до горизонтального H. Напротив, при $d > 0$ расстояние до горизонтального пикселя H больше. Таким образом:

при $d < 0$ выбираем горизонтальное движение;

при $d > 0$ выбираем диагональное движение;

при $d = 0$, когда расстояния от окружности до обоих пикселей одинаковы, выбираем горизонтальный шаг.

Так как диагональный пиксель всегда лежит внутри окружности($\Delta_D < 0$), а горизонтальный вне ее ($\Delta_H > 0$), то вычислить d можно следующим образом:

$$d = |\Delta_H| - |\Delta_D| = (x_i + 1)^2 + y_i^2 - R^2 + (x_i + 1)^2 + (y_i - 1)^2 - R^2.$$

Дополнение до полного квадрата члена $(y_i)^2$ с помощью добавления и вычитания $(-2y_i + 1)$ дает

$$d = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] + 2y_i - 1 = 2(\Delta_D + y_i) - 1.$$

Для варианта 2:

Из рис. 2.5 ясно, что должен быть выбран горизонтальный пиксель H. Проверка компонент d показывает, что $\Delta_H < 0$ и $\Delta_D < 0$, т.е. диагональный и горизонтальный пиксели лежат внутри окружности, причем $d < 0$, так как диагональная точка больше удалена от окружности, т.е. по критерию $d < 0$ следует выбрать горизонтальный шаг, что верно.

Случай Б. Здесь в качестве следующего пикселя могут быть выбраны диагональный D или вертикальный V.

Для определения того, какой пиксель выбрать – D или V, составим разность:

$$d^* = |\Delta_D| - |\Delta_V| = |(x_i + 1)^2 + (y_i - 1)^2 - R^2| - |(x_i)^2 + (y_i - 1)^2 - R^2|.$$

Для варианта 3:

При $d^* < 0$ расстояние от окружности до вертикального пикселя V больше, чем до диагонального пикселя D. Напротив, в случае $d^* > 0$ расстояние от окружности до диагонального пикселя больше. Таким образом:

при $d^* < 0$ выбираем диагональное движение;

при $d^* > 0$ выбираем вертикальное движение;

при $d^* = 0$, когда расстояния равны, выбираем диагональный шаг.

Так как диагональный пиксель находится вне окружности ($\Delta_D > 0$), а вертикальный пиксель лежит внутри ее ($\Delta_V < 0$), то вычислить d^* можно следующим образом:

$$d^* = |\Delta_D| - |\Delta_V| = (x_i + 1)^2 + (y_i - 1)_i^2 - R^2 + (x_i)^2 + (y_i - 1)^2 - R^2 .$$

Дополнение до полного квадрата члена $(x_i)^2$ с помощью добавления и вычитания $(2x_i + 1)$ дает

$$d^* = 2[(x_i + 1)^2 + (y_i - 1)^2 - R^2] - 2x_i - 1 = 2(\Delta_D - x_i) - 1 .$$

Для варианта 4:

Из рис. 2.5 ясно, что должен быть выбран вертикальный пиксель V. Проверка компонент d^* показывает, что $\Delta_D > 0$ и $\Delta_V > 0$ (т.е. диагональный и вертикальный пиксели лежат вне окружности), причем $d^* > 0$, так как диагональная точка больше удалена от окружности, т.е. по критерию $d^* > 0$, как и в предыдущем варианте 3, следует выбрать вертикальный пиксель V, что соответствует выбору для варианта 3.

Случай В. Здесь в качестве следующего пикселя выбирается диагональный D.

Для компонент d имеем $\Delta_H > 0$ и $\Delta_D = 0$, следовательно, по критерию $d > 0$ выбираем диагональный пиксель. С другой стороны, для компонент d^* имеем $\Delta_V < 0$ и $\Delta_D = 0$, так что по критерию $d^* \le 0$ выбираем диагональный пиксель. Полученные результаты сведем в табл. 2.1.

Таким образом, мы определили итерационный механизм перехода от одного пикселя к другому. Первоначальное значение ошибки можно вычислить, основываясь на следующем. Если исходной точкой является точка с координатами (0, R), то при генерации окружности по часовой стрелке y будет монотонно убывающей функцией от x. Тогда

$$\Delta_1 = (x_{i+1})^2 + (y_{i+1})^2 - R^2 = (x_i + 1)^2 + (y_i - 1)^2 - R^2 = x_i^2 + 2x_i + 1 + y_i^2 - 2y_i + 1 - R^2 =$$
$$= 1 + R^2 - 2R + 1 - R^2 = 2 - 2R$$

Схема алгоритма генерации окружности в первом квадранте приведена на рис. 2.6.

Таблица 2.1

Координаты пикселей-претендентов и значение ошибки

	d		Пиксель	x_{i+1}	y_{i+1}	Δ_{i+1}
Случай А $\Delta_i < 0$	$d = 2(\Delta_D + y_i) - 1$	$d \le 0$	H	$x_i + 1$	y_i	$\Delta_i + (2x_i + 1)$
		$d > 0$	D	$x_i + 1$	$y_i - 1$	$\Delta_i + 2(x_i - y_i + 1)$
Случай Б $\Delta_i > 0$	$d* = 2(\Delta_D - x_i) - 1$	$d* \le 0$				
		$d* > 0$	V	x_i	$y_i - 1$	$\Delta_i + (-2y_i + 1)$
Случай В $\Delta_i = 0$			D	$x_i + 1$	$y_i - 1$	$\Delta_i + 2(x_i - y_i + 1)$

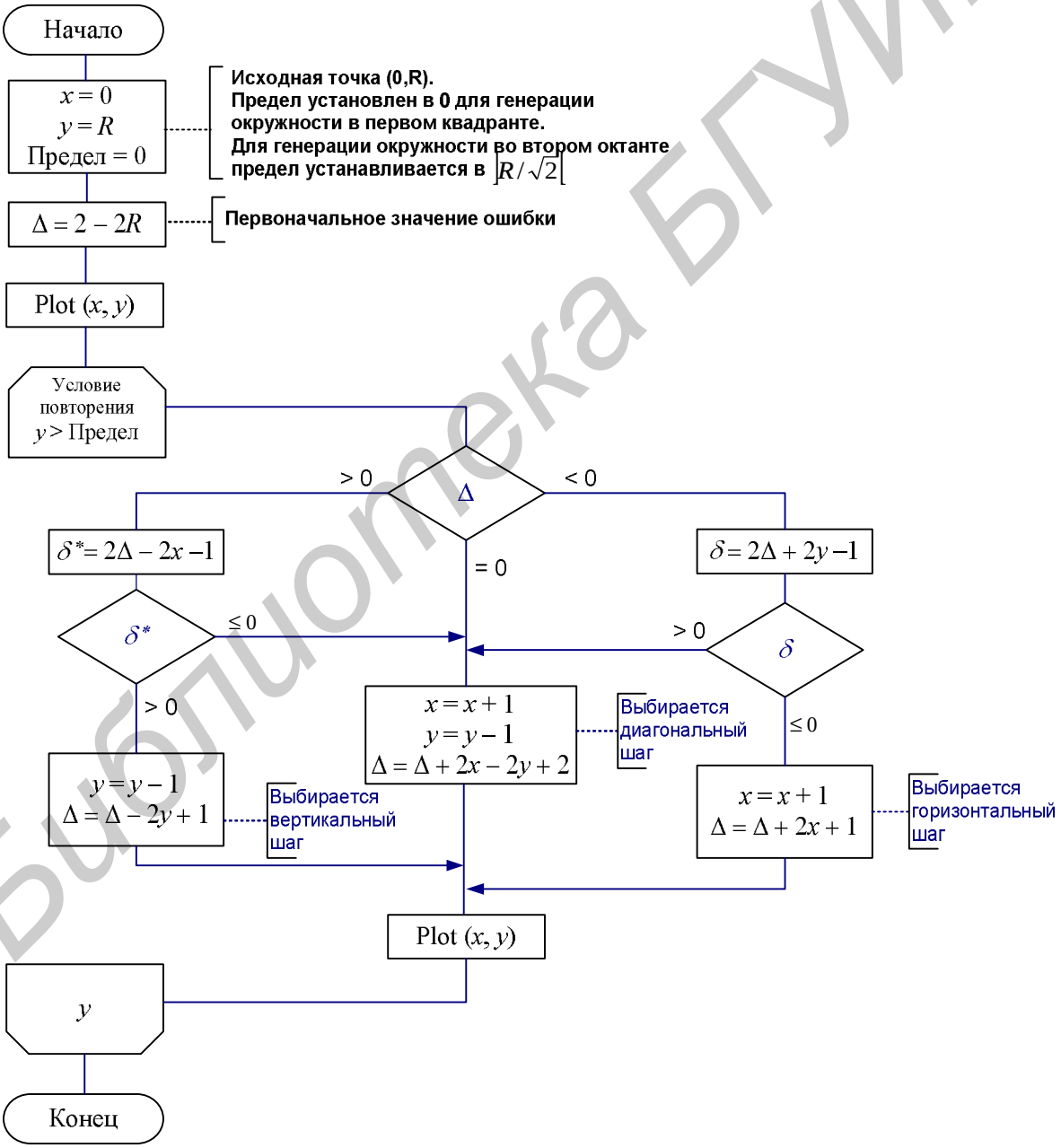


Рис. 2.6. Схема алгоритма Брезенхема генерации окружности в первом октанте

Пример 3. Сгенерировать окружность радиусом 8 с центром в начале координат. Генерируется только первый квадрант.

Начальные установки:

$x = 0, y = 8, \Delta_i = 2(1-8) = -14$, Предел = 0.

Результаты всех последовательных проходов сведены в таблицу и представлены на рис. 2.7.

шаг итерации	Δ_i	d	d^*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)
					0	8	-14	(0, 8)
1	-14	-13		H	1	8	-11	(1, 8)
2	-11	-7		H	2	8	-6	(2, 8)
3	-6	3		D	3	7	-12	(3, 7)
4	-12	-11		H	4	7	-3	(4, 7)
5	-3	7		D	5	6	-3	(5, 6)
6	-3	5		D	6	5	1	(6, 5)
7	1		-11	D	7	4	9	(7, 4)
8	9		3	V	7	3	4	(7, 3)
9	4		-7	D	8	2	18	(8, 2)
10	18		19	V	8	1	17	(8, 1)
11	17		17	V	8	0	18	(8, 0)

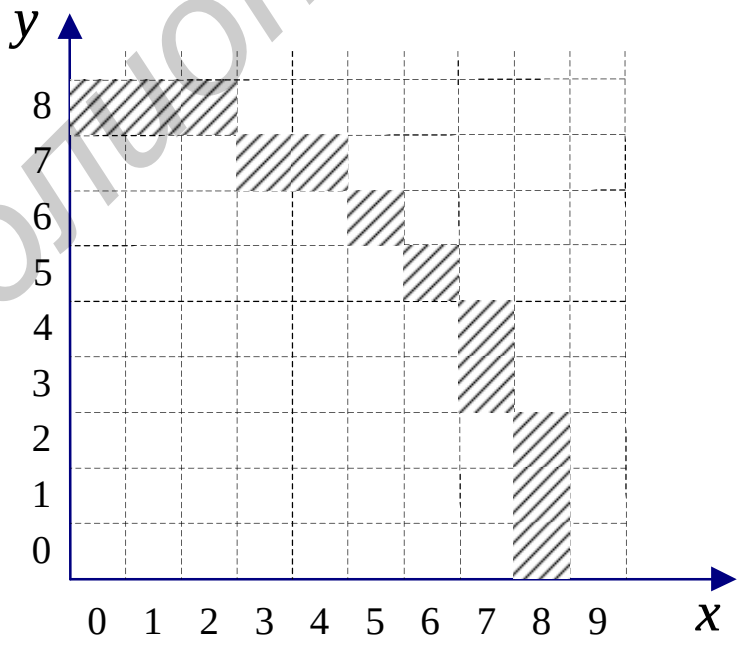


Рис. 2.7. Результат генерации окружности в первом квадранте с радиусом $R = 8$

2.2. Алгоритм генерации эллипса

Уравнение эллипса имеет следующий вид: $\frac{x^2}{a^2}+\frac{y^2}{b^2}=1$.

Перепишем это уравнение несколько иначе:

$b^2x^2+a^2y^2-a^2b^2=0$.

Алгоритм генерации эллипса разрабатывается аналогично окружности. Приведем конечный результат, описывающий способ вычисления ошибок (табл. 2.2).

Таблица 2.2

Координаты пикселей-претендентов и значение ошибки

	d		Пиксель	x_{i+1}	y_{i+1}	Δ_{i+1}
$\Delta_i<0$	$d=2(\Delta_D+a^2y_i)-1$	$d\le 0$	H	x_i+1	y_i	$\Delta_i+b^2(2x_i+1)$
		$d>0$	D	x_i+1	y_i-1	$\Delta_i+b^2(2x_i+1)+a^2(1-2y_i)$
$\Delta_i>0$	$d=2(\Delta_D-b^2x_i)-1$	$d\le 0$				
		$d>0$	V	x_i	y_i-1	$\Delta_i+a^2(1-2y_i)$
$\Delta_i=0$			D	x_i+1	y_i-1	$\Delta_i+b^2(2x_i+1)+a^2(1-2y_i)$

Первоначальное значение ошибки можно вычислить, основываясь на следующем. Если исходной точкой является точка (0, b), то при генерации эллипса по часовой стрелке y будет монотонно убывающей функцией от x. Тогда

$\Delta_1=b^2(x_i+1)^2+a^2(y_i-1)^2-a^2b^2=b^2+a^2(b^2-2b+1)-a^2b^2=a^2+b^2-2a^2b$.

Упражнения для самостоятельной работы

Упражнение 2.1. Разработайте алгоритм генерации окружности в первом квадранте и представьте результаты пошагового выполнения алгоритма для окружности радиусом R с центром в начале координат (рассмотрите случаи R = 7, R = 9, R = 10). Результаты представьте в виде таблицы.

шаг итерации	Δ_i	d	d*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)

Приведите алгоритмическую оценку алгоритма.

Упражнение 2.2. Разработайте алгоритм генерации эллипса в первом квадранте и представьте результаты пошагового выполнения алгоритма для эллипса с центром в начале координат (рассмотрите случай $a = 5$, $b = 3$). Результаты представьте в виде таблицы.

шаг итерации	Δ_i	d	d^*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)

Приведите алгоритмическую оценку алгоритма.

Упражнение 2.3. Разработайте алгоритм генерации гиперболы, заданной уравнением $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$, в первом квадранте и представьте результаты пошагового выполнения алгоритма с параметрами $a = 3$, $b = 5$. Исследуйте вид кривой, результаты представьте в виде таблицы.

шаг итерации	Δ_i	d	d^*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)

Приведите алгоритмическую оценку алгоритма.

Упражнение 2.4. Разработайте алгоритм генерации гиперболы, заданной уравнением $\frac{y^2}{b^2} - \frac{x^2}{a^2} = 1$, в первом квадранте и представьте результаты пошагового выполнения алгоритма с параметрами $a = 3$, $b = 5$. Исследуйте вид кривой, результаты представьте в виде таблицы.

шаг итерации	Δ_i	d	d^*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)

Приведите алгоритмическую оценку алгоритма.

Упражнение 2.5. Разработайте алгоритм генерации гиперболы, заданной уравнением $y^2 = 2px$, в первом квадранте и представьте результаты пошагового выполнения алгоритма с параметром $p = 8$. Исследуйте вид кривой, результаты представьте в виде таблицы.

шаг итерации	Δ_i	d	d^*	Пиксель	x	y	Δ_{i+1}	Plot (x, y)

Приведите алгоритмическую оценку алгоритма.

Лабораторная работа № 2

Разработать элементарный графический редактор, реализующий построение линий второго порядка: окружность, эллипс, гипербола, парабола. Выбор кривой задается из пункта меню и доступен через панель инструментов «Линии второго порядка». В редакторе кроме режима генерации линий второго порядка в пользовательском окне должен быть предусмотрен отладочный режим, где отображается пошаговое решение на дискретной сетке.

3. ИНТЕРПОЛЯЦИЯ И АППРОКСИМАЦИЯ КРИВЫХ

Значительное место в компьютерной графике занимает конструирование кривых и поверхностей. Часто при этом возникает *задача восполнения данных*: дано множество точек, через которые следует провести кривую или поверхность. Данная задача есть не что иное, как классическая задача интерполяции. Другой вид задач, когда кривая или поверхность должна проходить вблизи заданного множества точек, сводится к задаче аппроксимации.

С математической точки зрения, данные задачи имеют четкое решение. В теории аппроксимации вводятся следующие ограничения: интерполяционные ограничения, ограничения гладкости, условие ортогональности и вариационное ограничение. В компьютерной графике нас в первую очередь интересует не качество аппроксимации, измеряемое некоторой оценкой ошибки аппроксимации, а определенные свойства, выраженные через внешний вид кривой или поверхности. Кроме того, следует избегать «плохих» с вычислительной точки зрения методов.

В общем случае в задачах компьютерного проектирования формы не могут быть записаны в виде уравнения $y = f(x)$ с использованием обычных однозначных функций. Поэтому в компьютерной графике ведущую роль играет представление форм в *параметрическом виде*. Для двумерной кривой с параметром t координаты точки равны:

$x = x(t)$,

$y = y(t)$.

Тогда векторное представление точки на кривой: $P(t) = [x(t) \quad y(t)]$. Аналогично точка на пространственной кривой задается вектором $P(t) = [x(t) \quad y(t) \quad z(t)]$.

Производная, т.е. касательный вектор, есть $P'(t) = [x'(t) \quad y'(t)]$, где ' обозначает дифференцирование по параметру. Наклон кривой, dy/dx, равен

$$\frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{y'(t)}{x'(t)}.$$

Так как точка на параметрической кривой определяется только значением параметра, эта форма не зависит от выбора системы координат. Конечные точки и длина кривой определяются диапазоном изменения параметра. Часто бывает удобно нормализовать параметр на интересующем отрезке кривой к $0 \le t \le 1$. Такое параметрическое задание форм не только позволяет избежать математических проблем, но, кроме того, самым удобным образом описывает способ рисования кривых на печатающем устройстве и экране монитора, а также позволяет легко проводить аффинные преобразования.

Самое простое параметрическое представление у прямой. Для двух векторов положения P_1 и P_2 параметрический вид отрезка прямой между ними такой:

$$P(t) = P_1 + (P_2 - P_1)t, \qquad 0 \le t \le 1.$$

Так как $P(t)$ это вектор, у каждой его составляющей есть параметрическое представление $x(t)$ и $y(t)$ между P_1 и P_2:

$$\begin{aligned} x(t) &= x_1 + (x_2 - x_1)t, \\ y(t) &= y_1 + (y_2 - y_1)t. \end{aligned} \qquad 0 \le t \le 1.$$

3.1. Метод интерполяции Эрмита

В интерполяционном методе Эрмита по функции f подбирается многочлен, который интерполирует в заданном числе узлов не только саму функцию f, но и заданное число последовательных производных f: например, существует ровно один многочлен степени 3

$$p_3(t) = at^3 + bt^2 + ct + d,$$

удовлетворяющий условиям

$$p(a) = f(a), \quad p'(a) = f'(a),$$
$$p(b) = f(b), \quad p'(b) = f'(b).$$

Этот многочлен называется *кубическим интерполяционным многочленом Эрмита* для f, или кубическим сплайном [1].

Кривая разбивается на сегменты. В дальнейшем каждый кусочек кривой будем искать в параметрическом виде:

$$x(t) = f_x(t),$$
$$y(t) = f_y(t).$$

Кроме того, потребуем, чтобы $t \in [0,1]$. Значение функции будем искать в классе многочленов третьей степени, т.е.

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y. \end{aligned} \tag{3.1}$$

В эрмитовой форме граничным условием являются координаты P_1 и P_4 конечных точек сегмента и вектора касательных R_1 и R_4 в этих же точках. В соответствии с принятыми договоренностями получим

$$P_{1x} = x(0),\ R_{1x} = x'(0), P_{4x} = x(1),\ R_{4x} = x'(1),$$
$$P_{1y} = y(0),\ R_{1y} = y'(0), P_{4y} = y(1),\ R_{4y} = y'(1).$$

Будем описывать функцию $x(t)$ в матричном виде:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_x = T \cdot C_x. \tag{3.2}$$

Соединим информацию о граничных условиях с новой формой представления полинома:

$$x(0) = P_{1x} = [0\ 0\ 0\ 1] \cdot C_x,$$
$$x(1) = P_{4x} = [1\ 1\ 1\ 1] \cdot C_x.$$

Производная имеет следующий вид:

$$x'(t) = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot C_x.$$

[1] Форма математического сплайна повторяет контур физического сплайна, т.е. гибкой деревянной или пластмассовой линейки, проходящей через определенные точки. Для изменения формы сплайна используются свинцовые грузики. Меняя их количество и расположение, получившуюся кривую стараются сделать более гладкой, красивой и «приятной для глаза».

Для следующих двух граничных условий получим

$$x'(0) = R_{1x} = [0\ 0\ 1\ 0] \cdot C_x,$$

$$x'(1) = R_{4x} = [3\ 2\ 1\ 0] \cdot C_x.$$

Полученные четыре соотношения можно представить в матричном виде:

$$\begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}_x .$$

Решая это матричное уравнения для C_x, получим

$$C_x = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x , \text{ или } C_x = M_n \cdot G_{nx}, \qquad (3.3)$$

где M_n – матрица Эрмитовой формы (Эрмитова матрица), G_{nx} – вектор Эрмитовой геометрии.

Подставив выражение (3.3) в (3.2), получим

$$x(t) = T \cdot M_n \cdot G_{nx}. \qquad (3.4)$$

$$x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{1x} \\ P_{4x} \\ R_{1x} \\ R_{4x} \end{bmatrix}.$$

Функция $y(t)$ получается аналогичным способом. Таким образом, полином, который описывает сегмент кривой, будет иметь вид

$$P(t) = \begin{bmatrix} x(t) & y(t) \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{1x} & P_{1y} \\ P_{4x} & P_{4y} \\ R_{1x} & R_{1y} \\ R_{4x} & R_{4y} \end{bmatrix}. \qquad (3.5)$$

Пример 4. Построить сегмент кривой, используя форму Эрмита для следующих граничных условий:

$P_1 = [0 \quad 0]$, $P_4 = [1 \quad 0]$,

$R_1 = [0 \quad 1]$, $R_4 = [1 \quad 0]$.

Подставим значения граничных условий в выражение (3.5), получим вид кривой (рис. 3.1)

$$[x(t) \quad y(t)] = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 1 \\ 2 & -2 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

Параметр t изменяется в пределах [0,1]. Примем шаг изменения равный 0.1. В результате получим таблицу значений координат в зависимости от параметра t и по данным значениям построим сегмент кривой (рис. 3.1).

t	$x(t)$	$y(t)$
0.0	0.000	0.000
0.1	0.019	0.081
0.2	0.072	0.128
0.3	0.153	0.147
0.4	0.256	0.144
0.5	0.375	0.125
0.6	0.504	0.096
0.7	0.637	0.063
0.8	0.768	0.032
0.9	0.891	0.009
1.0	1.000	0.000

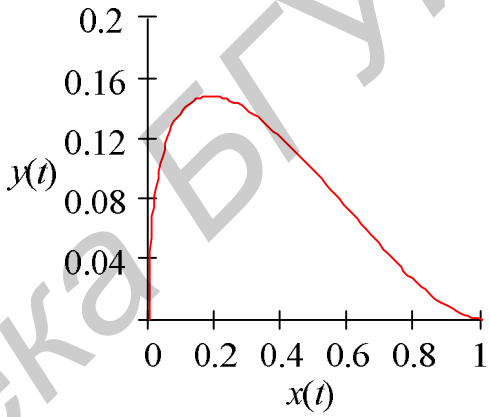


Рис. 3.1. Результат построения сегмента кривой с использованием формы Эрмита

На форму сплайна в большей степени влияет положение концевых точек, чем касательные векторов в данных точках. Однако, их эффект может оказаться значительным. Исследуем форму сплайна с заданными концевыми точками, с одинаковым направлением касательных векторов, но разной величины.

Пример 5. Задан симметричный сегмент сплайна его концевыми точками $P_1 = [0 \quad 0]$, $P_4 = [1 \quad 0]$ с одинаковым направлением касательных векторов α, но разной величины. Исследуем форму сплайна в зависимости от длины векторов касательных m.

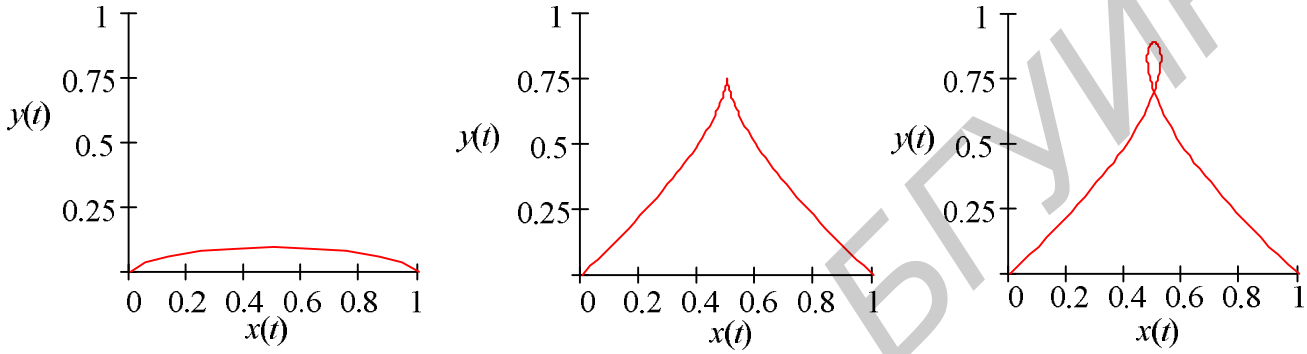
Обозначим отрезок, заданный концевыми точками сегмента сплайна, через l. Такой отрезок называется хордой. Для заданных граничных условий дли-

на хорды $l = 1$. Рассмотрим различные случаи в зависимости от длины векторов касательных.

1. Если $m << l$, то кривая выпуклая на концах и лежит внутри треугольника из хорды и касательных (рис. 3.2, а).

2. При возрастании длины векторов касательных кривая постепенно становится вогнутой и выходит за треугольник. В этом случае при значении $m = 3/\cos(a)$ у кривой появляется вершина (рис. 3.2, б).

3. При значениях $m > 3/\cos(a)$ у кривой появляется петля (рис. 3.2, в).



$a = p/4, m = 1/2,$

$R_1 = [\frac{1}{2\sqrt{2}} \quad \frac{1}{2\sqrt{2}}], R_4 = [\frac{1}{2\sqrt{2}} \quad -\frac{1}{2\sqrt{2}}]$

а

$a = p/4, m = 3/\cos(p/4),$

$R_1 = [3 \quad 3], R_4 = [3 \quad -3]$

б

$a = p/4, m = 5,$

$R_1 = [\frac{5}{\sqrt{2}} \quad \frac{5}{\sqrt{2}}], R_4 = [\frac{5}{\sqrt{2}} \quad -\frac{5}{\sqrt{2}}]$

в

Рис. 3.2. Влияние величины касательного вектора на форму сегмента кубического сплайна

При состыковке сегментов следует соблюдать следующие правила:

1) конец первого сегмента должен служить началом второго;

2) для обеспечения непрерывности первой производной задающие векторы должны иметь одинаковые направления, хотя могут иметь разные длины.

3.2. Формы Безье

Рассмотренные в предыдущем подразделе кубические сплайны неудобны в интерактивной работе и обладают следующими недостатками:

- пользователю трудно с их помощью задавать сегменты кривой,
- задание вектора касательной ненаглядно.

Формы Безье позволяют обеспечить дружественный интерфейс при их реализации. Здесь в качестве граничных условий выступают четыре точки

(рис. 3.3): две концевые вершины (P_1 , P_4) и две опорные (P_2 , P_3). В графическом редакторе пользователь может захватить любую из четырех вершин и, перемещая ее на плоскости, тем самым подобрать нужный вид кривой.

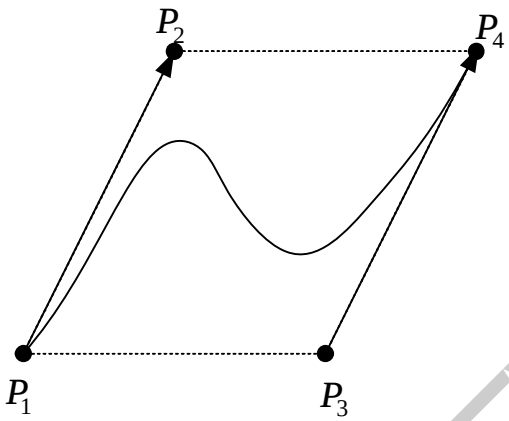


Рис. 3.3. Граничные условия формы Безье

Для определения векторов касательных в форме Безье используются следующие соотношения

$$R_1 = n(P_2 - P_1) = 3(P_2 - P_1),$$

$$R_4 = n(P_4 - P_3) = 3(P_4 - P_3),$$

где n – степень полинома.

Исходя из этого, вектор G_{nx} будет иметь вид

$$G_{nx} = \begin{bmatrix} P_1 \\ P_4 \\ R_1 \\ R_4 \end{bmatrix}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}_x = M_{nb} \cdot G_{bx}. \qquad (3.6)$$

Подставив выражение (3.6) в (3.3), получим

$$C_x = M_n \cdot G_{nx} = M_n \cdot M_{nb} \cdot G_{bx} = M_b \cdot G_{bx}, \qquad (3.7)$$

где $M_b = M_n \cdot M_{nb} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$ –

матрица Безье.

Подставив выражение (3.7) в (3.2), получим

$$x(t) = T \cdot M_b \cdot G_{bx}, \qquad (3.8)$$

$$x(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{1x} \\ P_{2x} \\ P_{3x} \\ P_{4x} \end{bmatrix}.$$

Функция $y(t)$ получается аналогичным способом. Таким образом, полином, который описывает сегмент кривой Безье, будет иметь вид

$$P(t) = \begin{bmatrix} x(t) & y(t) \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{1x} & P_{1y} \\ P_{2x} & P_{2y} \\ P_{3x} & P_{3y} \\ P_{4x} & P_{4y} \end{bmatrix}. \qquad (3.9)$$

Для того чтобы направления векторов касательных совпадали в точке P_4, необходимо, чтобы точки P_3, P_4, P_5 лежали на одной прямой (рис. 3.4).

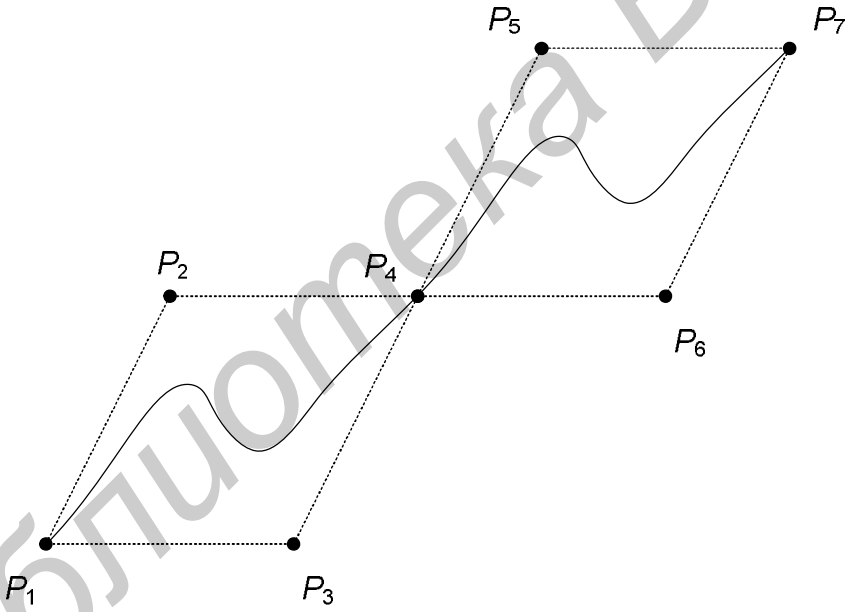


Рис. 3.4. Правило состыковки сегментов для кривых Безье

Пример 6. Построить сегмент кривой Безье для следующих граничных условий:

$P_1 = [0 \quad 5], P_2 = [5 \quad 0], P_3 = [6 \quad 6], P_4 = [3 \quad 4].$

Подставим значения граничных условий в выражение (3.9), получим следующий вид кривой

$$[x(t) \quad y(t)] = [t^3 \quad t^2 \quad t \quad 1] \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & 5 \\ 5 & 0 \\ 6 & 6 \\ 3 & 4 \end{bmatrix} = [t^3 \quad t^2 \quad t \quad 1] \cdot \begin{bmatrix} 0 & -19 \\ -12 & 33 \\ 15 & 15 \\ 0 & 5 \end{bmatrix}.$$

Параметр t изменяется в пределах [0,1]. Примем шаг изменения равный 0.1. В результате получим таблицу значений координат в зависимости от параметра t и по данным значениям построим сегмент кривой (рис. 3.5).

t	$x(t)$	$y(t)$
0.0	0.000	5.000
0.1	1.380	3.811
0.2	2.520	3.168
0.3	3.420	2.957
0.4	4.080	3.064
0.5	4.500	3.375
0.6	4.680	3.776
0.7	4.620	4.153
0.8	4.320	4.392
0.9	3.780	4.379
1.0	3.000	4.000

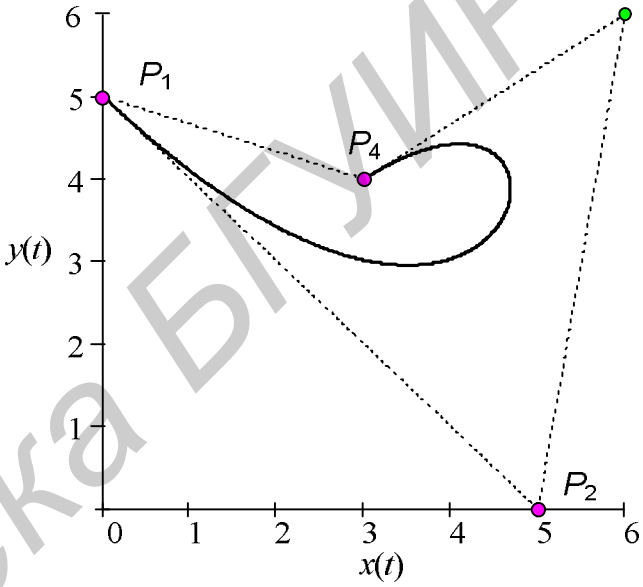


Рис. 3.5. Результат построения сегмента кривой Безье

В заключение отметим некоторые основные свойства кривых Безье:

- степень многочлена, определяющего сегмент кривой, на единицу меньше количества задающих вершин;
- кривая проходит через концевые вершины (P_1 , P_4), а форма кривой зависит от задания опорных и концевых вершин;
- кривая всегда лежит внутри выпуклой оболочки многоугольника, образованной из множества концевых и опорных вершин;
- кривая инвариантна относительно аффинных преобразований.

Ограничение кривых Безье заключается в том, что любая точка на кривой Безье зависит от всех определяющих вершин, поэтому изменение какой-либо одной вершины оказывает влияние на всю кривую. Локальные воздействия на кривую невозможны.

3.3. Сглаживание кривых методом B-сплайнов

Из нескольких возможных способов построения гладких кривых рассмотрим форму B-сплайна. Из заданной последовательности точек выбираются две соседние точки и между ними строится кривая кубического полинома на основе позиций четырех точек: двух уже упомянутых и двух соседних с ними. B-сплайн обеспечивает получение более гладких кривых, чем другие способы сглаживания, за счет того что получаемые кривые не проходят точно через заданные точки.

Математически гладкость кривых выражается в терминах непрерывности параметрических представлений $x(t)$ и $y(t)$ и их производных. Кривые типа B-сплайна обладают свойством непрерывности первой и второй производных в точках стыковки двух соседних сегментов кривой.

На рис. 3.6 можно видеть, как выглядят кривые, если их нулевая, первая и вторая производные не являются непрерывными в некоторой точке.

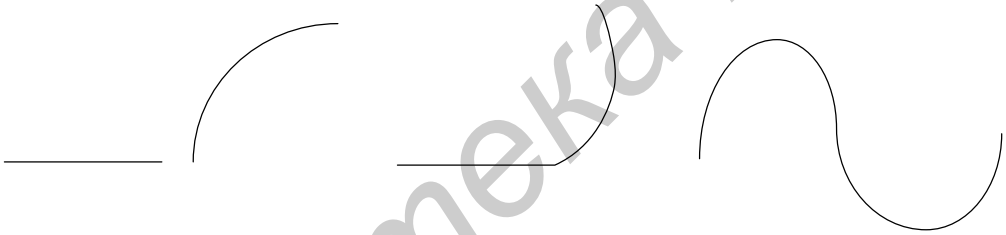


Рис. 3.6. Отсутствие свойства непрерывности функции, ее первой и второй производной

Рассмотрим метод построения B-сплайна.

Будем использовать параметрическое представление кривых. Любая точка части кривой между двумя заданными последовательными точками P и Q будет иметь координаты $x(t)$ и $y(t)$, где t – время.

Если имеются заданные вершины

$$P_1(x_1,y_1) \quad P_2(x_2,y_2) \quad \ldots \quad P_{n-1}(x_{n-1},y_{n-1}) \quad P_n(x_n,y_n),$$

тогда необходимо расширить множество заданных вершин точками $P_0(x_0,y_0)$, $P_{n+1}(x_{n+1},y_{n+1})$ – соседними точками для первого и последнего сегмента сплайна. Сегмент кривой B-сплайна между двумя последовательными

точками P_i и P_{i+1} $(1 \le i \le n-1)$ получится путем вычисления функций $x(t)$ и $y(t)$ для изменения t от 0 до 1:

$$x(t) = \{(a_3 \cdot t + a_2) \cdot t + a_1\}t + a_0,$$
$$y(t) = \{(b_3 \cdot t + b_2) \cdot t + b_1\}t + b_0.$$

Эти уравнения содержат следующие коэффициенты:

$$a_3 = (-x_{i-1} + 3 \cdot x_i - 3 \cdot x_{i+1} + x_{i+2})/6,$$
$$a_2 = (x_{i-1} - 2 \cdot x_i + x_{i+1})/2,$$
$$a_1 = (-x_{i-1} + x_{i+1})/2,$$
$$a_0 = (x_{i-1} + 4 \cdot x_i + x_{i+1})/6,$$

а коэффициенты b_3, b_2, b_1, b_0 вычисляются по значениям $y_{i-1}, y_i, y_{i+1}, y_{i+2}$ аналогичным образом.

Коэффициенты a_3, a_2, a_1, a_0 вычисляются только однажды для каждого сегмента кривой, что очень важно, так как на каждом сегменте кривой может вычисляться большое число промежуточных точек $x(t)$ и $y(t)$.

В матричном виде будем иметь

$x(t) = T \cdot M_S \cdot G_{Sx}$ (функция $y(t)$ обладает аналогичными свойствами),

где $M_S = \frac{1}{6}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$.

Для каждой пары точек P_i и P_{i+1}, определяющих сегмент, используется вектор

$$G_{sx} = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}, \quad 1 \le i \le n-1, \quad 0 \le t < 1,$$

где i – счетчик сегментов кривой, а n – количество вершин.

Построить замкнутую B-сплайновую кривую можно, дополнив набор базовых точек из n штук следующими точками: $P_{n+1} = P_0, P_{n+2} = P_1, P_{n+3} = P_2$.

Таким образом, полином, который описывает сегмент B-сплайна, будет иметь вид

$$P(t)=\begin{bmatrix}x(t) & y(t)\end{bmatrix}=\frac{1}{6}\begin{bmatrix}t^3 & t^2 & t & 1\end{bmatrix}\cdot\begin{bmatrix}-1 & 3 & -3 & 1\\ 3 & -6 & 3 & 0\\ -3 & 0 & 3 & 0\\ 1 & 4 & 1 & 0\end{bmatrix}\cdot\begin{bmatrix}P_{(i-1)x} & P_{(i-1)y}\\ P_{ix} & P_{iy}\\ P_{(i+1)x} & P_{(i+1)y}\\ R_{(i+2)x} & R_{(i+2)y}\end{bmatrix}, \quad (3.10)$$

$$1\le i\le n-1, \quad 0\le t<1,$$

где i – счетчик сегментов кривой, n – количество вершин, $(n-1)$– количество сегментов сплайна.

Пример 7. Построить замкнутый B-сплайн. Вершины многоугольника: $(2,0),(4,0),(4,2),(4,4),(2,4),(0,4),(0,2),(0,0)$.

В данном случае необходимо дополнить множество вершин точками $P_{n+1}=P_0, P_{n+2}=P_1, P_{n+3}=P_2$. В результате получим следующие граничные условия: $P_0=[2\quad 0]$, $P_1=[4\quad 0]$, $P_2=[4\quad 2]$, $P_3=[4\quad 4]$, $P_4=[2\quad 4]$, $P_5=[0\quad 4]$, $P_6=[0\quad 2]$, $P_7=[0\quad 0]$, $P_8=[2\quad 0]$, $P_9=[4\quad 0]$, $P_{10}=[4\quad 2]$.

Таким образом, B-сплайн состоит из восьми сегментов. Подставим значения граничных условий в выражение (3.10) для каждого из восьми сегментов. Параметр t изменяется в пределах [0, 1]. Примем шаг изменения равный 0.1. В результате получим таблицу значений координат для каждого сегмента B-сплайна в зависимости от параметра t и по данным значениям построим B-сплайн (рис. 3.7).

	Сегмент 1: P_0, P_1, P_2, P_3		Сегмент 2: P_1, P_2, P_3, P_4		Сегмент 3: P_2, P_3, P_4, P_5		Сегмент 4: P_3, P_4, P_5, P_6	
t	$x(t)$	$y(t)$	$x(t)$	$y(t)$	$x(t)$	$y(t)$	$x(t)$	$Y(t)$
0.0	3.667	0.333	4.000	2.000	3.667	3.667	2.000	4.000
0.1	3.757	0.443	4.000	2.200	3.557	3.757	1.800	4.000
0.2	3.829	0.571	3.997	2.397	3.429	3.829	1.603	3.997
0.3	3.886	0.714	3.991	2.591	3.286	3.886	1.409	3.991
0.4	3.928	0.872	3.979	2.779	3.128	3.928	1.221	3.979
0.5	3.958	1.042	3.958	2.958	2.958	3.958	1.042	3.958
0.6	3.979	1.221	3.928	3.128	2.779	3.979	0.872	3.928
0.7	3.991	1.409	3.886	3.286	2.591	3.991	0.714	3.886
0.8	3.997	1.603	3.829	3.429	2.397	3.997	0.571	3.829
0.9	4.000	1.800	3.757	3.557	2.200	4.000	0.443	3.757
1.0	4.000	2.000	3.667	3.667	2.000	4.000	0.333	3.667

	Сегмент 5: P_4, P_5, P_6, P_7		Сегмент 6: P_5, P_6, P_7, P_0		Сегмент 7: P_6, P_7, P_0, P_1		Сегмент 8: P_7, P_0, P_1, P_2	
t	$x(t)$	$y(t)$	$x(t)$	$y(t)$	$x(t)$	$y(t)$	$x(t)$	$Y(t)$
0.0	0.333	3.667	0.000	2.000	0.333	0.333	2.000	0.000
0.1	0.243	3.557	0.000	1.800	0.443	0.243	2.200	0.000
0.2	0.171	3.429	0.003	1.603	0.571	0.171	2.397	0.003
0.3	0.114	3.286	0.009	1.409	0.714	0.114	2.591	0.009
0.4	0.072	3.128	0.021	1.221	0.872	0.072	2.779	0.021
0.5	0.042	2.958	0.042	1.042	1.042	0.042	2.958	0.042
0.6	0.021	2.779	0.072	0.872	1.221	0.021	3.128	0.072
0.7	0.009	2.591	0.114	0.714	1.409	0.009	3.286	0.114
0.8	0.003	2.397	0.171	0.571	1.603	0.003	3.429	0.171
0.9	0.000	2.200	0.243	0.443	1.800	0.000	3.557	0.243
1.0	0.000	2.000	0.333	0.333	2.000	0.000	3.667	0.333

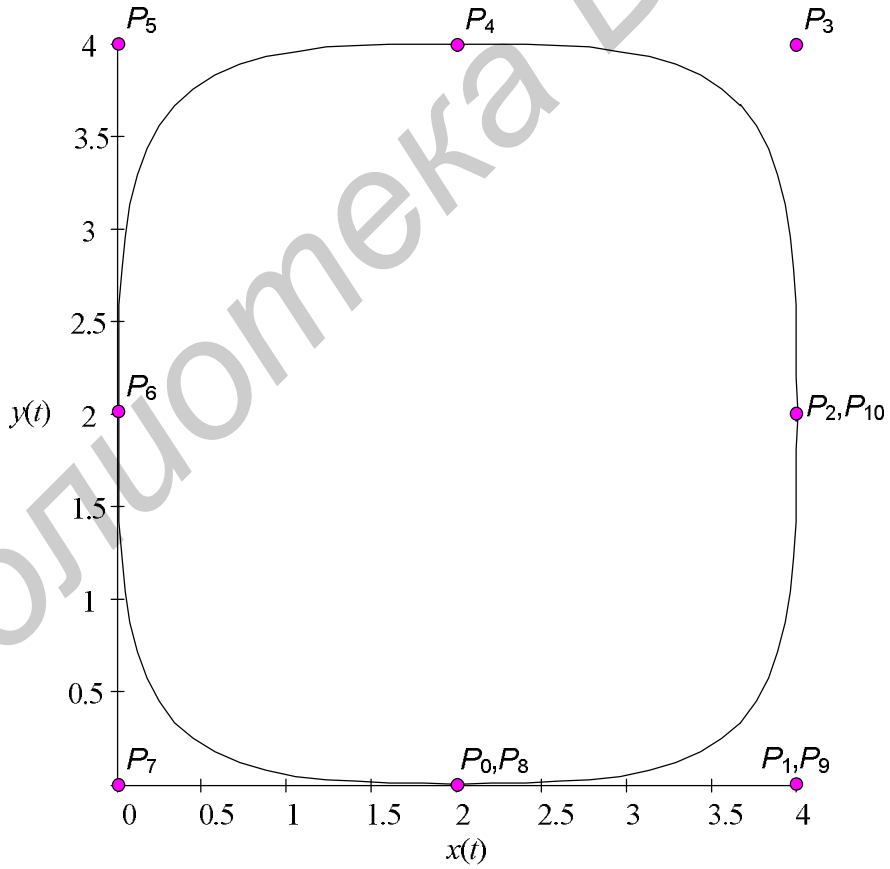


Рис. 3.7. Результат построения B-сплайна

Упражнения для самостоятельной работы

Упражнение 3.1. Постройте сегмент кривой, используя форму Эрмита для следующих граничных условий:

1. $P_1(0, 0)$, $P_4(4, 0)$, $R_1(1, 1)$, $R_4(1, 0)$.
2. $P_1(0, 0)$, $P_4(3, 5)$, $R_1(1, 1)$, $R_4(0, -1)$.
3. $P_1(0, 0)$, $P_4(3, 0)$, $R_1(0, 1)$, $R_4(1, 0)$.
4. $P_1(0, 0)$, $P_4(4, 4)$, $R_1(0, 1)$, $R_4(0, -1)$.
5. $P_1(0, 0)$, $P_4(4, 4)$, $R_1(0, 1)$, $R_4(0, -1)$.
6. $P_1(0, 0)$, $P_4(3, 5)$, $R_1(1, 1)$, $R_4(0, -1)$.
7. $P_1(0, 0)$, $P_4(4, 0)$, $R_1(0, 1)$, $R_4(1, 0)$.
8. $P_1(0, 0)$, $P_4(4, 0)$, $R_1(1, 1)$, $R_4(1, 1)$.

Упражнение 3.2. Постройте сегмент кривой Безье для следующих граничных условий:

1. $P_1(0, 0)$, $P_2(3, 7)$, $P_3(2, 0)$, $P_4(5, 5)$.
2. $P_1(0, 0)$, $P_2(5, 2)$, $P_3(2, 5)$, $P_4(1, 0)$.
3. $P_1(0, 0)$, $P_2(1, 5)$, $P_3(2, 0)$, $P_4(3, 3)$.
4. $P_1(0, 0)$, $P_2(1, 1)$, $P_3(2, 1)$, $P_4(1, 0)$.
5. $P_1(0, 0)$, $P_2(0, 1)$, $P_3(1, 0)$, $P_4(2, 1)$.
6. $P_1(0, 0)$, $P_2(1, 0)$, $P_3(2, 0)$, $P_4(2, 1)$.
7. $P_1(0, 0)$, $P_2(10, 1)$, $P_3(3, 8)$, $P_4(5, 0)$.

Упражнение 3.3. Постройте B-сплайн по следующему множеству исходных точек:

1. (0, 0), (5, 5), (2, 1), (3, 7).
2. (0, 0), (3, 3), (2, 1), (3, 6).
3. (0, 0), (5, 4), (2, 1), (3, 6).
4. (0, 0), (1, 2), (3, 3), (2, 1).
5. (0, 0), (1, 3), (2, 1), (3, 3).
6. (0, 0), (3, 3), (1, 2), (2, 2).
7. (0, 0), (1, 1), (2, 1), (3, 3).

Лабораторная работа № 3

Разработать элементарный графический редактор, реализующий построение параметрических кривых, используя форму Эрмита, форму Безье и B-сплайн. Выбор метода задается из пункта меню и доступен через панель инструментов «Кривые». В редакторе должен быть предусмотрен режим корректировки опорных точек и состыковки сегментов. В программной реализации необходимо реализовать базовые функции матричных вычислений.

ЛИТЕРАТУРА

1. Аммерал, Л. Машинная графика на персональных компьютерах / Л. Аммерал; пер. с англ. – М. : Сол Систем, 1992. – 232 с.
2. Аммерал, Л. Принципы программирования в машинной графике / Л. Аммерал; пер. с англ. – М. : Сол Систем, 1992. – 224 с.
3. Блинова, Т. А. Компьютерная графика : учеб. пособие / Т. А. Блинова, В. Н. Порев; под ред. В. Н. Порева. – Киев : Юниор, 2006. – 520 с.
4. Гилой, В. Интерактивная машинная графика : структуры данных, алгоритмы, языки / В. Гилой; пер. с англ. – М. : Мир, 1981. – 384 с.
5. Кравченя, Э. М. Компьютерная графика : учеб. пособие / Э. М. Кравченя, Т. И. Абрагимович. – Минск : Новое знание, 2006. – 248 с.
6. Павлидис, Т. Алгоритмы машинной графики и обработка изображений / Т. Павлидис; пер. с англ. – М. : Радио и связь, 1986. – 400 с.
7. Петров, М. Н. Компьютерная графика : учеб. пособие для вузов / М. Н. Петров, В. П. Молочков. – СПб. : Питер, 2002. – 735 с.
8. Поляков, А. Ю. Методы и алгоритмы компьютерной графики в примерах на Visual C++ / А. Ю. Поляков, В. А. Брусенцев. – 2-е изд. – СПб. : BHV-Петербург, 2003. – 545 с.
9. Пореев, В. Н. Компьютерная графика / В. Н. Пореев. – СПб. : БХВ-Петербург, 2002. – 432 с.
10. Рейнбоу, В. Компьютерная графика : [пер. с англ.] / В. Рейнбоу. – СПб. : Питер, 2003. – 768 с.
11. Роджерс, Д. Алгоритмические основы машинной графики / Д. Роджерс; пер. с англ. – М. : Мир, 1989. – 512 с.
12. Роджерс, Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс; пер. с англ. – М. : Мир, 2001. – 604 с.
13. Шикин, А. В. Компьютерная графика. Полигональные модели / А. В. Шикин, А. В. Боресков. – М. : Диалог-МИФИ, 2000. – 464 с.
14. Фень, Юань Программирование графики для Windows / Юань Фень; пер. с англ. – СПб. : Питер, 2002. – 1072 с.
15. Фоли, Дж. Основы интерактивной машинной графики : в 2 т. / Дж. Фоли, А. Ван Дэм. – М. : Мир, 1985.

Учебное издание

Самодумкин Сергей Александрович

Степанова Маргарита Дмитриевна

Колб Дмитрий Григорьевич

ПРАКТИКУМ ПО КОМПЬЮТЕРНОЙ ГРАФИКЕ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

В 3-х частях

Часть 1

Редактор *Т. П. Андрейченко*
Корректор *М. В. Тезина*
Компьютерная верстка *Е. Н. Мирошниченко*

Подписано в печать Формат 60x84 1/16. Бумага офсетная. Гарнитура «Таймс».
Печать ризографическая. Усл. печ. л. 2,79. Уч.-изд. л. 2,3. Тираж 150 экз. Заказ 643.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6