Алгоритм растеризации для рисования кривых

Разработка мультимедиа и программного обеспечения

Технический центр Вена

Алоис Зингль

Вена, 2016 г.

Абстрактный

Эта работа посвящена растеризации кривых. Этот процесс преобразует непрерывные геометрические кривые векторного формата в изображения дискретных пикселей. Растеризация — фундаментальная задача компьютерной графики. Проблема проиллюстрирована линиями, основанными на устоявшихся методах. Общий метод разработан на основе неявного уравнения кривой. Затем этот принцип доказывается на кругах и эллипсах. Впоследствии алгоритм применяется к более сложным кривым, таким как Безье. Этот алгоритм выбирает ближайший к истинным кривым пиксель сетки и позволяет рисовать сложные кривые почти так же быстро и легко, как простые линии. Рассматриваются возникающие в связи с этим проблемы и намечаются различные пути их решения. Затем метод применяется к квадратичным и кубическим Безье рациональных и рациональных форм, а также к сплайнам. Наконец, установлен общий алгоритм для рисования любых
изгиб.

Ключевые слова: графика, растеризация, кривые, алгоритмы, Безье, сплайн.

укороченная версия

Данная работа посвящена экранированию кривых. Непрерывные геометрические кривые преобразуются из векторного формата в изображения, состоящие из дискретных пикселей.
Растеризация — фундаментальная задача компьютерной графики. Проблема обычно решается с помощью строк, основанных на установленных процедурах. Затем разрабатывается общая процедура с использованием неявного уравнения кривой. Затем этот принцип проверяется на кругах и эллипсах. Затем алгоритм отрабатывается на более сложных кривых, таких как кривые Безье. Алгоритм выбирает ближайший к кривой пиксель и позволяет рисовать сложные кривые почти так же легко и быстро, как простые прямые.
Возникающие проблемы обсуждаются и

разработаны разные решения. Затем метод применяется к квадратному и кубическому Безье в рациональной и рациональной формах, а также к рисованию сплайнов. Наконец, установлен общий алгоритм для кривых произвольной формы

Ключевые слова: графика, растрирование, кривые, алгоритм, Безье, сплайн.

Содержание

1. Введение

Векторная графика используется в компьютерном геометрическом проектировании. В основе векторной графики на геометрических примитивах, таких как точки, линии, круги, эллипсыи кривые Безье [Фоли, 1995]. Однако, чтобыбыть полезной, каждую кривуюнеобходимо один раз растрировать на дисплеях. принтеры, плоттеры, машиныи т. д. Около пятидесятилет назад Дж. Э. Брезенхэм из лабораторий IBM разработал для плоттеров алгоритм растеризации линий [Bresenham 1965].
Проблема заключалась в том, что процессорытого времени не имели ни инструкций по умножениюили делению, ни арифметики с плавающей запятой. Он нашел алгоритм растеризации линии на однородной поверхности. сетка пикселей с использованием целочисленного сложения и вычитания . Позже он расширил этот алгоритм для круги.

Алгоритм этого документа улучшает линейный алгоритм Брезенхэма и расширяет его для эллипсыи кривые Безье.

Особенности алгоритма растеризации:

- Универсальность : этот алгоритм строит линии, круги, эллипсы, кривые Безье и т. д.
- Эффективность : Строит сложные кривые со скоростью рисования линий.
- Простота: пиксельный цикл основан только на сложении целых чисел.
- Точность : нет аппроксимации кривой.
- Гладкость : сглаживание кривых.
- Гибкость : Регулируемая толщина линии

Принцип алгоритма можно использовать для растеризации любой кривой.

Первая глава знакомит с алгоритмами рисования . Общий алгоритм рисования вводится и применяется на линиях. Во второй главе алгоритм отрабатывается на окружностях. и линии. В третьей главе используется алгоритм на основе квадратичных кривых Безье и объясняются возникающие проблемы Разрабатываются различные решения , которые также применяются на рациональных квадратичный Безье в четвертой главе. В пятой главе рассматривается кубическая кривая Безье и разрабатывает алгоритм рисования . Рациональные кубы Безье в шестой главе построены с помощью приближение. В седьмой главе применяется разработанный алгоритм рисования сплайнов. Работа завершается компиляцией алгоритма и объяснением возможных последствий.

Все алгоритмы кривых также содержат пример реализации, чтобы каждый мог протестировать алгоритм немедленно.

1.1 Рисование кривой

Сначала будут приведенынесколько определений и дифференциаций, которые используются на протяжении всей этой работы данный.

Например, возможно несколько представлений, определяющих плоскую кривуюиз двух размеры Некоторые определения лучше подходят для построения алгоритмов, чем другие.

Явная функция кривой:

Явное уравнение определяет одну переменную как функцию другой y = f(x). Это представление, как правило, непригодно для растеризации, поскольку возможно, что функция может иметь более одного значения y для определенного значения x. Круг является примером такого

изгиб. Функция y= $\sqrt{r^2 - x^2}$ определяет только верхнюю половину круга. Целый
поэтому для круга требуется два определения функции.

Неявная функция кривой:

Неявное уравнение кривой представляет собой нулевое множество функции двух переменных f(x,y) = 0. Алгебраические кривые, рассматриваемые в данной работе, могут быть представлены двумерными полиномами от , у = вещественных

f Их с коэффициентами $a_{ij} x^i y^j = 0 [i \; j \; n]$.

Каждая точка (x, y) на кривой удовлетворяет этому уравнению

Максимальное значение n уравнения определяет степень неявной функции.

Функция параметрической кривой:

Параметрическое уравнение кривой представляет собой векторную функцию одной переменной. Точки на кривой определяются значениями двух функций x = fx(t) и y = fy(t) при значениях x параметра t. Ограниченный интервал параметра t определяет ограниченный сегмент кривых.

Некоторые кривые, такие как Безье, легче определить с помощью параметрического представления, чем с помощью других. Это также позволяет быстро вычислить координаты(x, y) на кривой для целей рисования.

Градиент кривых:

Наклон или градиент кривой в точке (x, y) определяется как первая производная функции: dy/dx. Алгоритм рисования может опираться на постоянно повышающуюся или нисходящую кривую Поэтому может возникнуть необходимость разделить кривую при изменении направления рисования. Этими стационарными точками являются максимум и минимум на кривой, где наклон кривой горизонтальный или вертикальный. Эти точки можно вычислить, установив производную функции равной нулю в направлении x или y.

Определенным алгоритмам также требуются разные процедуры для наклонов ниже или выше значения, равного единице. Поскольку, если градиент ниже единицы, всегда происходит x-шаг и необходим условный y-шаг. Если наклон больше единицы, то используется шаг по оси Y и условный шаг по оси X.
необходимый.

Векторная графика по сравнению с пиксельным изображением:

Визуальный мир электронных мультимедиа состоит из двух противоположных областей: обработки изображений и компьютерной графики. [Фоли, 1995]

Одна сторона переносит изображения реального мира в компьютер. Фото- или кинокамерыиспользуются для создания изображений, которые могут быть обработаныкомпьютерами. Другая сторона создает внутри компьютера искусственные изображения и переносит их в реальный мир. Эти изображения созданыс помощью системы автоматизированного проектирования (САПР). [Фоли, 1995]

Изображения реального мира состоят из двумерной матрицыэлементов изображения (пикселей). Каждый пиксель содержит информацию о цвете в определенной позиции. Информация, не зафиксированная во время записи, теряется навсегда. Например, невозможно впоследствии увеличить разрешение изображения, чтобыувеличить детализацию.

Искусственные изображения, генерируемые компьютером, в основном используют геометрические примитивы, такие как точки, линии, площади, объемыи т. д. Векторная графика содержит информацию о положении в двух или трех измерениях, а также такие атрибуты, как цвет линии, толщина, типи т. д. Эта информация не зависит от определенного разрешения. [Фоли, 1995]

Шрифты являются хорошим примером векторной графики. Независимо от того, где выпрочитали эту работу, буквы этого текста состоят из векторной графики и должныбыть растрированыв пиксельные изображения, чтобывымогли можно прочитать текст.

1.2 Растеризация

Векторная графика — это всего лишь числа, обрабатываемые компьютером. Для визуализации векторной графики ее необходимо оцифровать в сетку пикселей. Это преобразование называется растеризацией. Хотя преобразование пиксельных изображений в векторную графику затруднено, другой способ сравнительно прост. Это преимущество, поскольку каждый раз требуется растеризация, чтобысделать числа видимыми. Растеризация необходима для всех устройств вывода, таких как мониторы, проекторы, принтеры, плоттерыи т. д. Поэтому важной целью этой работы является эффективность вычислений. [Фоли, 1995]

Невозможно перечислить все работы, связанные с растеризацией. Несколько недавних публикаций вместе с основными идеями послужили источником вдохновения для возможных алгоритмов.

1.2.1 Сопутствующая работа

Фоли описывает два способа построения параметрической кривой [Foley, 1995]. Первый заключается в итеративной оценке fx(t) и fy(t) для постепенно расположенных значений t. Второй — рекурсивное подразделение до тех пор, пока контрольные точки не окажутся достаточно близко к кривой. Оба метода имеют свои преимущества и недостатки. В этом документе описывается третий способ: преобразование параметрического уравнения кривой в неявное уравнение и рисование кривой путем итеративной оценки неявного уравнения.

Начнем с простой линии. Как можно растрировать линиюот P0 до P1? Пройдя через все x-позиции пикселей линии, y-позиции можно вычислить по формуле

$$y = (x-x0)(y1-y0)/(x1-x0)+y0.$$

У этого метода есть недостаток. Требуется умножение и деление с плавающей запятой. Может показаться, что подсчитать это не так уж и сложно. Но можно ли сделать это более эффективно?

Выражение для расчета положения у содержит отношение наклона Δx/Δy. Вместо дроби можно производить расчет по целым числам числителя и знаменателя. Это решение позволяет избежать вычислений с плавающей запятой.

Каждый шаг x добавляется разница у. Если выражение больше, чем разность по оси x, эта разница вычитается и выполняется шаг по оси y. Этот алгоритм называется алгоритмом Брезенхэма. [Брезенхем, 1965]

Но это решение работает только в том случае, если разность у меньше разницых. В другом случае необходима вторая процедура с обменом координат. Этот алгоритм проходит через все позиции y и вычисляет соответствующие позиции x. Необходимость двух разных процедур для одного и того же алгоритма является препятствием для упрощения и расширения его использования для более сложных кривых.

[Loop et al., 2005] представляет независимый от разрешения алгоритм рисования, который использует программируемое графическое оборудование для выполнения рендеринга. Преимущество этого подхода заключается в том, что сглаживание также может рассчитываться графическим процессором.

Алгоритм этого документа фокусируется на кривых до полиномиальной степени три.
Более высокие степени полинома содержат несколько особых точек или близких их сегментов кривой, которые не могут быть обработаныалгоритмом и требуют специальных решений. Такие кривые можно построить с помощью юалгоритмов субпикселизации, разработанных [Емельяненко, 2007] или дистанционных аппроксимаций в работе [Таубин, 1994].

Такое решение алгоритма рисования аналогично работе [Golipour-Koujali, 2005], Янга [Yang et al., 2000] и Кумара [Kumar et al., 2011]. Основное отличие их работы в том, что вместо восьми разных процедур для каждого октанта направления рисования разрабатывается общий алгоритм для любого направления. Это делает алгоритм более компактным.

1.2.2 Алгоритм средней точки

Должна быть проведена линия на рисунке 1. Pixel P уже установлен. Для упрощения предполагается, что наклон линии находится в диапазоне от нуля до единицы В этих условиях хряду можно установить только два пикселя: PX или PXY. Решение о том, какой из них следует установить, может быть принято по расстоянию линии до центра пикселя. Если линия ближе к Pixel PX, то этот пиксель установлен. В противном случае устанавливается пиксель PXY. Вместо расстояния до линии используется неявная функция кривой f(x,y)=0. Функция равна нулю для точек на линии. Он положителен для точек в верхнем левом углу и отрицателен для точек в правом нижнем углу. Таким образом, критерием может быть значение неявной функции в точке PM точно между PX и PXY. Если функция для точки PM является положительной точкой, устанавливается PXY, если это отрицательная точка, устанавливается PX. Поскольку значение средней точки между двумя пикселями определяется как решение, этот алгоритм называется алгоритмом средней точки. [Фоли, 1995]

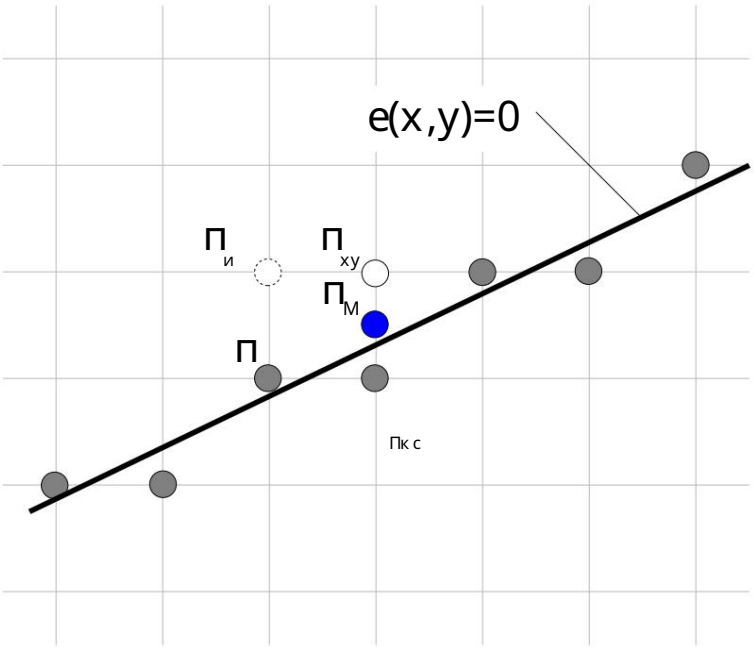


Рисунок 1: Алгоритм средней точки

Этот метод ограничен наклонами от нуля до единицы В случае линии аналогичный алгоритм необходим для наклонов выше единицы, чтобывыбрать между точками PXY и PY.

1.2.3 Алгоритм Горнера

Другой способпостроения сложных кривых —прямое дифференцирование по алгоритму Хорнера [Foley, 1995]. Этот алгоритм вычисляет значение функции, просто добавляя разницу к предыдущему значению f(t+Δt) = f(t)+d. Если функция представляет собой полином первой степени f(t) = a1 t+a0, то разница представляет собой только постоянную величину: d1 = t a1. Для полиномов степени n различия представляют собой последовательное сложение: di = di+di+1. Значения инициализации di могут быть вычислены

разностями функции f(t). Если этот алгоритм применить к параметрическому уравнениюкривой x = fx(t) и y = fy(t), то координатыБезье, например, можно будет вычислить только путем сложений. Проблема этого алгоритма заключается в выборе подходящего размера шага Δt. Если этот шаг слишком велик, несколько пикселей пропускаются, а если он слишком мал, один и тот же пиксель устанавливается несколько раз.

Алгоритм Горнера не ограничивается линиями. Его можно использовать и для других кривых. Нужна неявная функция кривой. Начиная с позиции P, каждый октант направления рисования требует решения, должен ли быть установлен пиксель в одном из восьми соответствующих направлений или нет.
В этом документе каким-то образом применяется алгоритм Хорнера к неявному уравнениюкривой.

Другой способрастрирования кривой —аппроксимация. Кривая разделена на короткие линии, каждая линия строится отдельно. Но аппроксимация также означает выбор одного из двух недостатков. Если аппроксимация должна быть точной, кривуюнеобходимо разделить на множество небольших сегментов. Это вычислительно дорого. С другой стороны, кривая становится резкой, если аппроксимация недостаточно точна. Быстрый и точный алгоритм растеризации
для кривых поэтому желателен.

1.3 Определение проблемы

Неявное уравнение f(x,y) = 0 определяет кривуюот точки P0(x0,y0) до P1(x1,y1). Градиент кривой должен постоянно быть либо положительным, либо отрицательным. Это ограничение решается разделением кривой.

Кривая может быть прямой линией, но также может быть частью эллипса или, например, кривой Безье.

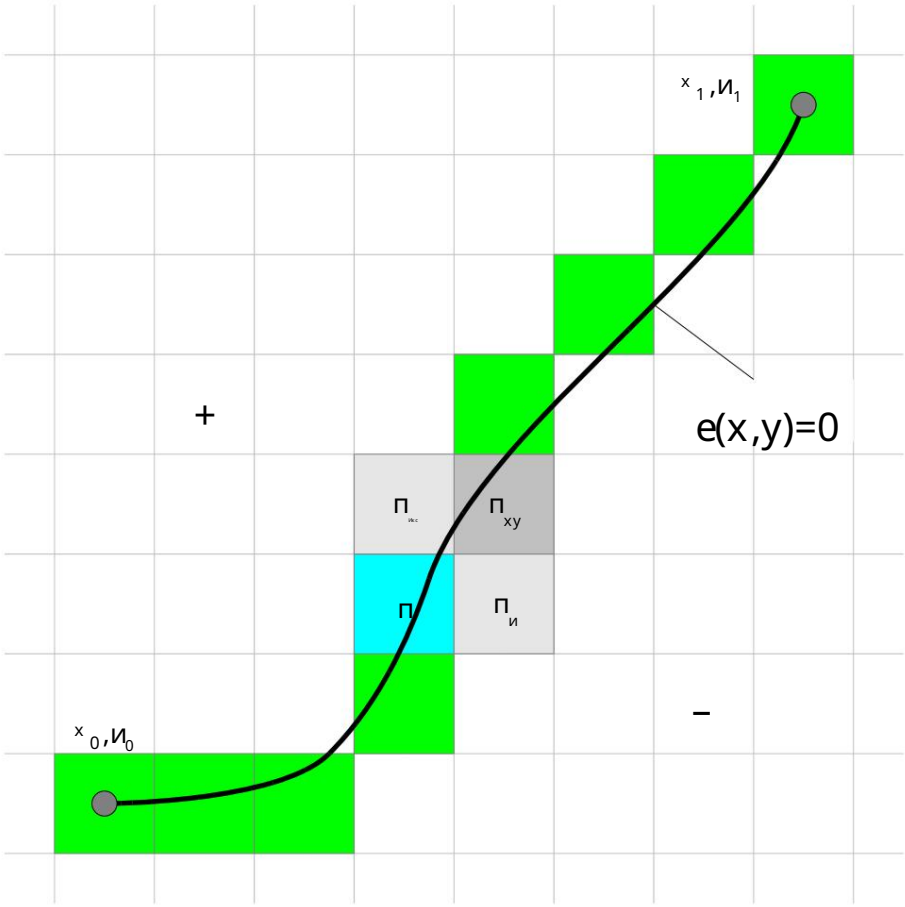


Рисунок 2: Пиксельная сетка с кривой f(x,y)=0

Кривуюна рисунке 2 следует преобразовать в сетку пикселей. Это преобразование называется растеризацией.

Какой пиксель сетки следует установить следующим, чтобынаиболее подходящим образом представить кривуюна рисунке 2?

1.4 Общее решение

Ошибка пикселя р вводится алгоритмом как измерение отклонения пикселя от кривой: e = f(x,y). Значение ошибки равно нулюдля пикселей точно на кривой, положительное для одной стороныкривой и отрицательное для другой стороныкривой. Этот расчет ошибки используется для принятия решения, какой пиксель следует установить следующим. Начиная с пикселя р только между тремя

Возможный пиксель может быть выбран в качестве следующего пикселя из-за положительного градиента: PX или PXY.

Алгоритм начинается с предположения, что следующей будет установлена точка PXY. Итак, если ошибка |exy| точки PXY меньше ошибки |ex| пикселей PX, чем будет увеличено направление x.
Такое же решение рассматривается для направления y. Если ошибка |exy| PXY меньше ошибки |ey | чем направление y будет увеличено. Вот почему на рисунке 2 пиксель над P обозначен PX, а помимо P обозначен PY.

Поскольку предполагается положительный градиент и ошибка на одной стороне кривой будет отрицательной, неравенство ex exy ey всегда будет истинным, что позволяет избежать вычисления абсолютного значения для сравнения. Условия прибавки таковы
сейчас:

если ex + exy > 0, то увеличить x, если ey + exy < 0,
то увеличить y

Преимущество этого подхода в том, что ошибка текущего пикселя уже известна, поэтому необходимо вычислить только разницу с предыдущим пикселем. Это вычисление более эффективно реализовать, чем вычисление всего выражения для каждого пикселя.

Ошибка следующего пикселя должна быть рассчитана для всех трех возможностей фактического пикселя P: ex, ey и exy. Можно ли это еще уменьшить? Если алгоритм отслеживает не ошибку текущего пикселя p, а ошибку exy следующего диагонального пикселя PXY, то необходимо выполнить только два расчета ошибок: ex и ey.
Поскольку ошибка e недоступна, ex и ey должны рассчитываться на один пиксель меньше фактической ошибки.

1.5 Псевдокод алгоритма

Расчет значения ошибки зависит от функции кривой, но условия приращения всегда будут одинаковыми.

```
установите x, y в x0, y0 установите
переменную ошибки exy для цикла P(x0+1,y0+1)

    установить пиксель x,
    y, если ex + exy > 0, затем увеличить x, ошибка субразности, если ey + exy < 0, затем
    увеличить y, добавить ошибку разницы
цикл до конца пикселя
```

Листинг 1: Псевдокод алгоритма

Обратите внимание, что если условие истинно, то погрешность разности необходимо рассчитывать после приращение производится, поскольку вычисление ошибки всегда смотрит на один диагональный пиксель вперед.

Некоторые алгоритмы этом документе содержат много деталей. Не все прямо упомянутые в тексте. Некоторые незначительные решения по реализации можно было бы лучше и более кратко объяснить с помощью юпримера кода. Используется язык программирования C, поскольку его можно легко преобразовать в другие языки. Рисование кривых также является системной задачей, и большинство операционных систем написаны этом языке. Примеры также позволяют сразу протестировать алгоритм.

Разрядность переменных иногда имеет решающее значение и предполагается, что она составляет не менее 16 бит. int, 32 бита для long или float и 64 бита для double.

1.6 Прямые линии

Неявное уравнение прямой линии от точки P(x0,y0) до P(x1,y1):

$$(x1-x0)(y-y0)-(x-x0)(y1-y0) = 0 \quad (1)$$

При определении dx = x1–x0 и dy = y1–y0 ошибка e составляет тогда: e = (y–y0)dx–(x–x0)dy.

Следующие расчеты простые, но немного запутанные из-за индексов и знаков.

Ошибка диагонального шага составляет: exy = (y+1–y0)dx–(x+1–x0)dy = e+dx–dy.

Расчеты погрешностей для направлений x и y составляют: ex = (y+1–y0)dx–(x–x0)dy = exy+dy и ey = (y–y0)dx–(x+1–x0)dy. = exy–dx.

Ошибка для первого шага составляет: e1 = (y0+1–y0)dx–(x0+1–x0)dy = dx–dy.

На рисунке 3 показана линия с dx = 5 и dy = 4. Значение ошибки и голубого пикселя равно 1. Следующие три серых пикселя являются возможными следующими вариантами. Увеличение в направлении x вычитает 4 (dy), увеличение в направлении y добавляет 5 (dx) к значениюошибки. Темно-серый пиксель имеет наименьшее абсолютное значение. Оно рассчитывается заранее, поскольку значение ошибки опережает на один диагональный пиксель.

Поскольку exy+ey = +2–3 = –1 меньше нуля, направление y увеличивается, а dx добавляется к значениюошибки. То же самое делается и для другого направления: exy+ex = +2+6 = +8 больше нуля, поэтому направление x увеличивается, а dy вычитается из значения ошибки.

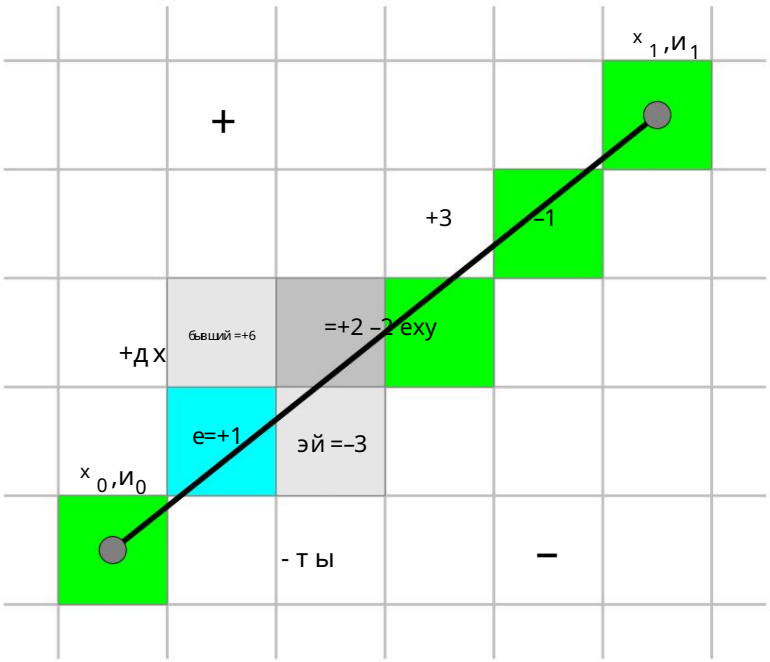


Рисунок 3: строка со значениями ошибок (dx=5, dy=4)

Хотя на рис. 3 кажется, что направления x и y поменялись местами, на самом деле это не так. Меняется только приращение ошибки для условия тестирования.

1.7 Программа для построения линии

Существуют различные возможности обработки отрицательных градиентов или перевернутых линий. Используемое здесь решение состоит в том, чтобы изменить направление шага.

```
voidplotLine (int x0, int y0, int x1, int y1) {

    int dx = abs(x1-x0), sx = x0<x1? 1:-1; int dy = -abs(y1-y0), sy =
    y0<y1? 1:-1; int err = dx+dy, e2;
                                                    /* значение ошибки e_xy */

    для (;;)                                                    /* петля */
        { setPixel(x0,y0); e2 =
        2*ошибка; if (e2
        >= dy) { if (x0 == x1)                          /* e_xy+e_x > 0 */
            сломать; ошибка += dy; x0
            += x;

        } if (e2 <= dx) { if (y0                        /* e_xy+e_y < 0 */
            == y1) Break; ошибка += dx;
            y0 += cy;
        }
    }
}
```

Листинг 2: Программа для построения линии

Алгоритм не имеет аппроксимации. Таким образом, значение ошибки последнего пикселя всегда равно
ровно ноль. Эта версия оптимизирована для проверки конца цикла только в том случае, если соответствующее
направление увеличивается.

Поскольку этот алгоритм работает в направлениях x и y симметрично, ему требуется дополнительный if
условие в пиксельном цикле, что на одно больше, чем в традиционном линейном алгоритме Брезенхема. Это
можно избежать этого дополнительного условия, если заранее известно, что градиент
линия всегда ниже или выше единицы

Программа также элегантно иллюстрирует xy-симметриюлинейного алгоритма Брезенхема.
те же соображения теперь можно применить к кривым более высокой полиномиальной степени.

Благодаря симметрии линии также можно начинать рисунок с обоих концов линии.
линиюи остановитесь посередине. Этот подход может ускорить рисование, но вносит небольшие
нарушения в линии.

1.8 Брезенхем в 3D

Также возможно распространить алгоритм на трехмерное пространство. Обычный подход
различает длину направлений, но симметричное решение более увлекательно.

Поскольку у трехмерное пространство имеет три плоскости, алгоритму Брезенхема требуется переменная ошибки для
каждый из них. Если две плоскости соглашаются двигаться в одном направлении, эта ось увеличивается.
вместе с обновлением соответствующих значений ошибок. Таким образом, алгоритм остается
симметрично, как и в 2D.

```
voidplotLine3D (int x0, int y0 , int z0, int x1, int y1, int z1)
{
    int dx = abs(x1-x0), sx = x0<x1? 1:-1; int dy = abs(y1-
    y0), sy = y0<y1? 1:-1; int dz = abs(z1-z0), sz = z0<z1 ?
    1:-1; int dm = max(dx,dy,dz), я  = dm; /*
    максимальная  разница */

    for (x1 = y1 = z1 = i/2; i-- >= 0; ) { /* цикл */
        setPixel(x0,y0,z0);
        x1 -= дх; если (x1 < 0) { x1 += dm; x0 += x; }
        y1 -= dy; если (y1 < 0) { y1 += dm; y0 += cy; }
        z1 -= дз; если (z1 < 0) { z1 += dm; z0 += сз; }
    }
}
```

Листинг 3: Программа для построения линии в 3D

Этот принцип также можно распространить на более чем три измерения.

Рисунок 4: 3D-линия dx=7, dy=8, dz=9.

2 эллипса

Не проще ли было бынач ать с симмет рич ного к руга вмест о более сложного выражения эллипса?

Было бы, но ненамного. Вын исления для эллипсов не т ак сложны, поэт ому решение можно лег к о адапт ироват ь для к ругов, полагая a = b = r , и эт о не нужно делат ь снова.

При правиль ном выборе сист емык оорд инат эллипс можно описат ь нея вным уравнением: (2)

$$2x62+y2a^{22a-б^2}=0$$

Т ак им образом, уравнение для расч ет а ошибк и для пик селя имеет следующий вид: e = x^{2}62+y2$a^{22a-б^2}$.

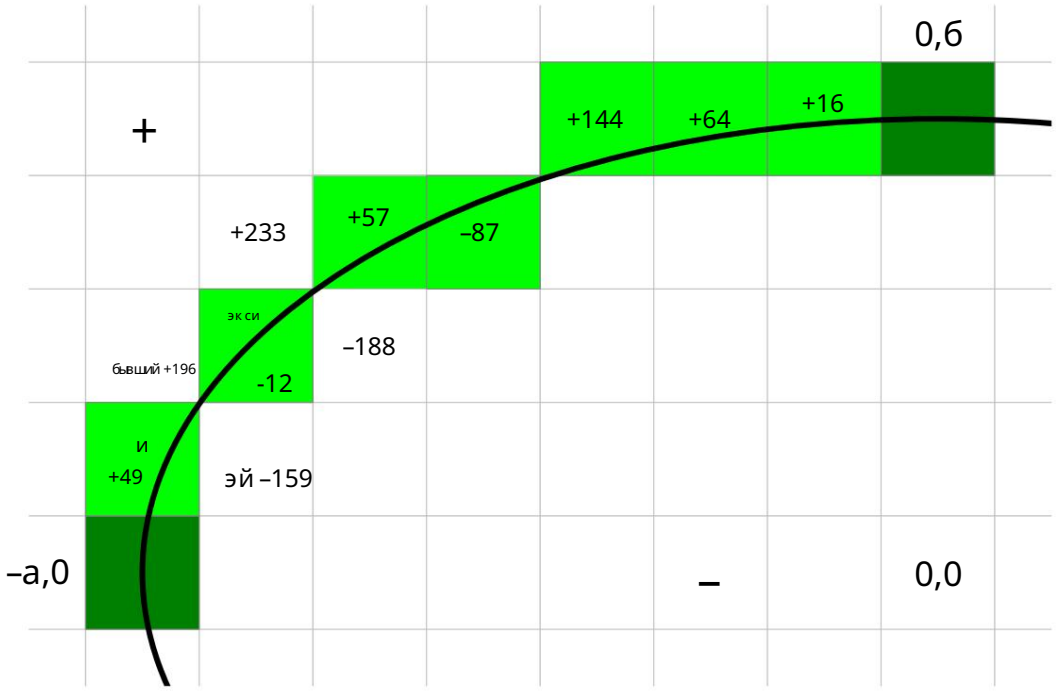


Рисунок 5: к вадрант эллипса со з нач ения ми ошибок для a=7 и b=4.

Ошибк а следующего диаг ональ ного пик селя x+1, y+1 сост авля ет :

$$exy = (x+1)2\ b\ 2+(y+1)2\ a^{22-a}б^{2} = e+(2x+1)b\ 2+(2y+1)a\ 2 .$$

Ошибк а следующего пик селя x+1 сост авля ет : ey = (x+1)2 b 2+y 2 $a^{22a-б^2}$ = exy–(2y+1)a^{2} и для y+1:

$$2экс = x\ 62+(y+1)2\ a^{22-a}б^{2} = exy-(2x+1)b^{2}.$$

Эллипс раз д елен на ч ет ыре к вадрант а. Вт орой к вадрант исполь зует ся из-з а положит ель ного г радиент а. Он нач инает ся в пик селе P(–a,0) и з ак анч ивает ся в P(0,b).

След оват ель но, ошибк а первого пик селя равна: e1 = (–a+1)2 b 2+(0+1)2 a $^{22-a}б^{22=a}$ –б ^{2}(2a–1).

Т еперь мыможем пост роит ь алг орит м.

2.1 Программа для построения эллипса

С помощью этих приготовлений легко записать алгоритм. Но эллипс нуждается в особом лечение, если оно очень плоское.

```
voidplotEllipse (int xm, int ym, int a, int b)
{
    инт ервал x = -a, y = 0; /* II. квадрант слева направо вверх */
    long e2 = (long)b*b, err = x*(2*e2+x)+e2; /* ошибка 1.шага */

    делать {
        setPixel(xm-x, ym+y);                    /* I. Квадрант */
        setPixel(xm+x, ym+y);                    /* II. Квадрант */
        setPixel(xm+x, ym-y);                    /* III. Квадрант */
        setPixel(xm-x, ym-y); e2 =               /* IV. Квадрант */
        2*ошибка;
        если (e2 >= (x*2+1)*(long)b*b)           /* e_xy+e_x > 0 */
            ошибка += (++x*2+1)*(long)b*b;
        если (e2 <= (y*2+1)*(long)a*a)           /* e_xy+e_y < 0 */
            ошибка += (++y*2+1)*(long)a*a;
    } Пока (x <= 0);

    while (y++ <b) { setPixel(xm,        /* досрочная остановка плоских эллипсов a=1, */
        ym+y); setPixel(xm, ym-y);                /* -> завершить кончик эллипса */

    }
}
```

Листинг 4. Простая программа для построения эллипса

Алгоритм останавливается слишком рано, когда радиус эллипса а становится равным единице. В таких случаях стратегия просмотра вперед терпит неудачу, поскольку она проверяет пиксель соседнего квадранта в точке конец. В нормальных условиях это не имеет значения, поскольку у эллипс уже готов. Но для a = 1, алгоритм должен завершить вершину эллипса дополнительным циклом (строки 18-21 в листинг 3).

Алгоритм можно объединить, чтобынарисовать четыре последовательных квадранта эллипса: что необходимо для плоттеров. Таким образом также можно нарисовать только определеннуюдугу эллипс от угла α до β. Необходимо рассчитать только начальное положение и значения ошибок. иначе.

Ценность ошибки может стать огромной. Его переменные (и сравнение с ними) должны иметь возможность хранить тройной размер слова радиусов a, b, чтобыизбежать переполнения. (Если a,b имеют 16 бит тогда err должен иметь как минимум 48 бит.)

2.2 Оптимизированная программа для построения эллипса

Алгоритм можно дополнительно оптимизировать по скорости, введя два дополнительных приращения переменные.

```
voidplotOptimizedEllipse (int xm, int ym, int a, int b)
{
    длинный x = -a, y = 0; /* II. квадрант слева направо вверх */
    длинный e2 = b, dx = (1+2*x)*e2*e2; /* приращение ошибки */
    long dy = x*x, err = dx+dy; /* ошибка 1.шага */

    делать {
        setPixel(xm-x, ym+y);                    /* I. Квадрант */
        setPixel(xm+x, ym+y);                    /* II. Квадрант */
        setPixel(xm+x, ym-y);                    /* III. Квадрант */
        setPixel(xm-x, ym-y);                    /* IV. Квадрант */
        e2 = 2*ошибка;
        если (e2 >= dx) { x++; ошибка += dx += 2*(long)b*b; } if (e2 <= dy) { y+   /* x шаг */
        +; err += dy += 2*(long)a*a; } } Пока (x <= 0);                           /* шаг */

    while (y++ <b)                  /* до ранней остановки для плоских эллипсов с a=1, */
        { setPixel(xm, ym+y);                    /* -> завершить кончик эллипса */
        setPixel(xm, ym-y);
    }
}
```

Листинг 5: Оптимизированная программа для построения эллипса

Конечно, уместно также ввести переменные для константы из $2b^2$ и $2a^2$.

Этот алгоритм делает рисование эллипса таким же простым, как рисование линии: только целые числа. необходимо сложение и вычитание.

Алгоритм не выполняет никакой аппроксимации. Значение ошибки последнего пикселя будет всегда быть ровно нулем.

2.3 Растеризация кругов

Предыдущий алгоритм можно изменить, чтобы нарисовать круг, установив a = b = r. Расчет значения ошибки можно упростить, разделив его на r^2

Но некоторые круги, подобные рисунку 6, выглядят странно с дополнительными точками. Что-то не так с алгоритмом?

Когда x равно y, затрагиваются только четыре пикселя по диагонали 45 градусов.

Более внимательное изучение значений ошибок на рисунке 6 показывает, что рассматриваемые точки имеют более низкое абсолютное значение, чем соседний пиксель.

С точки зрения алгоритма необходимо установить четыре подозрительных пикселя, поскольку они расположены ближе всего к кругу. Проблема возникает, поскольку это точка, в которой градиент круга меняется с уровня ниже 45 градусов на уровень выше 45 градусов. По этой причине необходимо также установить соседние пикселя, поскольку они имеют более низкое значение ошибки, чем альтернативы. Так что с алгоритмом все в порядке, просто неудачное математическое совпадение.

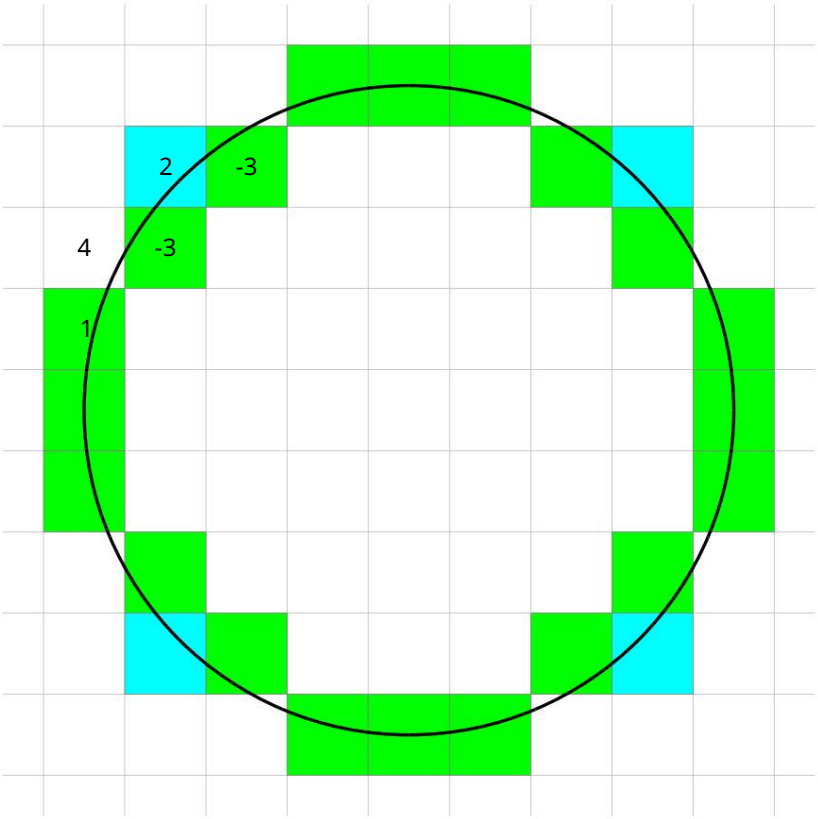


Рисунок 6: Ложный пиксель на круге радиуса 4.

В каких кругах появляются ложные пикселя? Любопытная последовательность чисел радиусов: 4, 11, 134, 373, 4552,...

Согласно Интернет-энциклопедии целочисленных последовательностей (http://oeis.org/A055979), $3\overline{8n12}3\overline{8n}3\overline{8n28}$ правило для нечетного n составляет: $\frac{3\overline{8n12}3\overline{8n}3\overline{8n28}}{\overline{n}}$ и для

даже n: $\frac{3\overline{8n13}\overline{8n123}\overline{8n13}\overline{8}}{\overline{n168}}$. (3)

Можно ли избежать этих паразитных пикселей, хотя они появляются редко?

Одним из простых способов было бы добавить постоянное значение единица к переменной err при инициализации, сделав все радиусы немного меньшими. Но это меняет и некоторые другие круги, особенно маленькие, которые тогда выглядят странно.

В обычных случаях эти дополнительные пиксели вряд ли будут заметны

Эта проблема возникнет и на других кривых.

Одной из возможностей избежать появления нежелательных пикселей является включение дополнительной проверки «ложных пикселей» при увеличении у. Эти дополнительные пиксели появляются на шагах у, когда не выполняется второй шаг у (и шаг х не происходит). Дополнительная проверка ошибок с просмотром на один пиксель вперед позволяет избежать ложного пикселя.

```
voidplotCircle (int xm, int ym, int r) {

    int x = -r, y = 0, err = 2-2*r; /* слева направо вверх */ do { setPixel(xm-x, ym+y); setPixel(xm-
    y, ym-
        x); setPixel(xm+x, ym-y);                    /* I. Квадрант +x +y */ /* II.
        setPixel(xm+y, ym+x);                        Квадрант -x +y */ /* III. Квадрант
                                                     -x -y */ /* IV. Квадрант +x -y */

        r = ошибка;
        если (r <= y) err += ++y*2+1; если (r > x                    /* e_xy+e_y < 0 */ /*
        || err > y) err += ++x*2+1;          e_xy+e_x > 0 или нет второго y-шага */ /* -> x-шаг
                                                                     сейчас */
    } Пока (x <0);
}
```

Листинг 6. Программа Circle, позволяющая избежать ложных пикселей

В отличие от эллипса, алгоритм круга также избегает двойной установки определенных пикселей. В дальнейшем его можно изменить на восемь октантов, установив 8 пикселей на цикл и очень похоже на другие алгоритмы круга.

2.4 Квадратура эллипса

Иногда требуется алгоритм для построения кругов или эллипсов, в которых вместо центра и радиуса указаны углы окружающего прямоугольника. Это также будет включать в себя круги или эллипсы диаметром нечетных пикселей, чего предыдущие алгоритмы не могли сделать.

Алгоритм должен рассчитывать на сетку двойного разрешения, чтобы построить такие эллипсы. На этой сетке алгоритм всегда делает двойные шаги. Если радиус b имеет долю½, то направление у начинается со смещения yb, равного единице; если это целое число, смещение равно нулю

Ошибка следующего диагонального пикселя x+2, y+2 составляет: exy = (x+2)2 b 2+(y+2)2 a $^{2\,2\,-a\,б^{2}}$.

Ошибка следующего пикселя x+2 составляет: ey = (x+2)2 b 2+y 2 b a 2 2 a – б 2 = exy–4(y+1)a 2 и для y+2: 2 экс = x 2+(y+2)2 a 2 2 –a б 2 = exy –4(x+1)b 2.

Следовательно, ошибка первого пикселя равна: e1 = (–a+2)2 b 2+(yb+2)2 a 2 2 –a б 2 = (yb+2)2 a 2 –4(a–1)б 2.

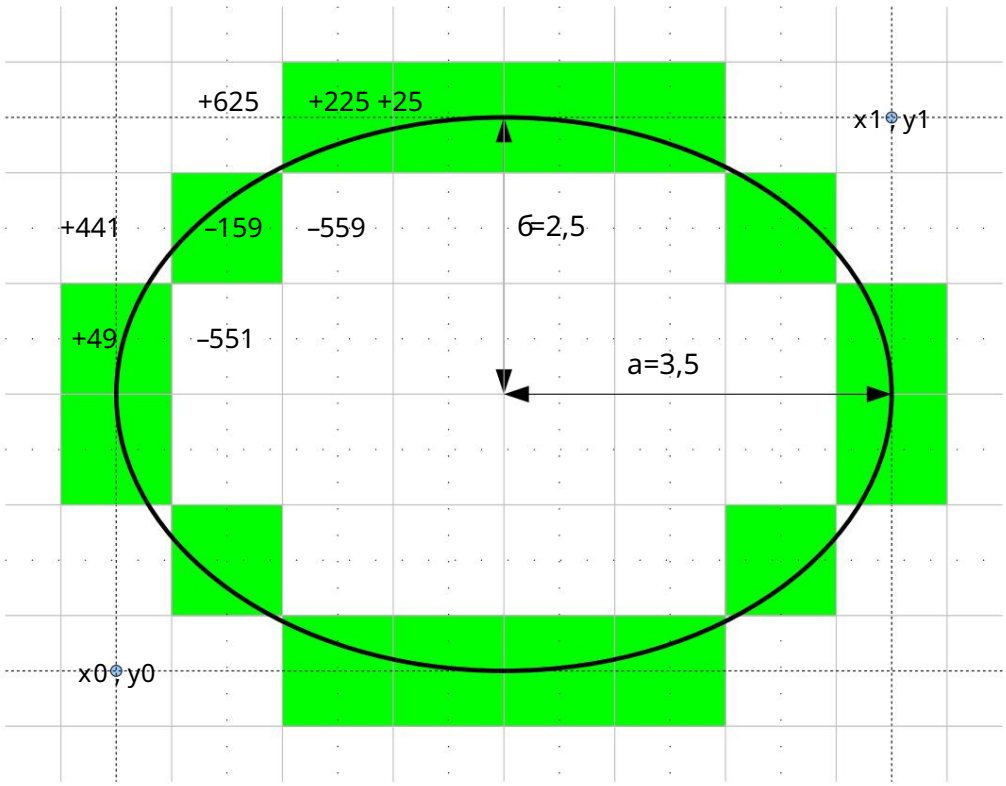


Рисунок 7: эллипс, заключенный в прямоугольник размером 7x5 пикселей.

2.5 Программа для эллипса внутри прямоугольника

Это оптимизированная версия для построения эллипса внутри указанного прямоугольника. Вместо 64-битного целого числа он использует арифметику с плавающей запятой, чтобы избежать переполнения при вычислении ошибок. Он также использует дополнительную проверку x-шага, чтобы избежать появления паразитных пикселей.

```
voidplotEllipseRect (int x0, int y0, int x1, int y1)
{                                       /* прямоугольный параметр, заключающий в себя эллипс */
    long a = abs(x1-x0), b = abs(y1-y0), b1 = b&1; /* диаметр */
    двойной dx = 4*(1.0-a)*b*b, dy = 4*(b1+1)*a*a; /* приращение ошибки */
    двойная ошибка = dx+dy+b1*a*a, e2; /* ошибка 1.шага */

    если (x0 > x1) { x0 = x1; x1 += a; } /* если вывьвается с переставленными точками */
    если (y0 > y1) y0 = y1; /* .. обменивает их */
    y0 += (б+1)/2; y1 = y0–b1; /* начальный пиксель */
    a = 8*a*a; б1 = 8*б*б;

    делать {
        setPixel(x1, y0);                                   /* I. Квадрант */
        setPixel(x0, y0);                                   /* II. Квадрант */
        setPixel(x0, y1);                                   /* III. Квадрант */
        setPixel(x1, y1); e2 =                              /* IV. Квадрант */
        2*ошибка;
        если (e2 <= dy) { y0++; y1--; ошибка += dy += a; } /* шаг */
        if (e2 >= dx || 2*err > dy) { x0++; x1--; err += dx += b1;} /* x */
    } Пока (x0 <= x1);

    в то время как (y0-y1 <= b) {           /* до ранней остановки плоских эллипсов a=1 */
        setPixel(x0-1, y0);                                 /* -> завершить кончик эллипса */
        setPixel(x1+1, y0++);
        setPixel(x0-1, y1);
        setPixel(x1+1, y1--);
    }
}
```

Листинг 7. Программа для построения эллипса, заключенного в прямоугольник.

Этот алгоритм работает для всех значений x0, y0, x1 и y1.

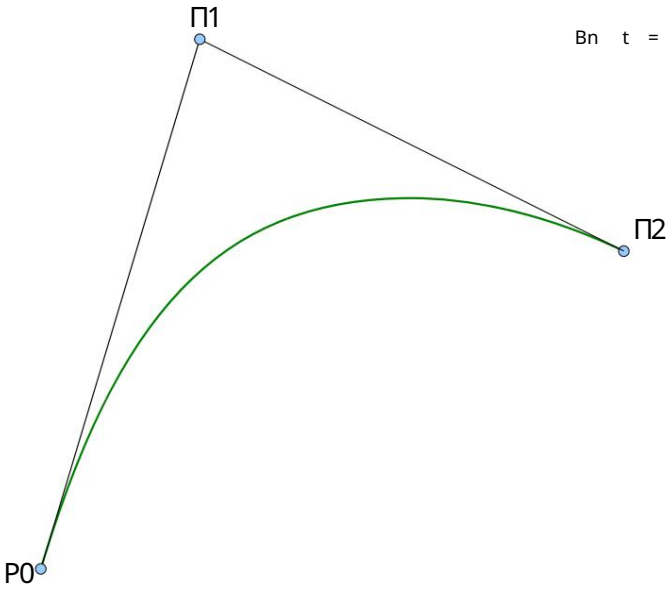
Алгоритм для повернутых эллипсов будет разработан позже, поскольку у алгоритма прямого рисования выполняется к проблемам, но может быть реализовано с помощью юрационального Безье.

3 квадратичные кривые Безье

Концепция универсальных кривых была независимо разработана французскими инженерами Пьером Этьеном Безье из Renault и Полем де Фаже де Кастельжау из Citroën с поя влением в автомобильной промышленности компьютерного производства для проектирования автомобильных кузовов. [Безье, 1986] [Кастельжо, 1963]

Кривые Безье состоят из набора контрольных точек. Количество точек определяет порядок Кривая .

Общее уравнение Безье порядка n в параметрической форме с учетом n +1 точки Pi определяется как [Marsh, 2005, p. 135]



$$Bn\ t = \sum_{i=0}^{n} ni\ 1\ t^{n-i}\ t^{i}\ Pi \tag{4}$$

Рисунок 8: Кривая Безье степени 2.

Это прямая для порядка n = 1. Для порядка n = 2 это квадратичная кривая Безье

$$B2\ t = 1\ t^{2}\ P0\ \ 2\ \ 1\ t\ \ t\ P1\ \ t^{2}\ П2\ . \tag{5}$$

Для алгоритма необходимо конечное неявное уравнение кривой Безье.
Общее неявное уравнение степени 2 составляет :

$$A\,x^{2}\ \ 2\,B\,x\,y\ \ C\,y^{2}\ \ 2\,D\,x\ \ 2\,E\,y\ \ F=0.$$

Это уравнение имеет шесть неизвестных коэффициентов, поэтому для вывода неизвестных необходимышесть линейно независимых уравнений. Если F не равно нулю, уравнение можно

разделить на F: ax 2-2b x y-c y 2-2d x-2e y-1=0 , оставив пять неизвестных.

Два из них можно получить , установив x = x0 y = y0 и x = x2 y

в хотя y2: 2 2 b x0 y0 c y0 2 2 d x0 2e y0= 1 и a x2 2 2b 2 2 d x2 2 e y2= 1. x2 y2 c y2

Третьюможно получить из параметрической формы, установив t = ½:

$$B2 \quad \frac{1}{4}\left(P0 \quad 2P1 \quad P2\right)$$

$$a \quad x0 \quad 2x1 \quad x2 \quad 2 \quad 2b \quad x0 \quad 2x1 \quad x2 \quad y0 \quad 2y1 \quad 2 \quad c \quad y0 \quad 2y1 \quad y2 \quad 2 \quad \ldots \qquad (6)$$

$$\ldots \quad 8d \quad x0 \quad 2x1 \quad x2 \quad 8e \quad y0 \quad 2y1 \quad y2 \quad = \quad 16$$

Последние две неизвестные вычисляются с помощью производной неявного уравнения:

$$, y = \quad 2ax \quad 2by \quad 2d \quad x \;, 2bx \quad 2cy \quad 2e$$

По градиентам в двух точках P0: $\frac{x0 \quad x1}{y0 \quad y1} \text{"="} \frac{ax0 \quad by0 \quad d}{бx0 \quad cy0 \quad e}$ и П2:

$\frac{x2\text{-}x1}{y2 \quad y1} \text{"="} \frac{ax2 \quad by2 \quad d}{бx2 \quad cy2 \quad e}$ можно вывести уравнения для двух последних неизвестных:

$$a\,x0 \qquad x2 - x1 \qquad b \quad y2$$

Вычисления неизвестных теперь становятся немного сложнее. Можно ли их упростить? При замене P i=Pi-P1 кривая Безье смещается на смещение –P1. Алгоритму не составит труда вернуть его позже. Таким образом, для упрощения этого вычисления предполагается, что точка P1 находится в начале координат: x 1= y1=0, а значения x1 и y1 вычитаются из других точек: x i=xi x1. Систему пяти линейных уравнений теперь можно было записать в виде матричного уравнения

$$\begin{bmatrix} x_0^2 & 2\,x0\,y0 & y0^2 & 2\,x0 & 2\,y0 \\ x_2^2 & 2\,x2\,y2 & y2^2 & 8\,x0 & 8\,y \\ x0\,x\,2 & 2\,x\,0 \quad x2 & y\,0 \quad y2 & y0\,y\,2 & 0 \quad y2 \\ x_0^2 & 2\,x0\,y0 & y0^2 & x0 & y0 \\ x_2^2 & 2\,x2\,y2 & y2^2 & x2 & y2 \end{bmatrix} \text{"="} \begin{bmatrix} 1 \\ 16 \\ x0\,x\,2 \\ 0 \\ 0 \end{bmatrix} \text{"="} \begin{bmatrix} -10 & 1 \\ 16 & 0 \end{bmatrix} \qquad (7)$$

Это матричное уравнение можно решить, как показано ниже:

$$A = \quad y0 \quad y \quad 2^2 \;, B = \quad x \quad 0 \quad x \quad 2 \quad y0 \quad y2 \;, C = \quad x0 \quad x \quad 2^2 \;,$$

$$D = \quad y0 \quad y2 \quad x \quad 0 \quad y2 \quad x2 \quad y0 \;, E = \quad x0 \quad x \quad 2 \quad x \quad 0 \quad y2 \quad x \quad 2 \quad y0 \;, F = \quad x \quad 0 \quad y2 \quad x2 \quad y0^2 .$$

Неявное уравнение квадратичной кривой Безье для x 1= y1=0 составляет:

$$2\,x \quad y0 \quad y2 \quad 2 \quad 2xy \quad x0 \quad x \quad 2 \quad y0 \quad y2 \quad y^2 \quad y0 \quad y2 \quad 2$$

$$2 \quad x \quad y0 \quad y2 \quad y \quad y0 \quad x \quad 2 \quad x_0 \quad y2 \quad x \quad 2 \quad y0 \quad x0 \quad y2 \quad x2 \quad y0 \quad 2=0.$$

Общая кривизна кривой Безье определяется выражением

$$\text{добавить} = x_0 \quad y2 \quad x2 \quad y0 = \quad x0 \quad x1 \quad y2 \quad y1 \quad x2 \quad x1 \quad y0 \quad y1 \;. \qquad (8)$$

Предыдущие заменымогут быть добавленыснова. В результате некоторых вычислений неявное уравнение квадратичной кривой Безье упрощается до:

(x (y0 2 y1+ y2) y(x0 2 x1+ x2)) 2+ 2(x (y0 y 2) y (x0 x2))cur+ cur 2=0. (9)

Квадратичная кривая Безье является частью юпараболы

3.1 Расчет погрешности

Алгоритм построения основан на постоянном положительном (или отрицательном) градиенте кривой (наклон либо увеличивается, либо падает). Поскольку точка P_1 является точкой начала координат P_0 должна находиться в третьем квадранте $_2$ в первом квадранте для положительного градиента. Следующее иусловия P всегда истинны, тогда: x 1= y1=0, x 0 0 x2 и y0 0 y2 .

Выполнить это требование не составляет труда, поскольку кривуюБезье можно разбить в точке сменызнака градиента, рисуя две кривые одну за другой.

Уравнение для расчета ошибки кривой Безье имеет следующий

вид: $x \quad 2 \quad x \quad 0 \quad y2 \quad x \quad 2 \quad y0 \quad y \quad x \quad 0 \quad x2$

что эквивалентно

$x \quad e \quad x, y \quad = \quad x \quad y0 \quad y2 \quad y \quad x \quad 0 \quad x0 \quad y2 \quad 2 \; x \; x \; 2 \; 2 \; y0 \; y0 \; . 2 \quad 4 \quad x \quad y0 \quad y$

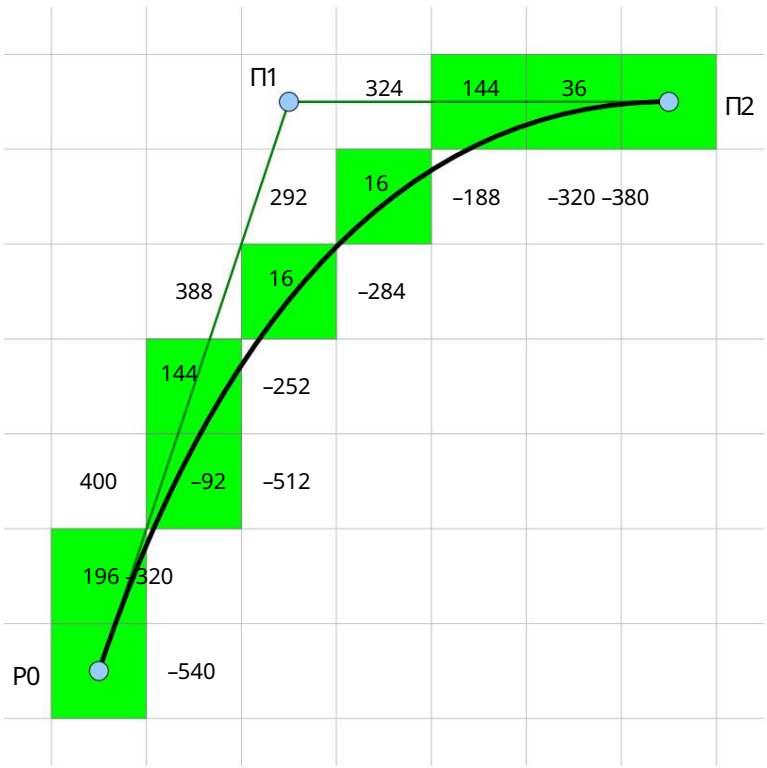


Рисунок 9: Значения ошибок квадратичной кривой Безье

Для алгоритма интерес представляют только изменяющиеся членыприращения шага, поскольку у остальные членыостаются постоянными во время пиксельного цикла.

Начальные значения этих изменяющихся членов представляют собой разности нее вного уравнения и вычисляются следующим образом:

d x=e x ±1, y e x, y = 1±2 x y0 y2 2 2 y x 0 x 2 y0 y2 ±2 cur y0 y2

2 . d y=e x, y±1 e x, y = 1±2 y x0 x 2 2 2 x x0 x2 y0 y 2 2 cur x0 x

Поск оль к у эт а к ривая Без ь е имеет вт оруюст епень , ошибк а приращения т ак же меня ет ся на к аждом шаге. Не т оль к о ошибк а расч ет а д олжна увелич иват ь ся в соот вет ст вии с шагами, но и само приращение dx и dy меня ет ся на к аждом шаге. В случ ае к вад рат ич ног о полинома эт о т ак же можно вьн ислит ь по вт орой произ вод ной.

Для шаг а по оси x приращение dx увелич ивает ся примерно

$$d\,xx = e\ x\ 2, y\ \ 2e\ x\ 1, y = \frac{2\,и}{x}\ 2=2\ \ y0\ \ y2\ \ 2=2\ \ y0\ \ 2\,y1\ \ y2^{2}$$

и dy увелич ивает ся

о

$$d\,xy = e\ x\ 1, y\ 1\ \ e\ x\ 1, y\ \ e\ x, y\ 1 = = \frac{2\,и}{2\ x\ \ 0}\ x2\ \ y0\ \ y2 =$$

$$= 2\ \ x0\ \ 2\,x1\ \ x2\ \ y0\ \ 2\,y1\ \ y2\ .$$

Для шаг а в направлении y приращение dy увелич ивает ся примерно

$$yy = e\ \ x, y\ 2\ \ 2\,e\ \ x, y\ 1 = y\ \ \frac{2\,и}{2=2\ \ x\ \ 0\ \ x2\ \ 2=2\ \ x0\ \ 2\,x1\ \ x2\ \ d}^{2}$$

и dx увелич ивает ся примерно $\frac{2_{\text{эт о}}}{x\ \ y}$.

Эт и приращения не з авися т от x и y.

3.2 Проблемысо слег к а из ог нут ьми линия ми

Пок а алг оритм раб от ает х орошо. Но эт о т ерпит неуд ач у, к ог д а к ривая Без ь е ст ановит ся поч т и пря мой. Чт о произ ойд ет , ст анет я сно, если проанализироват ь всю к ривую, а не т оль к о ее к ород к ую уч аст ь , к от орую алг оритм х оч ет пост роит ь . К ривая пред ст авля ет собой симмет рич ную параболу. У нег о ест ь вт орая ч аст ь . Для к ривьх с боль шой к ривиз ной вт орая полови на наход ит ся д алек о, ост авля я я сный пут ь , по к от орому может след оват ь алг оритм. Но на поч т и пря мьх линия х эт а вт орая полови на может попаст ь в пред елы т ек ущег о воз можног о пик селя ! Т ог д а алг оритм з апут ывает ся , поск оль к у он опирает ся на ч ет к ий г рад иент з нач ения ошибок .

Эт а проблема воз ник ала рань ше. На плоск их эллипсах с a = 1 алг оритм ост анавливался слиш ком рано. Но ситуац ия сложилась уд ач но. Эллипсывсег д а распол ог ались в симмет рич ной орт ог онал ьной ориент ац ии. Алг оритм д ал сбой т оль к о в од ном случ ае, к от орьй можно было испр авит ь д оп олнит ел ьнымц ик лом.

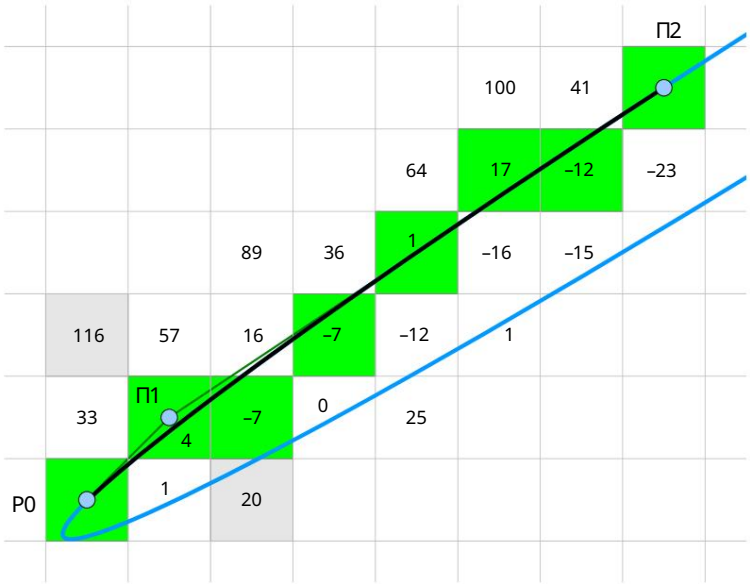


Рисунок 10. Алгоритм в беде: нет пути, по которому нужно идти

На рисунке 10 показана неоднозначная ситуация. Никакого положительного/отрицательного градиента не видно из точки П0. Даже если бы алгоритм мог каким-то образом обнаружить способ с низкими значениями ошибок, зеленый пиксель рядом с P1 со значением –7 будет выбран неправильно, поскольку у него низкий абсолютный показатель. значение из неправильной синей половины кривой, а не из желаемой черной.

Алгоритм дает сбой, если другая часть (невидимой) функции приближается к окрестности установить пиксель.

Что делать? Проблема возникает только на почти прямых линиях. Точки Безье также очень асимметричный. Есть несколько возможностей. Одним из решений является проверка, является ли значение ошибки несколько пикселей от начальной точки все еще находятся ниже (направление x) или выше (направление y) нуля. Когда алгоритм смотрит на n пикселей вперед по направлениям x или y от P0, он должен проверить

if $n\,dx\,dxy \quad n\,1\,dn_{yyy}d2y\,d^{0}xy \quad n\,1\,dтакже должно быть сделано\ _{xx}/2$ 0. Но эта проверка для P2, поскольку ситуация может быть обратной и конечная точка P2 вызовет путаница. Другой возможностью было бы просто увеличить разрешение пикселя растра. чтобы найти правильный путь пикселя кривой и установить ближайший к нему соответствующий пиксель.

Это решение вообще не удовлетворяет. Несколько почти диагональных прямых линий все еще получаются. дополнительные пиксели, хотя алгоритм работает. А также остается вопрос, что делать, если этот алгоритм терпит неудачу из-за этого теоретического недостатка? Самое простое решение — построить кривую вместо этого две или более прямых линий.

3.3 Программа для построения простых кривых Безье

Программа в листинге 8 принимает только базовые кривые Безье без смены знака. градиент (без горизонтальных и вертикальных поворотов). Смена знака потребует доп. вычисления в пиксельном цикле. Эта проблема решается позже путем подразделения.

```
voidplotBasicQuadBezier (int x0, int y0, int x1, int y1, int x2, int y2)
{
  int sx = x0<x2? 1:-1, sy = y0<y2? 1:-1; /* направление шага */
  двойной x = x0-2*x1+x2, y = y0-2*y1+y2, xy = 2*x*y*sx*sy;
  double cur = sx*sy*(x*(y2-y0)-y*(x2-x0))/2; /* кривизна */
                                              /* вычисляем приращение ошибки P0 */
  двойной dx = (1-2*abs(x0-x1))*y*y+abs(y0-y1)*xy-2*cur*abs(y0-y2);
  двойной dy = (1-2*abs(y0-y1))*x*x+abs(x0-x1)*xy+2*cur*abs(x0-x2);
                                              /* вычисляем приращение ошибки P2 */
  двойной ex = (1-2*abs(x2-x1))*y*y+abs(y2-y1)*xy+2*cur*abs(y0-y2);
  двойной ey = (1-2*abs(y2-y1))*x*x+abs(x2-x1)*xy-2*cur*abs(x0-x2);

                                            /* знак градиента не должен меняться */
  Assert((x0-x1)*(x2-x1) <= 0 && (y0-y1)*(y2-y1) <= 0);

  если (cur==0) { plotLine(x0,y0,x2,y2); возвращаться ; } /* прямая линия */

  x *= 2*x; y *= 2*y;
  если (cur < 0) {                                        /* отрицательная кривизна */
    x = -x; дх = -дх; бывший = -ex; xy = -xy;
    y = -y; дом = -дом; эй = –эй;
  }
  /* алгоритм не работает почти по прямой, проверьте значения ошибок */
  if (dx >= –y || dy <= –x || ex <= –y || ey >= –x) { x1 = (x0+4*x1+x2)/6; y1 =
    (y0+4*y1+y2)/6; /* аппроксимация */
    plotLine(x0,y0, x1,y1);
    plotLine(x1,y1, x2,y2);
    возвращаться ;
  }
  дх –= xy; ex = dx+dy; dy –= xy;                         /* ошибка 1.шага */

  for(;;)                                                 /* построить кривую*/
    { setPixel(x0,y0);
    ey = 2*ex–dy; if                                      /* сохраняем значение для проверки шага y */
    (2*ex >= dx) { if (x0 ==                              /* x шаг */
      x2) сломать ;
      x0 += x; dy –= xy; ex += dx += y;
    }
    if (ey <= 0) { if (y0 ==                              /* шаг */
      y2) сломать ;
      y0 += cy; дх –= xy; ex += dy += x;
    }
  }
}
```

Листинг 8: Программа для построения базовой кривой Безье

Несколько комментариев к листингу

8: Отрицательная кривизна сводит на нет градиент значений ошибок. Это оставляет возможность либо отрицать и другие значения, либо использовать другой пиксельный цикл с измененным условием (exy+dx<0 x++).

Приращения ошибки используются для просмотра на три пикселя в направлении x и y вперед и обнаружения поворотов прямых линий путем дополнительного изменения градиента значений ошибки. Это делается для обоих концов, на которых приращения P2 необходимы только для этой проверки. В этом случае кривая рисуется двумя линиями.

Алгоритм кривой не выполняет никакой аппроксимации. Значение ошибки последнего пикселя всегда будет равно нулю Вот почему условие прерывания цикла безопасно проверять только для последнего пикселя. Но во время тестирования полезно добавить дополнительный счетчик цикла в оператор for, если что-то пойдет не так.

Поскольку у каждый шаг также изменяет значение приращения, его пришлось сохранить для второго теста.
В противном случае несколько пикселей сдвинутся на один шаг по оси Y. Обычно это не было бы замечено, за исключением случаев, когда это был последний пиксель для проверки условия разрыва.

3.4 Растр высокого разрешения

Предыдущий алгоритм аппроксимирует кривуюБезье двумя линиями, если другая часть кривой находится слишком близко к заданному пикселю Построенная кривая Безье не выглядит гладкой в таком случае.
случай.

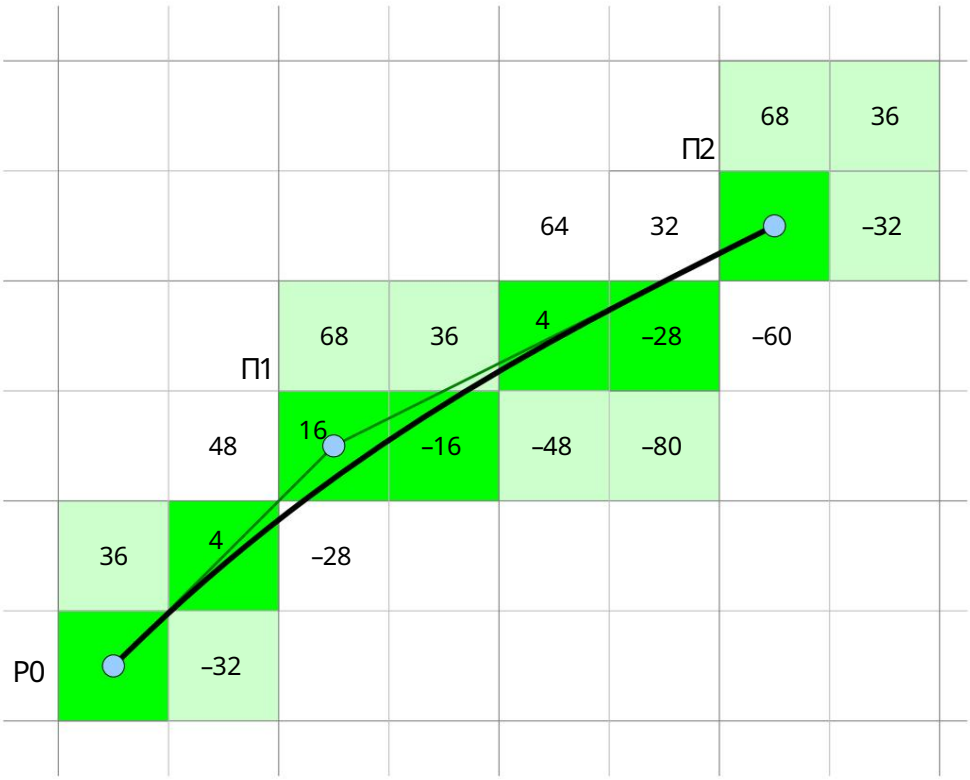


Рисунок 11. Более высокое разрешение субпиксельного растра.

Другая альтернатива — использовать более тонкий растр субпикселей и установить пиксель, ближайший к этой пиксельной кривой. Этот растр высокого разрешения должен быть достаточно мелким, чтобы избежать конфликта двух кривых на одном пикселе или очень близком пикселе.

На рисунке 11 показана кривая Безье с субпиксельным растром двойной точности. Каждый пиксель (светло-зеленый) разделен на субпикселя (зеленый).

Сам алгоритм работает с более мелким пиксельным растром и поэтому без проблем находит путь с подходящими значениями ошибок. Каждый раз, когда субпиксель завершается, устанавливается сам пиксель.

Концепция субпикселизации также используется в алгоритме [Емельяненко, 2007] для точного рисования неявных кривых. Эта концепция предлагает решение, если алгоритм этого документа дает сбой или становится слишком сложным для реализации.

Версия в листинге 9 требует большего количества вычислений в пиксельном цикле, чем базовый алгоритм, и поэтому немного медленнее, но никогда не приближается к квадратичной кривой Безье.

Расчет коэффициента разрешения гарантирует, что знак значения ошибки не изменится из-за замыкания кривой на три субпикселя от P0 или P2 в направлении x или y. Но это решение заканчивается делением на ноль в случае максимума, которому нужно уделять особое внимание.

Поскольку вычисление ошибки происходит на один пиксель вперед, вычисление последнего шага выходит за пределы конечного пикселя. В этом случае базовый алгоритм без проблем останавливается, хотя значения приращения уже могут быть недействительными, поскольку кривая может сделать резкий поворот. Но тонкий алгоритм дорабатывает все субпиксели пикселя. Но в случае поворота субпиксели не могут быть завершены, так как значения приращения уже изменили знак. В этом случае во внутренний цикл необходимо вставить дополнительное условие прерывания, чтобы избежать бесконечного цикла.

В некоторых случаях Безье этот коэффициент разрешения может стать довольно большим. Но лишь несколько эксцентричных кривых замедляют работу алгоритма.

Преимущество этого алгоритма в том, что построенная кривая не имеет ошибок аппроксимации. Все заданные пиксели максимально приближены к аналоговой кривой Безье.

Альтернативой является использование тонкого алгоритма только в случае сбоя базовой версии (f >1). Длинную кривую также можно разделить на длинную, почти прямую часть, построенную с помощью более быстрой базовой версии, и более короткую изогнутую часть с помощью тонкого алгоритма.

```
voidplotFineQuadBezier (int x0, int y0, int x1, int y1, int x2, int y2)
{
    int sx = x0<x2? 1:-1, sy = y0<y2? 1:-1; длинные f = 1, fx = x0–2*x1+x2,          /* направление шага */
    fy = y0–2*y1+y2;
5   длинный длинный x = 2*fx*fx, y = 2*fy*fy, xy = 2*fx*fy*sx*sy;
    long long cur = sx*sy*(fx*(y2–y0)–fy*(x2–x0)); /* кривизна */
                                                   /* вычисляем приращение ошибки P0 */
    long long dx = abs(y0–y1)*xy–abs(x0–x1)*y–cur*abs(y0–y2);
    long long dy = abs(x0–x1)*xy–abs(y0–y1)*x+cur*abs(x0–x2);
10                                                 /* вычисляем приращение ошибки P2 */
    long long ex = abs(y2–y1)*xy–abs(x2–x1)*y+cur*abs(y0–y2);
    long long ey = abs(x2–x1)*xy–abs(y2–y1)*x–cur*abs(x0–x2);

                                                /* знаки градиента не должны меняться */
15  Assert((x0–x1)*(x2–x1) <= 0 && (y0–y1)*(y2–y1) <= 0);

    если (cur == 0) { plotLine(x0,y0, x2,y2); возвращаться ; } /* прямая линия */

                                          /* вычисление требуемого минимального коэффициента разрешения */
20  если (dx == 0 || dy == 0 || ex == 0 || ey == 0)
        e = абс(xy/cur)/2+1;                          /* деление на ноль : используйте кривизну */
    еще {
        x = 2*y/dx; если (fx > f) f = fx; x = 2*x/dи; если                  /* увеличиваем разрешение */
        (fx > f) f = fx; x = 2*y/ex; если (fx > f) f = fx; fx =
25      2*x/ey; если (fx > f) f = fx;

    } /* отрицательная кривизна? */
    если (cur < 0) { x = –x; y = –y; дx = –дx; дy= –ди; xy = –xy; }
    dx = f*dx+y/2–xy; dy = f*dy+x/2–xy; ex = dx+dy+xy; /* ошибка 1-го шага */
30
    for (fx = fy = f; ; ) { setPixel(x0,y0);                                    /* построить кривую*/

        if (x0 == x2 && y0 == y2) сломать ;
        do { /* переместить субпиксель */
35          ey = 2*ex–dy; /* сохраняем значение для проверки шага y */
            если (2*ex >= dx) { fx--; dy –= xy; ex += dx += y; } /* x шаг */
            если (ey <= 0) { fy--; дx –= xy; ex += dy += x; } /* шаг */
        } while (fx > 0 && fy > 0); /* пиксель завершен? */
        если (2*fx <= f) { x0 += sx; x += e; } /* достаточные подэтапы. */
40      если (2*fy <= f) { y0 += sy; фу += e; } /* .. для пикселя ? */
    }
}
```

Листинг 9. Построение кривой Безье на мелкой сетке

3.5 Построение умной кривой

Когда базовый алгоритм дает сбой и требует особого внимания? Учитывая всю кривая Безье является параболой. Алгоритм дает сбой, когда две симметричные части кривые сближаются внутри одного пикселя. Но от этой точки до вершины Кривая Безье становится прямой, поскольку другая часть начинается с того же пикселя. Проблема аналогична алгоритму эллипса. В случае неудачи для построения остается только линия.

Алгоритм можно упростить при двух условиях: рисование начинается с более длинного часть кривой, где они все еще четко разделены, а во-вторых, оставшаяся часть представляет собой простая строка, когда алгоритм дает сбой. Проблема не возникает, когда вершина рисуется парабола, поскольку в этом случае кривуюнеобходимо разделить.

```
voidplotQuadBezierSeg (int x0, int y0, int x1, int y1, int x2, int y2)
{ /* построить ограниченный квадратичный сегмент Безье */
    int sx = x2-x1, sy = y2-y1;
    длинный xx = x0-x1, yy = y0-y1, xy; двойной                    /* относительные значения для проверок */
5   dx, dy, err, cur = xx*sy-yy*sx;                                          /* кривизна */

    Assert(xx*sx <= 0 && yy*sy <= 0); /* знак градиента не должен меняться */

    if (sx*(long)sx+sy*(long)sy > xx*xx+yy*yy) { /* начинаем с более длинной части */
10      x2 = x0; x0 = x+x1; y2 = y0; y0 = си+y1; кур = -cur; /* поменять местами P0 P2 */

    } if (cur != 0) { xx +=                                          /* нет прямой линии */
        sx; xx *= sx = x0 < x2 ? 1:-1; yy += сы; yy *= sy = y0 < y2 ?          /* направление шага x */
        1:-1; xy = 2*xx*yy; xx *= xx; yy *= yy; если (cur*sx*sy <          /* направление шага по оси */
15      0) {                                                    /* различия 2-й степени */
                                                                    /* отрицательная кривизна? */
            xx = -xx; yy = -yy; xy = -xy; кур = -cur;
        }
        dx = 4,0*sy*cur*(x1-x0)+xx-xy; dy =                      /* различия 1-й степени */
20      4,0*sx*cur*(y0-y1)+yy-xy;
        xx += xx; yy += yy; ошибка = dx+dy+xy; сделать                    /* ошибка 1-го шага */

        { setPixel(x0,y0); /* построить кривую*/
            if (x0 == x2 && y0 == y2) return; /* последний пиксель -> кривая закончена */
25          y1 = 2*ошибка < dx; /* сохраняем значение для проверки шага y */
            если (2*err > dy) { x0 += sx; дх -= xy; ошибка += dy += yy; } /* x шаг */
            если ( ) { y0 y+= sy; ды-= xy; ошибка += dx += xx; } /* шаг */
        } while (dy <0 && dx > 0);                    /* градиент отрицает -> алгоритм не работает */
    }
30  plotLine(x0,y0, x2,y2);                                      /* отображаем оставшуюся часть до конца */
}
```

Листинг 10. Алгоритм быстрой кривой Безье

Алгоритм начинается с конца, который находится дальше от вершины, поскольку тогда другая часть кривой, вероятно, находится достаточно далеко. Алгоритм останавливается, если две симметричные части параболыподходят слишком близко друг к другу и алгоритм терпит неудачу. Это можно проверить, если производная градиента значения ошибки меняет свой знак. Затем кривая завершается проведением прямой линии до конца кривой.

Проблема не возникает, когда кривая Безье состоит из обеих частей параболы
Если кривизна вершинысжимается до одной точки, то кривуюнеобходимо предварительно разделить, поскольку градиент там также меняет направление. Если это не одна точка, то две сторонынаходятся достаточно далеко друг от друга и алгоритм не дает сбоя.

Это решение очень эффективно для построения квадратичной кривой Безье. Преимущество алгоритма состоит в том, что если кривая подходит слишком близко друг к другу, чтобыработать, она становится достаточно прямой, чтобыее можно было завершить линией, поэтому он никогда не дает сбоя.

Поскольку значения ошибок могут быть довольно большими (вплоть до четвертой степени), вместо длинного целого числа используется тип double. Если доступен 64-битный целочисленный тип, можно использовать этот тип. вместо.

3.6 Общие кривые Безье

Предыдущие алгоритмыБезье для простотыполагались на непрерывный положительный или отрицательный градиент. Смена знака будет означать изменение направления заданного пикселя внутри цикла. Расчет ошибки опережает на один пиксель и тоже потребует изменений. Безье
кривая разделена на горизонтальные и вертикальные повороты, чтобыизбежать этих проблем.

Разделение кривой Безье также дает дополнительное преимущество, позволяющее избежать неблагоприятных поворотов.

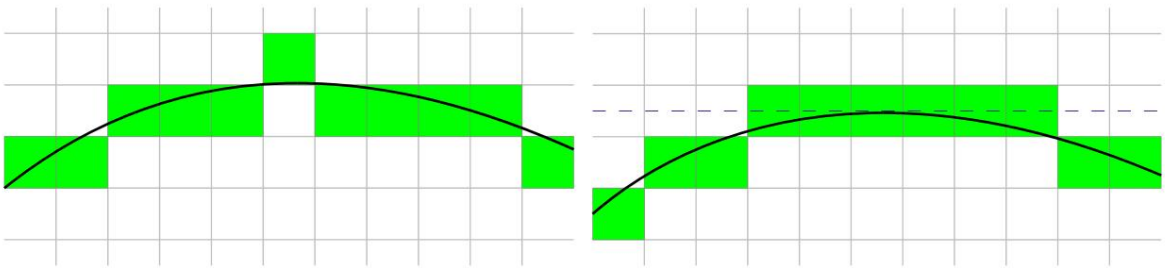


Рисунок 12: Неблагоприятный поворот кривой

Слева на рисунке 12 показан поворот кривой с изолированным пикселем, который кажется не в том месте. С точки зрения алгоритма это правильно, поскольку оно ближе всего к кривой. Чтобыизбежать неблагоприятной ситуации, поворот кривой привязывается к центру, пикселю, показанному справа на рисунке 12 пунктирной линией. Конечно, это немного меняет всю кривую, но выглядит намного лучше.

Алгоритм должен разделить кривуюБезье в точках, в которых градиент меняет свой знак. Это вертикальные и горизонтальные точки поворота кривой.

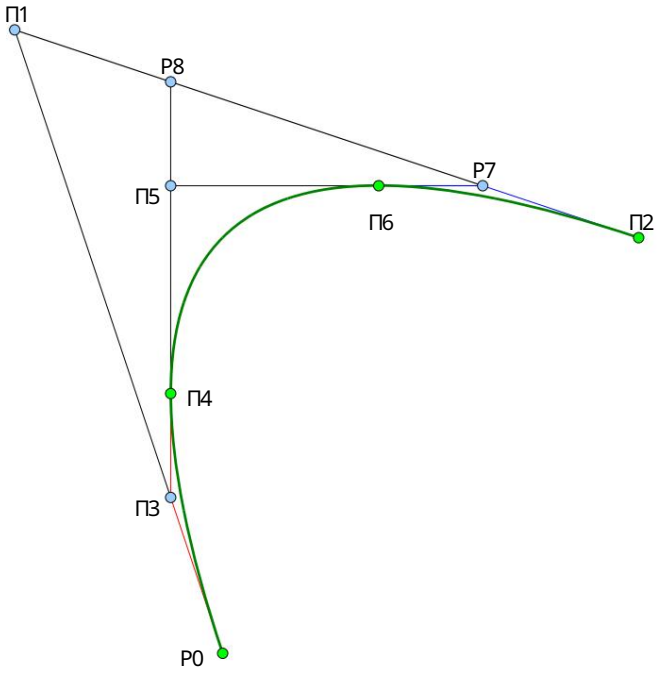


Рисунок 13. Деление кривой Безье.

Даже если это трудно поверить, одна квадратичная кривая Безье может иметь два изменения градиента и должна быть разделена на три части, чтобы ее можно было построить с помощью предыдущей программы. Такая более сложная кривая видна на рисунке 13. Данные точки P0, P1 и P2 кривой пришлось разделить в вертикальной точке P4 и в горизонтальной точке P6 на более короткую кривую Безье P0/P3/P4, кривую P4/P5/P6 и кривая P6/P7/P2.

Установка уравнения градиента нея вного уравнения Безье на ноль дает точку изменения градиента: t (P0 – 2P1 + P2) – P0 + P1 = 0. Только значения 0 t 1 указывают на изменение знак градиента. Точка P5 вычисляется по формуле $t_x = \frac{x0 - x1}{x0 - 2x1 + x2}$, $t_y = \frac{y0 - y1}{y0 - 2y1 + y2}$ и $x5 = \frac{2x0x2 - x1}{y1 - y}$, $y5 = x0 - 2x1 + x2\,y0 - 2\frac{y0y2 - y1^2}{и}$. P4 и P6 рассчитываются одинаково:

$x6 = 1 - t_и^2 x0 - 2 1 - t_и t_и x1 - t_и x2$, $y4 = 1 - t_{макс}^2 y0 - 2 1 - t_x t_{макс} y1 - t_{макс} y2$.

P3 и P7 можно вычислить путем пересечения линий: $y3 = y0\frac{x0 - x5}{x0 - x1} y0 - y1$,

$x7 = x2 - \frac{y2 - y5}{y2 - y1} x2 - x1$.

3.7 Программа для построения любой кривой Безье

Программа подразделяет кривую при изменении горизонтального и вертикального градиента и передает построение графика подпроцедуре.

Сначала он проверяет, необходимли горизонтальный разрез (dx = 0) в точке P4 и нужен ли вертикальный разрез (dy = 0).
тоже происходит. Если происходя т оба разреза, он гарантирует, что горизонтальное разделение произойдет первым с помощью юусловного
обмен точками. Затем он чертит деталь и отрезает ее. То же самое делается и по вертикали.
резать.

Интересно отметить, что целочисленное деление отрицательных чисел (и напоминаний)
зависит от приложения, платформыилия зыка программирования. Математическое определение
(в Ruby или Python) округляется в сторону меньшего целого значения (при этом остаток всегда
остается положительным): (–5)/3 = –2, тогда как техническое определение (в C/C++ или Java) округляет
к нулю (–5)/3 = –1. Имейте в виду, что ваша целевая система может обрабатывать это иначе, чем программа-пример.

```
voidplotQuadBezier (int x0, int y0, int x1, int y1, int x2, int y2)
{                                                   /* построить любую квадратичную кривуюБезье */
    int x = x0-x1, y = y0-y1;
    двойной т = x0-2*x1+x2, г;

    if ((long)x*(x2-x1) > 0) { if ((long)y*(y2-          /* горизонтальный разрез на P4? */
        y1) > 0) if (fabs((y0-2*y1+y2)/t*x) >           /* вертикальный разрез на P6 тоже? */
            абс(y)) {                                    /* что сначала? */
                x0 = x2; x2 = x+x1; y0 = y2; y2 = y+y1; /* теперь      /* точки обмена */
            }                                   горизонтальный разрез на P4 идет первым */
        т = (x0-x1)/т;
        r = (1-t)*((1-t)*y0+2,0*t*y1)+t*t*y2; т = (x0*x2-x1*x1)*т /          /* By(t=P4) */
        (x0-x1); x = пол(т +0,5); y = пол(г+0,5);          /* градиент dP4/dx=0 */

        г = (y1-y0)*(t-x0)/(x1-x0)+y0; /* пересекаем P3 | P0 P1 */
        plotQuadBezierSeg(x0,y0, x,floor(r+0.5),x,y);
        г = (y1-y2)*(t-x2)/(x1-x2)+y2; /* пересекаем P4 | П1 П2 */
        x0 = x1 = x; y0 = y; y1 = пол(r+0,5); /* P0 = P4, P1 = P8 */

    } if ((long)(y0-y1)*(y2-y1) > 0) {                  /* вертикальный разрез на P6? */
        т = y0-2*y1+y2; т = (y0-y1)/т; r = (1-t)*((1-
        t)*x0+2,0*t*x1)+t*t*x2; t = (y0*y2-y1*y1)*t/(y0-y1); x =          /* Bx(t=P6) */
        пол(г+0,5); y = пол(т +0,5);          /* градиент dP6/dy=0 */

        г = (x1-x0)*(т -y0)/(y1-y0)+x0; /* пересекаем P6 | P0 P1 */
        plotQuadBezierSeg(x0,y0, floor(r+0.5),y,x,y);
        г = (x1-x2)*(т -y2)/(y1-y2)+x2; /* пересекаем P7 | П1 П2 */
        x0 = x; x1 = пол(r+0,5); y0 = y1 = y; /* P0 = P6, P1 = P7 */

    } plotQuadBezierSeg(x0,y0, x1,y1,x2,y2);          /* оставшаяся часть */
}
```

Листинг 11. Деление комплексной квадратичной кривой Безье

Округление в C немного сложнее, особенно если оно должно работать и для отрицательных чисел. Поэтому арифметика с плавающей запятой используется только для правильного округления делений. Это также возможно сделать только в целых числах, но это сложнее: round(a/b) = (a+sign(a)*abs(b)/2)/b.

Разделение кривой Безье на части всегда приводит к определенным целочисленным ошибкам округления точек Безье. Это становится особенно заметно, если две точки Безье сближаются (около < 10 пикселей). Иногда одна из кривых даже становится прямой и как бы не стыкуется с остальными. С другой стороны, это также имеет то преимущество, что горизонтальное

и/или вертикальные интервалы кривой всегда растеризуются на полный пиксель. Изогнутые переходы выглядят намного лучше.

Среднюю контрольную точку P1 можно заменить на проходную точку P'_1 на $P1=2P'_1 - \frac{P0+P2}{2}$. (10)

4 Рациональный Безье

Для рационального Безье каждая точка Pi уравнения (4) получает дополнительный вес wi: [Марщ 2005, с. 175]

$$Бн\ t = \frac{\sum_{i=0}^{n} Da\,Pi \cdot \binom{n}{i} t^{i} (1-t)^{n-i}}{\sum_{i=0}^{n} Wi \cdot \binom{n}{i} t^{i} (1-t)^{n-i}} \cdot [0 \le t \le 1] \qquad (11)$$

4.1 Квадратично-рациональный Безье

Для n = 2 уравнения (11) рациональный квадратичный Безье принимает вид

$$B2\ t = \frac{(1-t)^{2} w0\,P0 + 2(1-t)t\,w1\,P1 + t^{2} w2\,P2}{(1-t)^{2} w0 + 2(1-t)t\,w1 + t^{2} w2} \qquad (12)$$

То же матричное уравнение (7) для нерационального Безье используется для вычисления неявного квадратичного рационального уравнения Безье, за исключением того, что уравнение (6) для t = ½ должно учитывать веса: a w0 x0 2w1 x1 w2 x2 2 2b w0 x0 2 w1 x1 w2 x2 w0 y0 2 w1 y1 w2 y2 c w0 y0 2w1 y1 w2 y 2 2 2d w0 2w1 w2 w0 x0 2w1 x1 w2 x2 2 e w0 2 w1 w2 w0 y0 2w1 y1 w2 y2 = w0 2w1 w2 2

Решение матричного уравнения неявного уравнения рациональной квадратичной кривой Безье делает тогда:

$$x^{2}(w0w2 - y0 - y2 + 2)^{4} + 2(w0y2 + 2xy - w0w2 - x0 - x2 + y0 + y2 - 2w1)^{2}(x0y2 - x2y0) + y(w0w2 - x0 - x2 + 2)^{4} + 2(w1^{2} - x0x2 - 2w0w2 + x0 + y0 + y2 - y)(x0 - x2)^{2}(y0 - y2)^{2} + x(w0w2 - x0y0 - y2 + x2 + y0 - 2)^{4} = 0.$$

Это уравнение предполагает замену $w = w0w2$ $2 = \frac{w1^{2}}{w}$ для упрощения, выражая все весит только по одному значению

Тогда неявное уравнение рационального квадратичного Безье принимает вид: (13)

$$x^{2}(y0 - y2 + 2)^{4} + 4(w2 - y0 - y2 + 2xy - x0 - x + 2)^{2}(y0 - y2 + 2w2 - x0 - y2 - x2 + y0) + y(x0 - x + 2)^{4} + 2(x0y + 2x + 2y0 - x - 2)^{2}(4w2 - y0 - y2 + x + 2y0 - 2) = 0.$$

Отдельные веса в конечных точках P0 и P2 рационального квадратичного Безье изменяют кривую так же, как вес средней точки P1. Таким образом, веса можно было нормализовать без изменения кривой путем замены веса средней точки P1 на $w=w1 / w0w2$. Тогда веса двух конечных точек становятся одним и их можно игнорировать.

При w =1 кривая представляет собой параболу, при w < 1 кривая представляет собой эллипс, при w = 0 кривая представляет собой прямую линию, а при w>1 кривая представляет собой гиперболу. Обычно предполагается, что все веса положительны.

Квадратичная кривая Безье снова должна быть разделена на горизонтальные и вертикальные точки поворота. Эти четыре точки рассчитываются путем установки первой производной уравнения параметра (12) на ноль:

$$t = \frac{2w(P0 - P1) \quad P0 + P2 \pm \sqrt{4w^2(P0 - P1)(P2 - P1) + (P0 - P2)^2}}{2(w - 1)(P0 - P2)}. \qquad (14)$$

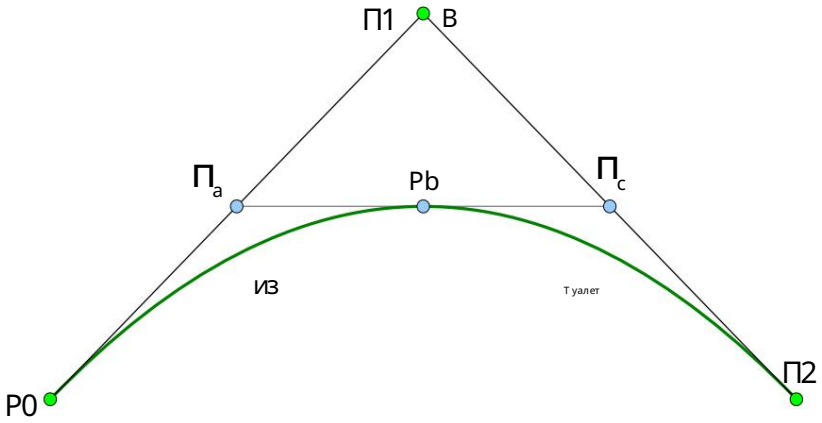


Рисунок 14: Подразделение квадратично-рационального Безье

Алгоритм де Кастельжо можно расширить для рационального Безье путем преобразования двумерного вектора [xi, yi] плюс вес wi рациональной кривой в трехмерный вектор нерациональной кривой [wi xi, wi yi, wi]. После подразделения трехмерный вектор [xi, yi, wi] отображается обратно в 2D [xi/wi, yi/ wi] плюс вес wi. Когда кривая на рисунке 14 подразделяется в положении

$$\text{параметра } t\text{, дополнительные точки}, \text{из} = \frac{\text{параметра } t\text{, дополнительные точки}}{\text{становятся : } t(w\,P1 \quad P0) + P0\; t(w \quad 1) + 1\; Pa = t(w \quad 1) +},$$

$$Pb = \frac{^{2}\tau\,1 \quad 2\,t(1 \quad t)(w \quad 1) + 1\,(P0 \quad 2w\,P1 + P2) +}{2t(w\,P1 \quad P0) + P0\;2},$$

$$\frac{t(1 \quad t)(w \quad 1) + 1\,(1 \quad t)}{(\,wP1 \quad P2) + P2\;Pc = (1 \quad t)(w \quad 1) + 1}, \text{туалет} = \frac{(1 \quad t)(w \quad 1) + 1}{2t(1 \quad t)(w \quad 1) + 1} \qquad [0 \quad \tau \quad 1] \qquad (15)$$

На этот раз сначала представлен алгоритм подразделения, а позже — алгоритм рисования. Алгоритм подразделения такой же, как в листинге 11, за исключением того, что он учитывает вес для вычислений. В рациональных уравнениях вес чаще всего отображается в квадрате. Для упрощения расчетов параметр веса в алгоритме рисования сегмента задан как квадрат.

```
voidplotQuadRationalBezier (int x0, int y0, int x1, int y1,
                            int x2, int y2, float w)
{                                   /* построить любую квадратичную рациональную кривую Безье */
    int x = x0-2*x1+x2, y = y0-2*y1+y2;
    двойной xx = x0-x1, yy = y0-y1, ww, t, q;
    утверждать (ш>= 0,0);

    if (xx*(x2-x1) > 0) { if (yy*(y2-y1)                    /* горизонтальный разрез на P4? */
        > 0) if (fabs(xx*y) > fabs(yy*x))                   /* вертикальный разрез на P6 тоже? */
            {                                                   /* что сначала? */
                x0 = x2; x2 = xx+x1; y0 = y2; y2 = yy+y1; /* точки обмена */
            }                                   /* теперь горизонтальный разрез на P4 идет первым */
        если (x0 == x2 || w == 1,0) t = (x0-x1)/(double)x; /* нерациональный
        else { q                                            или рациональный случай */
            = sqrt(4.0*w*w*(x0-x1)*(x2-x1)+(x2-x0)*(long)(x2-x0));
            если (x1 < x0) q = -q; t =
            (2,0*w*(x0-x1)-x0+x2+q)/(2,0*(1,0-w)*(x2-x0));/* t в P4 */

        } q = 1,0/(2,0*t*(1,0-t)*(w-1,0)+1,0); xx =             /* разделить в точке t */
        (t*t*(x0-2,0*w*x1+x2)+2,0*t*(w*x1-x0)+x0)*q; yy =                   /* = P4 */
        (t*t*(y0-2,0*w*y1+y2)+2,0*t*(w*y1-y0)+y0)*q;
        ww = t*(w-1,0)+1,0; ву *= вв*q; /* квадрат веса P3 */
        ш= ((1.0-t)*(w-1.0)+1.0)*sqrt(q); /* вес P8 */
        x = пол(xx+0,5); y = пол(yy+0,5); /* P4 */
        yy = (xx-x0)*(y1-y0)/(x1-x0)+y0; /* пересекаем P3 | P0 P1 */
        plotQuadRationalBezierSeg(x0,y0, x,floor(yy+0.5), x,y, ww);
        yy = (xx-x2)*(y1-y2)/(x1-x2)+y2; /* пересекаем P4 | П1 П2 */
        y1 = пол(yy+0,5); x0 = x1 = x; y0 = y; /* P0 = P4, P1 = P8 */

    } if ((y0-y1)*(long)(y2-y1) > 0) { /* вертикальный разрез на P6? */
        если (y0 == y2 || w == 1,0) t = (y0-y1)/(y0-2,0*y1+y2);
        else { /* нерациональный или рациональный случай */
            q = sqrt(4.0*w*w*(y0-y1)*(y2-y1)+(y2-y0)*(long)(y2-y0));
            если (y1 < y0) q = -q; t =
            (2,0*w*(y0-y1)-y0+y2+q)/(2,0*(1,0-w)*(y2-y0));/* t на P6 */

        } q = 1,0/(2,0*t*(1,0-t)*(w-1,0)+1,0); xx =             /* разделить в точке t */
        (t*t*(x0-2,0*w*x1+x2)+2,0*t*(w*x1-x0)+x0)*q; yy =                   /* = P6 */
        (t*t*(y0-2,0*w*y1+y2)+2,0*t*(w*y1-y0)+y0)*q;
        ww = t*(w-1,0)+1,0; ву *= вв*q; /* квадрат веса P5 */
        ш= ((1.0-t)*(w-1.0)+1.0)*sqrt(q); /* вес P7 */
        x = пол(xx+0,5); y = пол(yy+0,5); /* P6 */
        xx = (x1-x0)*(yy-y0)/(y1-y0)+x0; /* пересекаем P6 | P0 P1 */
        plotQuadRationalBezierSeg(x0,y0, floor(xx+0.5),y, x,y, ww);
        xx = (x1-x2)*(yy-y2)/(y1-y2)+x2; /* пересекаем P7 | П1 П2 */
        x1 = пол(xx+0,5); x0 = x; y0 = y1 = y; /* P0 = P6, P1 = P7 */

    } plotQuadRationalBezierSeg(x0,y0, x1,y1,x2,y2,w*w); /* оставшийся */
}
```

Листинг 12. Деление квадратичной рациональной кривой Безье

Программа из листинга 12 подразделяет квадратичнуюрациональнуюкривуюБезье в точках горизонтального и вертикального поворота так же, как в листинге 11 это делается для рациональной кривой Безье. Таким образом, примечания в листинге 12 относятся к рисунку 13.

Эта реализация также строит рациональные графики Безье.

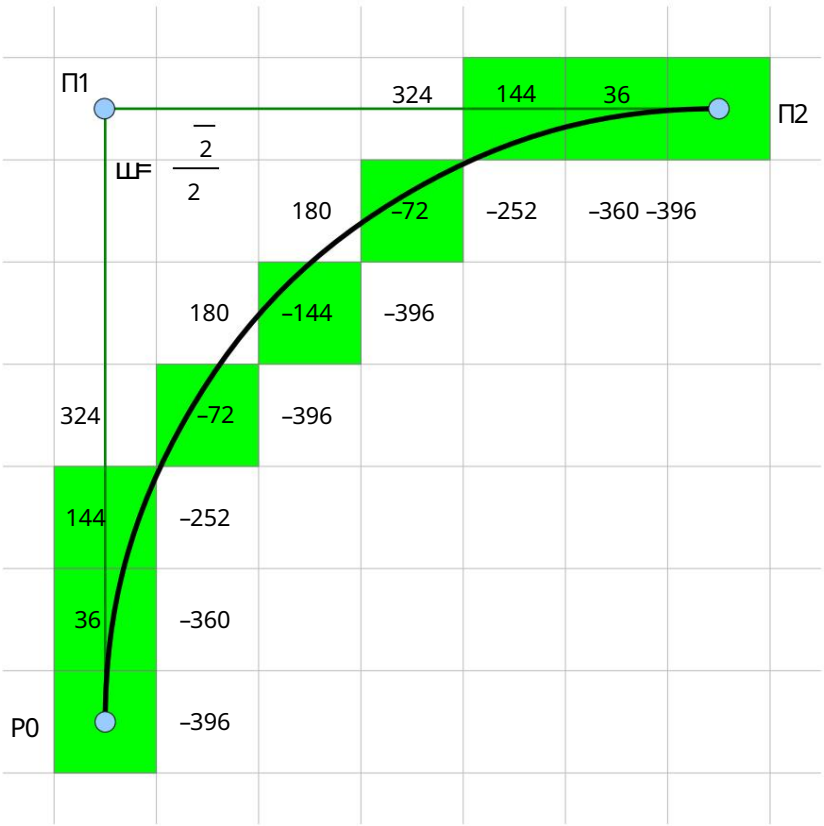


Рисунок 15: Значения ошибок квадратичного рационального Безье

Значения погрешности расчета составляют:

$$dx = e_{x0,y0} = 4w^2 (y_0 x_2 - y_0 x_0 - y_2 x_2 + y_2 x_0)^2, \quad (16)$$

$$dy = e_{x0,y0} = 4B_T^2 (x_0 y_2 - x_0 y_2 - x_2 y_0 + x_2 x_0)^2,$$

$$dxx = \frac{\partial^2}{\partial x^2} = 2 \quad 4w^2 (y_0 - y_2)^2, \quad dyy = \frac{\partial^2}{\partial y^2} = 2 \quad 4w^2 (x_0 - x_2)^2,$$

$$dxy = \frac{\partial^2}{\partial x \partial y} = -2w (x_0 - y_2 - x_2 + y_0).$$

Для w = 1 эти значения соответствуют уравнениям главы 3.1.

4.2 Рациональный квадратичный алгоритм

Очень маленькие значения веса могут привести к сбою алгоритма. Причина та же, что и раньше. Другая часть эллипса подходит слишком близко. Хотя это редкий случай, его легко исправить: просто разделить кривую пополам и построить каждый сегмент отдельно.

```
voidplotQuadRationalBezierSeg (int x0, int y0, int x1, int y1,
                               int x2, int y2, float w)
{                                  /* построим ограниченный рациональный сегмент Безье, квадрат веса */
  int sx = x2-x1, sy = y2-y1; /* относительные значения для проверок */
  двойной dx = x0-x2, dy = y0-y2, xx = x0-x1, yy = y0-y1; двойной xy = xx*sy+yy*sx,
  cur = xx*sy-yy*sx, ошибка; /* кривизна */

  Assert(xx*sx <= 0,0 && yy*sy <= 0,0); /* знак градиента не должен меняться */

  if (cur != 0,0 && w > 0,0) {                              /* нет прямой линии */
    if (sx*(long)sx+sy*(long)sy > xx*xx+yy*yy) {            /* начинаем с более длинной части */
      x2 = x0; x0 -= dx; y2 = y0; y0 -= dy; кур = -cur;     /* поменять местами P0 P2 */

    } xx = 2,0*(4,0*w*sx*xx+dx*dx); yy =                    /* различия 2-й степени */
    2,0*(4,0*w*sy*yy+dy*dy);
    sx = x0 <x2? 1:-1; си = y0 < y2?                        /* направление шага x */
    1:-1; xy =                                              /* направление шага по оси */
    -2,0*sx*sy*(2,0*w*xy+dx*dy);

    если (cur*sx*sy < 0,0) {                                /* отрицательная кривизна? */
      xx = -xx; yy = -yy; xy = -xy; кур = -cur;
    }
    dx = 4,0*w*(x1-x0)*sy*cur+xx/2,0+xy; dy = 4,0*w*(y0-    /* различия 1-й степени */
    y1)*sx*cur+yy/2,0+xy;

    if (w <0,5 && dy > dx) {                                /* плоский эллипс, алгоритм не работает */
      Cur = (w+1,0)/2,0; ш= sqrt (ш); xy = 1,0/(w+1,0);
      sx = пол((x0+2,0*w*x1+x2)*xy/2,0+0,5); sy =            /* разделить кривуюпополам */
      Floor((y0+2.0*w*y1+y2)*xy/2.0+0.5);
      dx = пол((w*x1+x0)*xy+0,5); dy = пол((y1*w+y0)*xy+0,5);
      plotQuadRationalBezierSeg(x0,y0, dx,dy, sx,sy, cur);/* построить график отдельно */
      dx = пол((w*x1+x2)*xy+0,5); dy = пол((y1*w+y2)*xy+0,5);
      plotQuadRationalBezierSeg(sx,sy, dx,dy,x2,y2,cur);
      возвращаться ;
    }
    ошибка = dx+dy-xy;                                      /* ошибка 1-го шага */

    сделать { setPixel(x0,y0); /* построить кривую*/
      if (x0 == x2 && y0 == y2) return; /* последний пиксель -> кривая закончена */
      x1 = 2*ошибка > dy; y1 = 2*(err+yy) < -dy;/* сохраняем значение для проверки шага x */
      если (2*err < dx || y1) { y0 += sy; ды+= xy; ошибка += dx += xx; }/* шаг */
      если (2*err > dx || x1) { x0 += sx; дх += xy; ошибка += dy += yy; }/* x шаг */
    } while (dy <= xy && dx >= xy);                         /* градиент отрицает -> алгоритм не работает */
  }
  plotLine(x0,y0, x2,y2);                                   /* отображаем оставшуюся иглу до конца */
}
```

Листинг 13. Постройте ограниченный рациональный сегмент Безье

Этот алгоритм также позволяет избежать установки ложного пикселя . Таким образом, пиксельный цикл листинга 13 проверяет , не происходит ли второй шаг х или у, изаранее выполняет соответствующий шаг. Такая проверка может быть включена в каждый цикл пикселей.

4.3 Вращение эллипса

В настоящее время разработаныинструментырешения задач иповернутого эллипса.

Когда эллипс преобразуется матрицей вращения R =[cos cossin] нейвнный уравнение становится :

$$2x \quad a^2 \text{грех} 2 \quad б^{2}_{\text{потому что}^2} \quad 2xy \quad a2 \quad b2 \quad \sin \quad \cos \quad y \quad ^{2} \quad ^{2}_{\text{потому что}^2} \quad b \ 2\sin 2 \quad a \ ^{2}62=0.$$

Определениями xd $^{2=a\,2}_{\text{потому что}^2}$ b , я$^{2}_{\text{рд}}$ a 2 sin2 2=a 2 sin2 b $^{22\,\text{потому что}}$ и

$$zd= \quad a2 \quad b2 \quad \sin \quad \cos \quad = \quad a2 \quad b2 \quad \frac{\tan}{1 \quad \tan 2} \text{"="} \frac{^{22}_{\text{но}}}{2} \text{грех} 2$$

(э .. эксцентриситет) нейвнный уравнение эллипса, повернутого на угол θ , принимает вид : xd 2 xd

$$2x \quad 2 \quad 2_{xy\,zd+\,y\,yd} \quad ^{2}_{2} \quad ^{2}_{zd\,yd^{2=0+}} \quad [|\text{зд}| \quad xd\,yd] \tag{17}$$

xd и yd — размер нового прямоугольника, охватывающего повернутый эллипс. Если |зд | равно xd yd , то эллипс становится прямой диагональной линией.

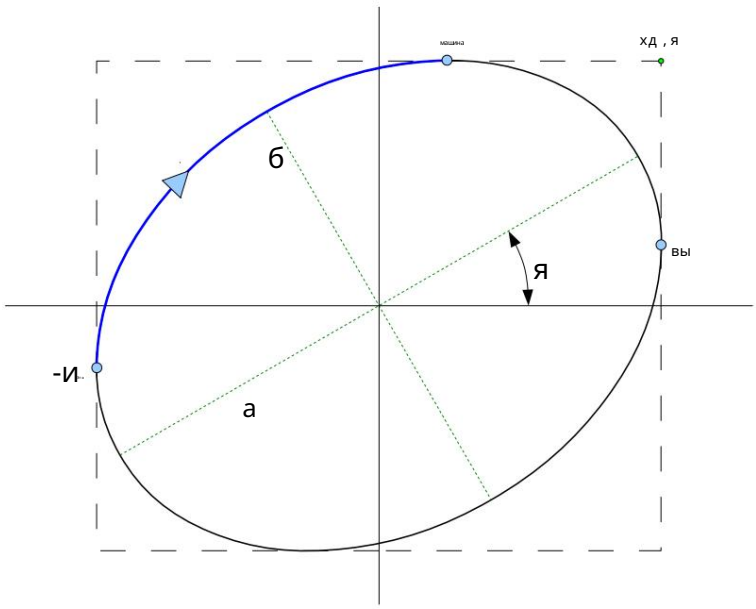


Рисунок 16: Повернутый эллипс

Значения максимумов (точки контакта с прямоугольником) составляют xe yd = xd ye = зд.

Полезные соотношения : 2 a 2=xd 2 + yd ^{2} + $\sqrt{(xd\,2 \quad yd\,2)^{2} + 4\,зд.^{2}}$, 2b 2=xd 2) + yd 2 $\sqrt{(xd\,2 \quad yd\,2 \quad + 4\,зд.^{2}}$.

Алгоритм разделен на две части. Первая часть вычисляет xd, yd, zd и вывывает вторуючасть по параметрам углов прямоугольника. Этот процесс привязывает кривуюэллипса к целочисленным значениям и снова позволяет рисовать повернутые эллипсынечетных диаметров или без вычисления тригонометрических функций.

4.4 Рациональные эллипсыБезье

Алгоритм сталкивается с той же проблемой, что и квадратичные кривые Безье. Если вторая половина плоского эллипса подойдет слишком близко, алгоритм потерпит неудачу. Умное решение предыдущей реализации на этот раз невозможно использовать, поскольку у эллипс представляет собой замкнутую кривую Проблемывозникают на двух концах плоского эллипса. Чтобыиспользовать то же решение, алгоритм должен начать с серединыи двигаться к узким концам. Это станет сложной программой. Другой подход, разработанный ранее, заключается в увеличении разрешения растра за счет более мелкой сетки. Этот растр высокого разрешения должен быть достаточно мелким, чтобыизбежать конфликта двух кривых на одном пикселе или очень близком пикселе.

Другим решением этой проблемы является использование уже существующего рационального алгоритма Безье. Эллипс также можно рассматривать как композицию четырех рациональных кривых Безье. Сравнивая членыдвухнезависимых уравнений эллипса и рационального Безье, вес P1 равен 2 = xd yd±zd для длинной/короткой стороныэллиптического сегмента. 2 хорда рассчитывается по w ———————

Программа рисования повернутых эллипсов теперь может делегировать процесс рисования подпрограмме.

```
voidplotRotatedEllipse (int x, int y, int a, int b, float angular) {
                                        /* построить эллипс, повернутый на угол (радианы) */ float xd
    = (long)a*a, yd = (long)b*b; float s = sin(угол), zd = (xd-
    yd)*s; /* вращение эллипса */ xd = sqrt(xd-zd*s), yd = sqrt(yd+zd*s); /* окружающий прямоугольник */
    a = xd+0.5; б = ядро+0,5; zd = zd*a*b/(xd*yd); /* масштабируем до целого числа */
    plotRotatedEllipseRect(xa,yb, x+a,y+b, (long)(4*zd*cos(angle)));

}

voidplotatedEllipseRect (int x0, int y0, int x1, int y1, long zd) {
                    /* прямоугольник, охватывающий эллипс, целочисленный угол поворота */
    int xd = x1-x0, yd = y1-y0; float w =
    xd*(long)yd; if (zd == 0)
    returnplotEllipseRect (x0,y0, x1,y1); /* выглядит лучше */ if (w != 0.0) w = (w-zd)/(w+w); /* квадрат веса
    P1 */ Assert(w <= 1.0 && w >= 0.0); /* ограничиваем угол до |zd|<=xd*yd */ xd = floor(xd*w+0.5); yd =
    пол(yd*w+0,5); /* привязать xe,ye к int */ plotQuadRationalBezierSeg(x0,y0+yd, x0,y0, x0+xd,y0, 1.0-
    w); plotQuadRationalBezierSeg(x0,y0+yd, x0,y1, x1-xd,y1, w); plotQuadRationalBezierSeg(x1,y1-yd, x1,y1,
    x1-xd,y1, 1.0-w); plotQuadRationalBezierSeg(x1,y1-yd, x1,y0,x0+xd,y0,w);


}
```

Листинг 14. Программыдля построения повернутых эллипсов

Программа в листинге 13 работает даже если P0 и P1 поменять местами.

Единственным недостатком этого решения является то, что эллипс не всегда точно симметричен. Иногда кривая находится между двумя пикселями, и алгоритм рисования всегда округляется в одном направлении (например, нижний пиксель). В случае эллипса это означает, что одна сторона сегмента закруглена внутрь, а другая симметричная сторона наружу.

Кривая становится гиперболой, если знак квадратичного терма a^2 или b^2 предыдущих уравнений эллипса отрицателен:

$$^{2}x\ (a\ ^{2}\sin 2\theta\quad b^{\text{22потомучто}}\theta)\quad 2xy\ (a\quad ^{2}+6\ ^{2})\sin\theta\cos\theta+y\quad ^{2}\ (a^{\text{22потомучто}}\theta\quad b\ ^{2}\sin 2\theta)+a\ ^{2}62=0.$$

5 кубических кривых Безье

Готовыликвыкретьему образованию? Растеризация кубических кривыхтребует немного больше математики. Кубический метод Безье может оказаться весьма сложным, как показано на рисунке 17. Петли и выступы доставляют множество неприятностей. Возможно, было бынеплохимрешениемразделить кубическую кривуюна короткие кривые с помощью алгоритма де Кастельжо и преобразовать их в квадратичные кривые Безье, которые можно было быпостроить с помощью упредыдущего алгоритма.

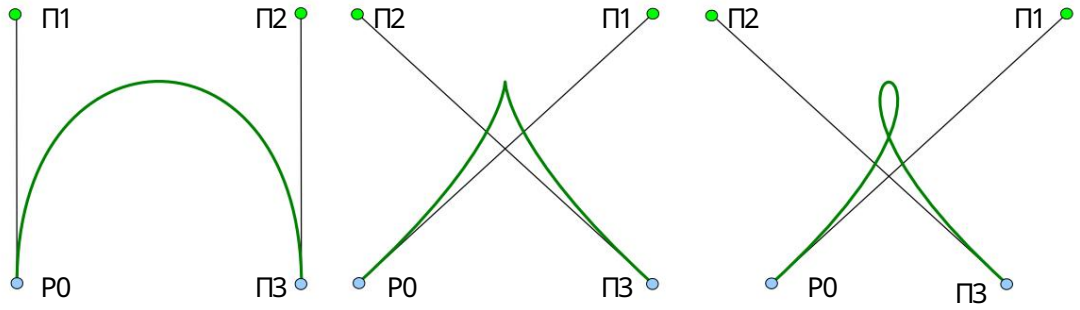


Рисунок 17. Различные кубические кривые Безье.

5.1 Уменьшение кубического градуса

В обычных случаях невозможно точно уменьшить степень кривой Безье. Только если кубический член обращается в нуль, уравнение (29) можно использовать для получения приведенного квадратного уравнения.

Однако можно аппроксимировать кубическое Безье квадратичным Безье. Чтобыотклонение было небольшим, целесообразно провести подразделение в три этапа:

- кривая подразделяется на горизонтальные и вертикальные стационарные точки (глава 5.9)
- кривая подразделяется на точку перегиба (глава 5.7)
- оставшаяся кривая подразделяется на две квадратичные точки Безье

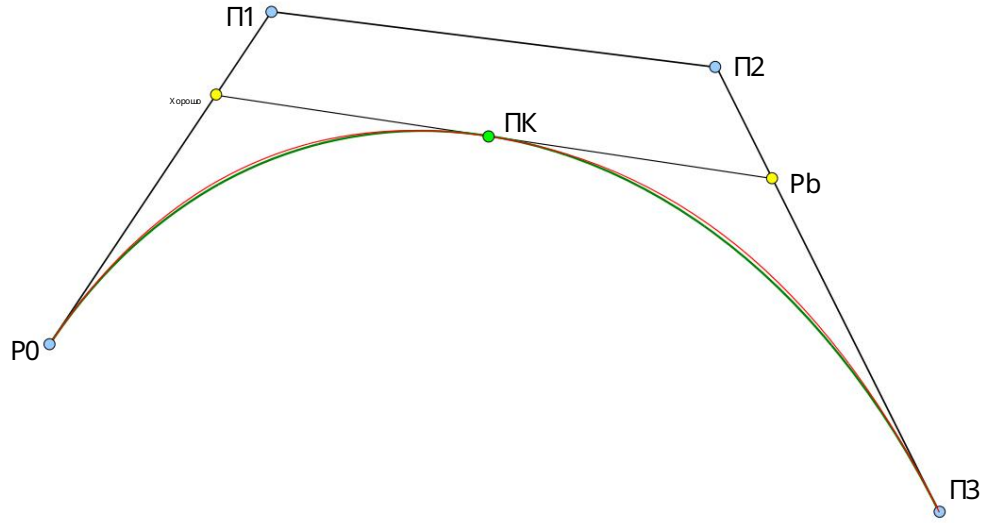


Рисунок 18: Приближение кубического Безье (красный) двумя квадратичными (зеленый).

Конечно, есть несколько возможностей для понижения степени и подразделения . Точный метод гарантирует, что кривизна в конечных точках аппроксимации не изменится, что также упрощает расчеты

Следующие соображения позволяют сделать отклонения небольшими. Кубическая кривая Безье P0-P1-P2-P3 на рисунке 18 делится ровно пополам (при t = ½) путем деления Де Кастельжу. Точка подразделения Pc остается точно на кубической кривой Безье. Теперь касательные в конечных точках P0 и P3 разделенной кубики и двух квадратичных кривых P0-Pa-Pc и Pc-Pb-P3 сделаныравными по направлениюи величине.

Если кубический сегмент Безье P0-Pa-Pc получен разделением Де Кастельжу, то касательная в точке P0 составляет P'(0) = 3(P1-P0)/2 , а тангенс квадратичного сегмента Безье P0-Pa-Pc составляет 2(Па-П0). Если сделать эти два тангенса равными, точки станут : Pa = (P0 +
3P1)/4, Pb = (3P2 + P4)/4 и Pc = (Pa+Pb)/2. (18)

Если аппроксимация квадратичными кривыми Безье недостаточно точна, кубическая кривая должныбыть растрированы

5.2 Полиномиальные результаты

И снова необходимо неявное уравнение кубической кривой Безье: [Marsh, 2005]

$$B3(t) = (1-t)^3 P0 + 3(1-t)^2 t P1 + 3(1-t) t^2 P2 + t^3 П3. \quad (19)$$

Для вывода общего неявного уравнения третьего порядка

$$\text{топор}3 + 3бx^2 y + 3cxyy + d^3 + 3ex^2 + 3fxy + 3gy^2 + 3hx + 3iy + j = 0 \quad (20)$$

необходимыдействительно линейно независимых уравнений. Это кажется слишком сложным, поэтому давайте попробуем другой подход.

Результаты — мощный инструмент для вычисления общих корней двух (или более) полиномов без трудоемкой работы по поиску всех отдельных корней. Основная идея состоит в том, чтобы найти набор линейно независимых полиномов для применения теории линейных систем уравнений. Если все f_i и g_j являются корнями двух многочленов $f(t) = \sum_{i=0} a_i t^i$ и $g(t) = \sum_{i=0} b_i t^i$, то результат определяется как произведение разностей их корней. $R(f, g) = \prod_i \prod_j (f_i - g_j)$ и может быть рассчитано с помощью определителя матрицы Сильвестра или Безу. [Безу, 1764 г.]

Безу впервые заметил, что «...определенное уравнение всегда можно рассматривать как результат двух уравнений с двумя неизвестными, когда одно из неизвестных исключено». (Лейбниц уже использовал результаты, хотя никогда не публиковал свои выводы) Проблема также связана с алгоритмом НОД Евклида или методом исключения Гаусса.

Если два многочлена f(t) и g(t) имеют общий корень t_0, то уравнение $f(t)\,g(s) - f(s)\,g(t) = 0$ всегда будет выполняться для любого значения s. Уравнение также должно содержать $(s - t)$ в качестве множителя, поскольку ему будет удовлетворять $t = s$, даже если нет общего корня. После деления на $(s - t)$ уравнение можно рассматривать как полиномиальное по s, где коэффициент каждого члена является полиномом по t. Поскольку при общем корне $t = t_0$ все выражение должно обращаться в нуль при любом значении s, каждый из полиномов-коэффициентов по t должен обращаться в нуль при t_0. [Кейли, 1867 г.]

Матрица коэффициентов для двух одномерных полиномов вычисляется по выражению Кэли

$$R = \det \frac{f(t)\,g(s) - f(s)\,g(t)}{t - s} = \det_{i,j=0}^{n-1} \left(\sum_{k=\max(0,i-j)}^{\min(i,n-j-1)} a_{i-k}\, b_{j+k+1} - a_{j+k+1}\, b_{i-k} \right) = c_{ij}. \tag{21}$$

Это уравнение также можно записать в матричной форме

$$\begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0,n-1} \\ c_{10} & c_{11} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ \vdots \\ t^{n-1} \end{bmatrix} = 0.$$

Квадратная матрица этого уравнения называется матрицей Безу многочленов f(t) и g(t).
Определитель этой матрицы равен нулю только в том случае, если существует общий корень. Поскольку выражение должно обращаться в нуль при любом значении s, результат создается матрицей коэффициентов n однородных линейных уравнений.

Общий корень $t = t_0$ существует только в том случае, если (22)

$$\begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0,n-1} \\ c_{10} & c_{11} & \cdots & c_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1,0} & c_{n-1,1} & \cdots & c_{n-1,n-1} \end{bmatrix} \begin{bmatrix} 1 \\ t_0 \\ \vdots \\ t_0^{n-1} \end{bmatrix} = 0$$

Может быть удивительно рассматривать это как линейные уравнения, но в этом случае неизвестными являются просто степени t, и результат будет тождественно равен нулю тогда и только тогда, когда матрица коэффициентов n однородных линейных уравнений степени t равна все ноль.

Самый простой пример имеет первую степень : f t =a1 t a0=0, g t =b1 t b0=0.

$$\frac{(a1\ t+ a0\ )(b1\ s+ b0\ )(a1\ s+ a0\ )(b1\ t+ b0\ )}{\text{черт возьми}}=c00=a0\ b1\ \ a1\ b0=0.$$

Результат второй степени вычисляется по формуле f t =a2 t 2 a1 t a0=0, g t =b2 t 2 b1 t b0=0.a(12t+ a0)(b2 s 2 + b1 s+ b0)(a2 s 2 + a1 s+ a0)(b2 t 2 + b1 t+ b0) =(c11 t+ c01)s+ (c01 t+ c00)=0.

$$\frac{}{\text{черт возьми}}$$

Матрица Безу делает тогда [c00c01c01c11][1 t] = [a0 b0a0 b1a1 b0a1 b1a0 b2a2 b0a0 b2a1 b2a2 b1][1 t] =0.

Результат двух кубических полиномов вычисляется по формуле

f t =a3 t 3 a2 t 2 a1 t a0=0, g t =b3 t 3 b2 t 2 b1 t b0=0.

$$\frac{f(t)g(s)\ \ f(s)g(t)=(c22}{\text{черт возьми}}t^2+c12\ t+c02)c^2+(c12\ t^2+c11\ t+c10)s+(c02\ t^2+c01\ t+c00)=0.$$

Результатом матрицыБезу является

$$\begin{bmatrix} a0\ b1\ \ a1\ b0 & a0\ b2\ \ a2\ b0\ a0\ b3\ \ a3\ b0\ a0\ b2\ \ a2\ b0\ a1 \\ & a2\ b3\ \ a3\ b2 \end{bmatrix}\begin{bmatrix} t \\ 1 \end{bmatrix}^2=0.$$

5.3 Неявное кубическое уравнение Безье

Теперь результаты также можно применять для преобразования параметрических уравнений кривой в неявная форма f(x,y)=0 для любого значения t: R(f , (23)

$$\text{ж})=\ c00\ \ 1c0,\ n1\ 1=0.\ \ 1,0$$

Уравнение рациональной кривой Безье степени n составляет

$$\text{Бн}(\text{т})=\frac{\text{я мы}(\text{я})(1\text{-t})\text{я}}{\text{мы}(\text{я})(1\text{-t})}.\qquad(24)$$

Общий корень двух параметрических многочленов

$$\text{ж}x\ \ t=\sum_{\text{я буду n если}\ \ 1\ t}^{n\ i\ \text{я}}\text{т}\ \ x\ xi=0\ \text{и}\ f\ \ \text{и}\ t=\sum_{\text{я буду n если}\ \ 1\ t}^{n\ i\ \text{я}}\text{т}\ \ y\ yi=0$$

построим неявное уравнение кривой Безье f(x,y)=0 по x и y. fx(t) — полином от t , коэффициенты которого линейны по x, а fy(t) — многочлен от t , коэффициенты которого линейны по y. Любое значение x и y , для которого f(x,y)=0, приводит к тому, что результирующая fx(t) и fy(t) становится равной нулю, следовательно, является частью параметрической кривой.

Форму Бернштейна многочлена необходимо преобразовать в степеннуюформу с помощью

$$\text{я } wi(n\text{я})(1\ \ t) \qquad \text{т я} = \text{я}\ (\text{пя})(\ \ k\,wk(\text{я}\ i\ \ k)\,tk)(\ \ 1)$$

Коэффициентыai и bi полиномов f(t) и g(t) матрицы(22) можно дополнительно упростить из-за линейностиопределителей. Добавляя к каждой строке и столбцу m определителя все предыдущие строки и столбцы, эта сумма становится

$$\text{я ми к нед я}\quad 1 \qquad \text{я -к}\ x\ xk\ =wm\ \ x\ xm\ .$$

Полиномиальные матричные коэффициентыкривой Безье составляют тогда

$$ai=wi\ \ n\,i\ \ x\ xi \quad \text{и}\, bi=wi(n\ i)(y\ \ yi).$$

Третьюстепень кубической кривой Безье теперь можно было бынеявно представить следующимобразом:

$$\text{ж}(x,y)= \begin{matrix} d01 \\ \text{дд}\ d03\ d13 \end{matrix} \qquad \begin{matrix} d02\ d\ d02\ d\ 03+\ 03 \\ 12 \qquad 13 \\ d\ 23 \end{matrix} \quad =0\ \text{где}\ dij=(\ 3\ i)(\ 3\ j)\ \ xy\ 1\ 1 \qquad \begin{matrix} \text{Си И1} \\ \text{икс} \\ \text{джйдж} \end{matrix} \tag{25}$$

К сожалению,членыэтого неявного уравнения (20) довольно запутаны, например

$$a=9\ y01\ (\ y03\ y23+9\ y12\ y23\ \ \ 3\ y13^2)\ \ 27\ y02^2(\ y03+9\ y12\ )\ y23+18\ y02^2 y03\ y13\ \ \ y03^2$$

где yij = yi – yj. Возможно ли более простое выражение?

Ключ к успеху – симметрия . Но определение кривых Безье асимметрично; параметр t варьируется от нуля до единицы

Его необходимо изменить на симметрично определенный диапазон параметра от минус одного до плюс одного:

$$B3\ (\ ^t)=(1\ \ \ ^t)\ 3\ P0+3(1\ \ \ ^t)\ 2\ (1+^t)P1\ +3(1\ \ \ ^t)(1+^t)\ 2\ P2+(1+^t)\ 3\ P3 \text{ для } [\ \ 1\ \ \ ^t\ \ \ 1].$$

Чтобыупростить расчеты, константы кубического уравнения Безье преобразуются следующимобразом:

$$\begin{matrix} xb \\ [xa] \\ xc \end{matrix} \ \text{"="} \ \begin{matrix} 1 & 1 & 1\ 1 \\ [\ 1 & 3\ 3 & 1\ 1 \\ 1\ 3 & & 1\ 3^1 \end{matrix} \ \begin{matrix} x1 \\ [x0x3] \\ x2 \end{matrix} \quad \text{и} \quad \begin{matrix} a \\ \text{Мистер} \\ [\text{кукуруза}] \\ yc \end{matrix} \ \text{"="} \ \begin{matrix} 1 & 1 & 1\ 1 \\ [\ 1 & 3\ 3 & 1\ 1 \\ 1\ 3 & & 1\ 3^1 \end{matrix} \ \begin{matrix} y1 \\ [y0y3] \\ y2 \end{matrix} . \tag{26}$$

Уравнения Безье принимают следующий вид: $8\ x= \ \ ^t\ 3\ xa+3\ ^t\ 2\ xb \ \ 3^t\ xc+xd$ и $8\ y= \ \ ^t\ 3\ ya+3\ ^t\ 2\ yb \ \ 3^t\ yc+\ yd$.

Обратное преобразование делает :

$$\begin{matrix} x1 \\ x2 \\ [xa] \end{matrix} \ \text{"="} \ \begin{matrix} 3\ 3 \\ 1 \quad 1\ 1 \\ 1\ 1\ 1 \quad 1 \quad 1\ 1 \\ 1\ 3 \quad 3\ 1][x\ a\ xd\ ]8[\ 1 \end{matrix} \ \begin{matrix} xb \\ xc \\ [xd] \end{matrix} \quad \text{и} \quad \begin{matrix} y1 \\ y2 \\ [ya] \end{matrix} \ \text{"="} \ \begin{matrix} 3\ 3 \\ 1 \quad 1\ 1 \\ 1\ 1\ 1 \quad 1 \quad 1\ 1 \\ 1\ 3 \quad 3\ 1][y\ a\ yd\ ]8\ 1 \end{matrix} \ \begin{matrix} \text{Мистер} \\ yc \\ [yd] \end{matrix}$$

Матрица постоянного преобразования аналогична биномиальной матрице Bn

Б1=[1 1 1] , B2= [1 1 0 1 1 2 2 1] , B3= [1 1 3 3 1 1 1 1 1 1 3 3 1 1 1] , B4= [1 1 2 4 0 2 1 1 1 -4 6 -4 1 1 0 1 1 2 0 2 1]

(n) с определением пунктов brc строки и г и столбца с как:

brc (n)= (1) я (р я)(п-р с-i) .

Выражения стали еще проще за счет введения констант

сий=| xi yi | =xi y j x j yi . (27)

Члены нея вного кубического уравнения Безье (20) для f(x,y)=0 затем вычисляются по формуле:

3 2 , e=cad of 2 3(2cab yc+ cac yb) of+ 9 cab yb , b =xa of , d=xa of a=of , c=xa 3 2 ,

f =(2 cad+ 3cbc) xa ya+ 9(2 cab xb yb+ xb xc ya 2 xa 2 yb y c), g=cad xa 2 3(2cab xc+ cac xb) xa+ 9cab xb 2 ,

2 ya+ 6(3 cab cbd cac cad) yb+ 3(3cac 4cab cad 9 cab cbc) yc+ 9cab cac yd , h=cad 2 cbd ccd) xa+ 6(3 cab cbd cac cad) xb+ 3(3cac 4 cab cad 9cab cbc)xc+ 9 cab cac xd i=cad 9 cad (cac cbd +2 cab ccd)+27(cab 3 81cab cbc ccd . и j=cad +cac 2 2 (28)

Последний член j не нужен алгоритму.

Большинство характерных свойств кривой Безье можно определить только значениями cab, cac и cbc (кривизны кубической Безье). Обратите внимание, что cij меняет свой знак, если i и j (или x и y) меняются местами.

Если xa и ya равны нулю, то кубический член уравнения Безье исчезает и кривая принимает приведенную квадратичную (или даже меньшую) форму: B3 t = 1 t (точное ^{2}P3 1 t t 3P1 P0 t^2 P0 (29)

приведение степени).

5.4 Расчет кубической погрешности

Уравнение расчета погрешности для кубической кривой Безье имеет вид 2 d

x , y =a x 3 3b x e ^{2} xyy y 3c 3 3e x 2 3 fx y 3 gy 2 3h x 3 i y j

К счастью, это сложное выражение необходимо оценить только при инициализации. Во время пиксельного цикла вычисления требуют только дополнений из-за прямой разницы

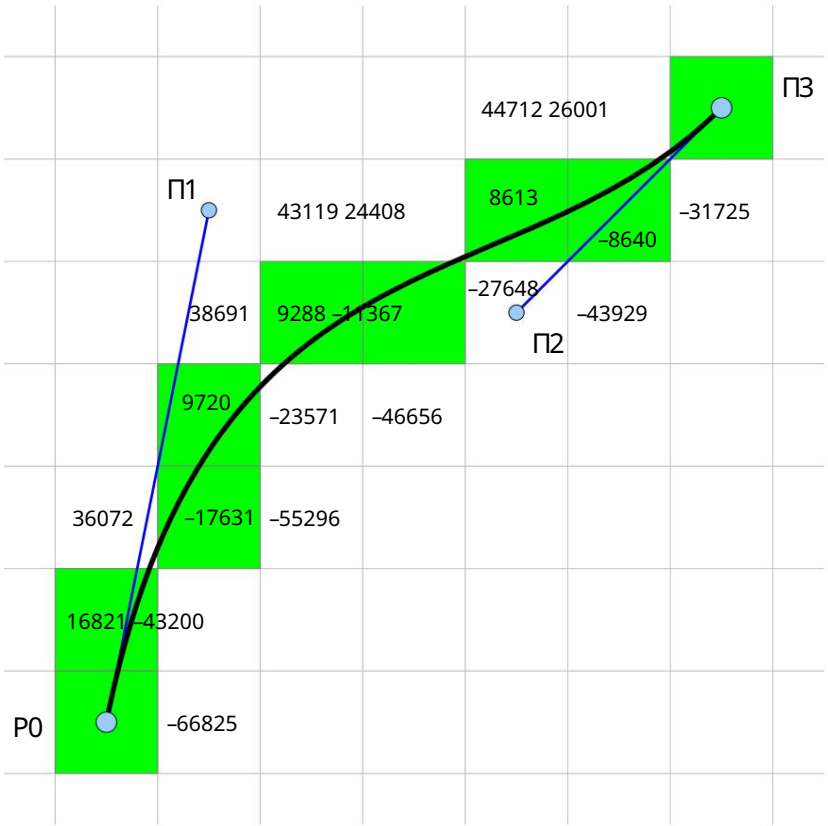


Рисунок 19: Зна чения ошибок кубич еской кривой Безь е

Шаги приращения в направлении x сост авля ют : err += d , Д += Д , Д += Д ,

Д = 3 =6 a=6 да , dxy += x = 6 b = 6 из , Д = 2=6 c=6 хайа .

И в направлении y: err += d , Д += Д , Д += Д ,

Д = 3= 6 d= 6 xa , Д = 2=6 c=6 x a y , x Д = = 6b= 6 ха у и .

При вын ислении раз ност ей первог о и вт орог о поря д к а форвардные раз ниц ыслед ют з ак ономерност и. Раз ност и з нач ений ошибок пред ст авля ют собой к онеч ную поря мую раз ност ь n-г о поря д к а.
сущест вует в д вух из мерения х:

$$Д_{nx\,n\,y=} \sum_{я\ x=0}^{нх} \sum_{я\ y=0}^{т\,от} (\ 1)^{я\ x+я} \qquad) [nx+ ny = n]. \tag{30}$$

$$y\,(^{я}nx\,x\,)(^{я}ny\,y\,)\,e\,(x+\,nx\quad i\,x\,,\,y+\,n\,y\quad i\,y$$

Зна чения иниц иализ ац ии переменных приращения д ля P0 д елают к онеч ными первое и вт орое з нач ения .
раз ниц ыв з ак аз е вперед : (31)

d x=e x0 1, y0 =3 a x0 x0 1 y0 c y0 f 2 x0 1 b y0 e h a= =27 (ya+ 2 yb+ yc)(cab)+ 27(yb 2 ya
2 2 2 3
yc)cab 9 ya (cab+ cac+ cbc)+ ya + cab cac 3 27ab xbc 2 xa cy=ecabx0 9,y0a 1 = 3 d y0 y0 1 x0 b x0 f 2 y0 1 c x0 g i d= ,

= 27 xa 2 xb xc cab cab cac cbc xa cab cac 3 cab cbc cac
2 2 2 3
,

d xx = e x0 2, y0 2e x0 1 , y0 = 6 a

2 cab cac 3 я 2 cab yc cac yb 9cab yb 2 ,

d xy=e x0 1, y0 1 e x0 1, y0 e x0 , y0 1 = 3 b 2 x0 1 c 2 y0 1 f =

=3 xa из 6 cab cac 3 cbc xa of 9 xb xc из 2 xa 2 yb yc 2 cab xb yb

, d yy=e x0 , y0 2 2 и x0 , y0 1 =6 c x0 d y0 1 g =6 cab cac 3 xa 2 cab xc cac xb 9cab xb 2 .

3 3 =6 =xa 2 x

Эти значения не зависят от xd или yd. Расчет ошибки может быть выполнен в целых числах.

Но значения могут достигать весь ма больших значений (вплоть до шестой степени). Переменные dxx, dxy, dyy требуют четырехкратного размера исходного слова, dx и dy — пятикратного, а значения ошибки — шестикратного.

5.5 Точка самопересечения

Для точки скругления или самопересечения мы хотим найти два соответствующих параметра t1 и t2 такие, что B3(t1) = B3(t2), t1 t2.

Проблему точки самопересечения можно решить путем подразделения или поиска корня . Но есть и алгебраическое решение этой проблемы

Точка самопересечения — это седловая точка, в которой производная неявного уравнения кривой равна нулю в любом направлении. Для расчета этой точки производная неявного уравнения параметризуется :

xy= 3 ax 2 6 bx y 3 cy 2 6 e x 3 f y 3 h , x 2 6c x y 3 dy (32) 2 3 f x 6 g y 3i 3b

Эта производная заменяется параметрическим уравнением

x= t 3 xa 3 t 2 xb 3 t xc xd и y= t 3 ya 3 t 2 yb 3 t yc yd .

Два общих корня этих двух производных — это параметры t1 и t2 точки самопересечения . Кажется , что это многочлен шестой степени, но, к счастью, в этом случае две высшие степени исчезают .

Точка самопересечения кубической кривой Безье вычисляется по двум общим корням двух многочленов : такси xc

т 4 такси

2 2

2 2

t 6 xb cab cbc 3 xb cac xa cac cbc xc cac 3 cab cbc =0, 2 t 3 cab ya

2 2

t 4 ya cab cac 2 yb cab t 2 ya cac ya cab cbc 3 yc cab

2 2

t 6 лет cab cbc 3 yb cac ya cac cbc yc cac 3 cab cbc =0.

Итерацию Ньютона-Рафсона можно использовать для вычисления корней многочленов. Но для нахождения общих корней двух многочленов результат Безу снова помогает получить аналитическое решение.

Даны два многочлена f t = ai t я , g t = bi t я однородное линейное уравнение Результат Безу четвертой степени составит :

$$\begin{bmatrix} d_{01} & d_{02} & d_{03} & d_{04} \\ d_{12} & d_{13} & d_{14} & d_{24} \\ d_{03} & d_{23} & d_{24} & d_{34} \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix} = 0 , \text{ где } d_{ij} = a_i b_j - a_j b_i \quad (33)$$

Многочлены f(t) и g(t) имеют общий корень тогда и только тогда, когда определитель R(f,g) равен нулю. Эти корни можно найти, выполнив метод исключения Гаусса в строках R(f,g). Если после исключения последняя ненулевая строка равна (0, ..., 0, h0, h1, ..., hk), то общий корни f(t), g(t) — это просто корни многочлена $h(t) = \sum h_i t^i$ [Голдман и др., 1985].

Рисунок 20: Трехмерный график поверхности кубической кривой Безье с самопересечением.

Матричные коэффициенты результата Безу кубической кривой Безье составляют:

$d_{01} = 2 c_{bc} (3 c_{ab} c_{bc} - c_{ac}) d_{02} = -c_{ac}(c_{ab} c_{bc} - c_{ac})(3 c_{ab} c_{bc} - c_{ac}^2)$, $d_{03} = -c_{ab} (2 c_{ab}^2 c_{bc} + c_{ac})(3 c_{ab} c_{bc} - c_{ac})$, $d_{04} = c_{ab} c_{ac} (3 c_{ab}^2 c_{bc} - c_{ac}^2)$, $d_{12} = -3 c_{ab} (5 c_{ab}^2 c_{ac} c_{ac} (4 c_{ab} c_{bc} - 3 c_{ac}^2)$, $d_{13} = c_{ab}^2 c_{bc} - 8 c_{ab}^4 c_{bc} - c_{ac}^2$ $d_{14} = -3 c_{ab}^3 (2 c_{ab} c_{bc} - c_{ac}^2)$, $d_{23} = c_{ab}^3 (8 c_{ab} c_{bc} + c_{ac}^2)$, $d_{24} = -3 c_{ab}^4 c_{ac}$, $d_{34} = 2 c_{ab}^5$.

Эти значения зависят только от c_{ab}, c_{ac} и c_{bc}. Определитель этой результирующей матрицы всегда равен нулю.

Исключив одну строку и столбец матрицы, получим квадратное уравнение для вычисления корней точки самопересечения. Принимая строки 2 и 3 за параметр самопересечения воздействия составляет: $t_{1,2} = \frac{c_{ac} \pm \sqrt{c_{ac}^2 - 12 c_{ab} c_{bc}}}{2 c_{ab}}$. (34)

Обратите внимание, что t_1 и t_2 также не зависят от xd, yd, которые подобны центру кривая Безье.

Для 4 кабин cbc cac^2 кубическое нелинейное уравнение имеет комплексную точку самопересечения

Если 4 cab cbc=cac имеет точку у^2 кривая имеет точку возврата. Если |^t1,2|<1 , то кубический сегмент Безье (acnode). самопересечения (crunode). В противном случае только одно нелинейное уравнение вне интервала параметров имеет самопересечение.

5.6 Градиент в точке P0

Градиент кривой в точке P0 может быть положительным или отрицательным в направлении графика. Поскольку алгоритм зависит от непрерывно положительного (или отрицательного) наклона, значения необходимо инвертировать в случае отрицательного градиента, чтобы иметь возможность использовать тот же алгоритм. Как рассчитать градиент в точке P0?

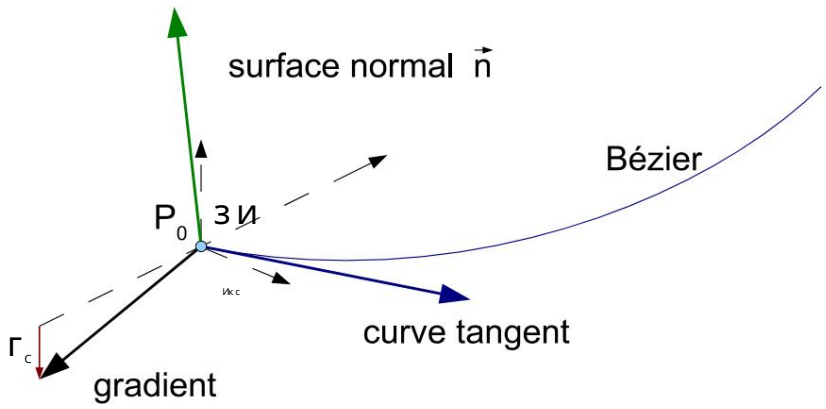


Рисунок 21: Градиент кривой

Градиент представляет собой векторное произведение нормали к поверхности и вектора касательной кривой.

Нормаль к поверхности уравнения e(x,y) = z на рисунке (21) составляет

$$n=[\frac{\quad}{e(x,y)},-\frac{e(x,y)}{y},1].$$

При определении класса g= 4 $\frac{27}{}$ ((cab+ cac) 2 cab (cac+ 3cbc)) производные

уравнение составить $\frac{e\ x0,y0}{x}$ =g y0 y1 и $\frac{e\ x0,y0}{}$ = g x0 x1 . Градиент в точке y P0 делает тогда

$$e\ x0,y0 = n\times[x1\ x0,y1\ y0,0]=[y0\ y1,x1\ x0,g\ x0\ x1\ 2\ y0\ y1\ 2].$$

Компонент z (gz на рисунке 21) градиента (пропорциональный g) отрицателен только в том случае, если P0 является частью цикла самопересечения . Если класс g равен нулю, то точка P0 сама является крунодой (или каспом).

Эту информацию можно использовать, чтобыпроявлять особуюосторожность, поскольку рисование цикла самопересечения всегда включает в себя опасность того, что другая часть кривой подойдет слишком близко для правильной работыалгоритма.

5.7 Точка перегиба

Положение точек перегиба становится важным, если кубика Безье аппроксимируется наборами связанных квадратичных отрезков Безье. Кубическая кривая Безье меняет направление изгиба в точке перегиба.

Точка перегиба — это точка, в которой кривизна кривой равна нулю Кривизна (t) ˙f кривой рассчитывается по формуле

$$к = р \frac{1}{д\,ж/д\,с} = \frac{B'(t)\times B''(t)}{B'(t)^3} = \frac{\dot{f}_{икс}(т)\,\ddot{\phi}_{и} - \dot{f}_{и}(т)\,\ddot{\phi}_{икс}(т)}{(\dot{f}_{икс}^{2}(т) + \dot{f}_{и2(т))3}} . \quad (35)$$

В случае кубической кривой Безье точку перегиба можно определить с помощью квадратного уравнения $t^2 ab + tac + cbc = 0.$ (36)

5.8 Кубические проблемы

Алгоритм снова сталкивается с проблемами для некоторых кривых Безье. На рисунке 22 показана кривая с амбициозными значениями ошибок при P0. Ситуация становится понятной, если построить всю кривую Точка P0 находится рядом с точкой самопересечения.

Алгоритм сталкивается с настоящей дилеммой с плоскими циклами самопересечения. Он не может строить график со стороныпересечения, так как там значения ошибок меняют знак. Если алгоритм хочет построить график со стороны плоского кончика, существует опасность того, что другая часть кривой окажется слишком близко и значения ошибок будут слишком запутанными. Алгоритм не работает, если кривая содержит плоскую петлюсамопересечения.

На рисунке 22 показана кривая, на которой петля очень плоская. В этом случае оба конца могут представлять собой острые углыи вести в узкий или одинаковый пиксель. Алгоритм не сможет построить кривую, если оба конца начинаются с таких запутанных значений ошибок. Два варианта могут решить проблему. Первое будет таким же, как и для квадратично-рационального Безье. Кривая дополнительно подразделяется в самом широком месте контура, и график начинается там. Из-за этого решения кривые иногда становятся немного резкими. Другой вариант — использовать для графика растр более высокого разрешения.
Разрешение выбирается по длине ног. Этот метод немного снижает скорость построения, но приводит к более плавным кривым.

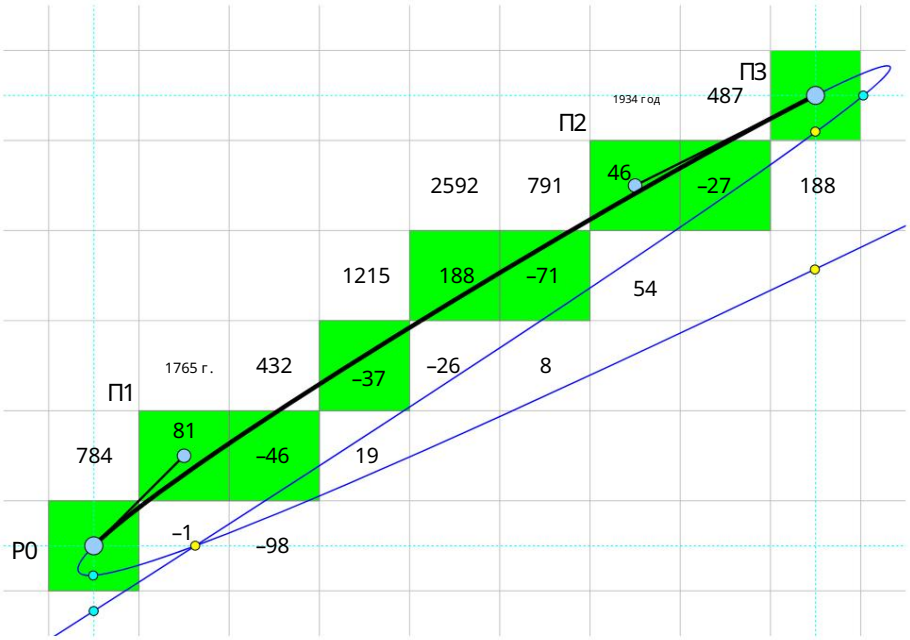


Рисунок 22: Кубический Безье с запутанными значениями ошибок

5.9 Кубический алгоритм

Базовый алгоритм требует непрерывно возрастающей или нисходящей кривой. Непросто даже включить проверку параметров, чтобыобеспечить это условие. Во-первых, линии P1-P0 и P3-P2 либо растут, либо падают. Но наклон P2-P1 можно было бысвести на нет по отношениюк остальным, и кривуювсе равно можно было быпостроить. Таким образом, второе условие должно состоять в том, чтобылибо линия P3-P0 имела тот же наклон, что и P2-P1, либо первая производная параметрического уравнения имела

настоящие корни.

Оператор утверждения также должен учитывать определенные ошибки округления. Если параметр имеет тип одинарной плавающей запятой с 23-битной мантиссой и параметр координат не превышает 16 бит, то эпсилон должен быть больше 2^{16-23}.

Для простого (и более быстрого) кода вычисление dx и dxx выполняется на один шаг вперед. Расчет dy и dyy выполняется на один шаг вперед. Вычисление значения ошибки выполняется на один шаг х и один шаг вперед.

Кубическая кривая Безье может иметь точку возврата или петлюсамопересечения. Даже если петля невидна напрямую с конкретным параметром, вся кривая может иметь петлю, и видимый сегмент может быть ее частью Эта точка возврата или петля вызывает множество проблем. Алгоритм не может строить график через самопересечение. Поэтому сюжет начинается с обоих концов. Если значения ошибок путаются из-за того, что градиент меняет свой знак, то пробуется другой конец.
Следовательно, функция способна обрабатывать кривые Безье с точками самопересечения.

Другая проблема возникает, если эта петля очень плоская, как на рисунке 22. Тогда один конец кривой, вероятно, заблокирован точкой самопересечения, а на другом конце слишком запутанная точка.

значения ошибок для алгоритма. В этом случае разрешение пикселя увеличивается.

Алгоритм использует значение оценки, чтобыпроверить, нужно ли увеличить коэффициент разрешения.

Алгоритм также смотрит на один пиксель вперед, чтобыобнаружить точку самопересечения (dx > 0 или dy < 0). Но в начальной точке эти значения все еще могут быть инвертированына один пиксель вперед, хотя для текущего момента это не так. Поэтому алгоритм должен проверить значения в текущем пикселе (dx > dxy или dy < dxy) и переключиться на проверку на один пиксель вперед, только если значения там стал действительным.

```
voidplotCubicBezierSeg (int x0, int y0, float x1, float y1,
                        float x2, float y2, int x3, int y3) /* построение
{                                                           ограниченного кубического сегмента Безье */
    int f, fx, fy, leg = 1;
    int sx = x0 <x3? 1:-1, sy = y0 < y3? 1:-1; /* направление шага */
    float xc = -fabs(x0+x1-x2-x3), xa = xc-4*sx*(x1-x2), xb = sx*(x0-x1-x2+x3);
    float yc = -fabs(y0+y1-y2-y3), ya = yc-4*sy*(y1-y2), yb = sy*(y0-y1-y2+y3);
    двойной ab, ac, bc, cb, xx, xy, yy, dx, dy, ex, *pxy, EP = 0,01;

                                                            /* проверка ограничений кривой */
    /* наклон P0-P1 == P2-P3 и (P0-P3 == P1-P2 или без изменения наклона) */
    Assert((x1-x0)*(x2-x3) < EP && ((x3-x0)*(x1-x2) < EP || xb*xb < xa*xc+EP)); Assert((y1-y0)*(y2-y3) < EP && ((y3-
    y0)*(y1-y2) < EP || yb*yb < ya*yc+EP));

    if (x == 0 && of == 0) {                                /* квадратичная Безье */
        sx = пол((3*x1-x0+1)/2); sy = пол((3*y1-y0+1)/2); /* новая средняя точка */
        returnplotQuadBezierSeg (x0,y0, sx,sy,x3,y3);
    }
    x1 = (x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)+1; x2 = (x2-x3)*(x2-                /* длинысторок */
    x3)+(y2-y3)*(y2-y3)+1;
    do { /* цикл по обоимконцам */
        аб = xa*yb-xb*ya; ac = xa*yc-xc*ya; bc = xb*yc-xc*yb; ex = ab*(ab+ac-3*bc)
        +ac*ac; /* P0 часть цикла самопересечения ? */
        e = бывший > 0? 1: кврт (1+1024/x1); /* вычисляемразрешение */
        аб *= e; переменныйток *= e; до н.э. *= e; бывший *= e*e; /* увеличиваемразрешение */
        xy = 9*(ab+ac+bc)/8; cb = 8*(xa-ya);/* инициализируемразности 1-й степени */
        dx = 27*(8*ab*(yb*yb-ya*yc)+ex*(ya+2*yb+yc))/64-ya*ya*(xy-ya);
        dy = 27*(8*ab*(xb*xb-xa*xc)-ex*(xa+2*xb+xc))/64-xa*xa*(xy+xa);
                                                            /* инициализируемразличия 2-й степени */
        xx = 3*(3*ab*(3*yb*yb-ya*ya-2*ya*yc)-ya*(3*ac*(ya+yb)+ya*cb))/4;
        yy = 3*(3*ab*(3*xb*xb-xa*xa-2*xa*xc)-xa*(3*ac*(xa+xb)+xa*cb))/4;
        xy = xa*ya*(6*ab+6*ac-3*bc+cb); ак = да*я; КБ = xa*xa;
        xy = 3*(xy+9*f*(cb*yb*yc-xb*xc*ac)-18*xb*yb*ab)/8;

        if (ex < 0) { /* отрицание значений, если внутри цикла самопересечения */
            дх = -дх; ды= -ды; xx = -xx; yy = -yy; xy = -xy; ак = -ac; КБ = -КБ;
                                                            /* инициализируемразличия 3-й степени */
        } АБ = 6*ya*ac; ак = -6*xa*ac; bc = 6*ya*cb; cb = -6*xa*cb;
        дх += xy; ex = dx+dy; ды+= xy;                      /* ошибка 1-го шага */

```

```
        for (pxy = &xy, fx = fy = f; x0 != x3 && y0 != y3; ) {
            setPixel(x0,y0); do { if                                    /* построить кривую*/
            (dx >                                                /* перемещаем подшаги на один пиксель */
                *pxy || dy < *pxy) перейти к выходу; y1 = 2*ex-dy; если          /* запутанные значения */
45              (2*ex >= dx) { fx--;                                /* сохраняем значение для проверки шага y */
                ex += dx += xx; dy += xy                                    /* x подэтап*/
                    += переменный ток; yy += до нашей эры; xx += АБ;
                }
                если (y1 <= 0) { fy--;                                      /* подэтапу */
50                  ex += dy += yy; dx += xy += до н.э.; xx += переменный ток; yy += К Б;
                }
            } while (fx > 0 && fy > 0); /* пиксель завершен? */
            если (2*fx <= f) { x0 += sx; x += e; } /* x шаг */
            если (2*my <= f) { y0 += sy; мой += e; } /* Шаг */
55          if (pxy == &xy && dx < 0 && dy > 0) pxy = &EP;/* допустимый пиксель вперед */
        }
выход: xx = x0; x0 = x3; x3 = xx; cx = -cx; xб = -xб;                       /* поменять ноги */
        yy = y0; y0 = y3; y3 = yy; си = -сы; yb = -yb; x1 = x2;
    } Пока (нога--);                                                /* попробуем другой конец */
60  plotLine(x0,y0, x3,y3);                  /* оставшаяся часть в случае каспа или кронода */
}
```

Листинг 15. Построение кубического сегмента Безье

Алгоритм пытается построить график с любого конца кривой и останавливается, если значения ошибок становятся равными.
смущенный. Это происходит при самопересечении и особенно при каспе. Кривая
затем закончился линией.

Можно было бы проверить условие разрыва пиксельного цикла только для конечной точки. Но если
конечная точка пропущена из-за небольшой ошибки округления, алгоритм не остановился.
реализация в листинге 14 гарантирует, что кривая не выйдет за пределы конечной точки.

Возможны некоторые дальнейшие оптимизации, например, написание отдельного цикла для кривых без цикла.
(самопересечение). Это уменьшит накладные расходы в пиксельном цикле для большинства нарисованных изображений.
Кривые Безье. Если применимо, вместо переменной-указателя можно использовать ссылку.

Алгоритм можно было бы упростить, если бы не требовалось создавать кубические кривые Безье с петлей или точкой возврата.
растеризованный. В этом случае вариант увеличения разрешения можно опустить.

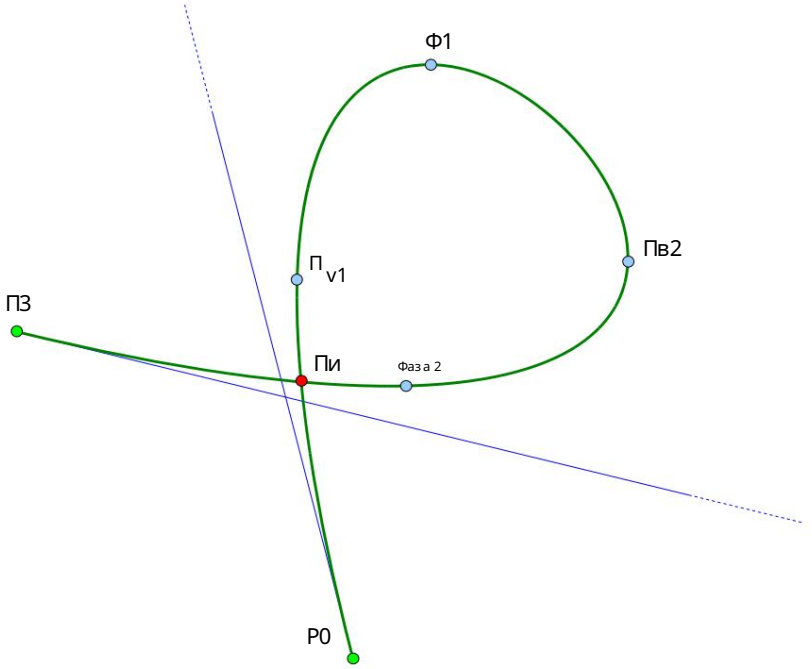


Рисунок 23: К убич еск ая к ривая Безь е, раз д еленная на сег мент ы

5.10 Д еление к убич еск ого Безь е

Сложнуюк убич еск уюк ривуюБезь е необход имо раз рез ат ь по г ориз онт аль ному и верт ик аль ному мак симуму и минимуму. Эт о дост иг ает ся с помощь юпервой парамет рич еск ой произ вод ной, к от орая опред еля ет к онт роль ные т оч к и Di = n(Pi+1–Pi).

Парамет рич еск ое з нач ение t эт их т оч ек вьн исля ет ся по к орня м первого парамет рич еск ого

производная : $\frac{dx}{t} = 3 \quad t\,d^{2}\,xa \quad 2 \quad t\,xb \quad xc \quad =0$ и $\frac{dy}{t} = 3 \quad t\,d^{2}\,of \quad 2 \quad t\,yb \quad yc \quad =0.$

К аж д ое к вад рат ное уравнение не имеет ни од ного, од ного или д вух д ейст вит ель ных к орней. В эт их т оч к ах Ph1, Ph2, Pv1 и Pv2 парамет ра t к ривуюнеобход имо раз бит ь по алг оритм у д е К аст ель жо .

К ривая Безь е перепарамет риз ует ся в новом д иапаз оне от t1 д о t2:

^t= ~t (t 2 t1)+t 2+t1 [t 1 ^t t2] [1 ~t 1].

Если необход имо пост роит ь от рез ок меж д у t1 и t2 , новые т оч к и к ривой Безь е сост авя т :

$x \quad 0= \quad t\,1^{3}xa \quad 3\,t \quad {}^{2}_{1}xb \quad 3\,t1\,xc \quad x\,d\,, \qquad x \quad 1= \quad t\,2 \quad t1^{2}xa \quad 2t\,1\,xb \quad xc \qquad t1 \qquad {}^{2}xb \quad 2t\,1\,xc \quad xd\,,$

$x \quad 2= \quad t\,1 \quad t\,2^{2}xa \quad 2\,t2\,xb \quad xc \qquad t\,2\,xb^{2} \quad 2t\,2\,xc \quad xd$ и $x \quad 3= \quad t\,2\,xa \quad 3\,t \qquad {}^{3} \qquad {}^{2}_{2}xb \quad 3\,t\,2\,xc \quad xd\,.$ (37)

Под ст ановк а нового парамет ра в к убич еск ое уравнение Безь е

$x= \quad t \quad {}^{3}x \quad a \quad 3 \quad t^{2} \quad xb \quad 3 \quad t\,x \quad c \quad x \quad d$

(плюс соот вет ст вующее уравнение д ля у) и сравнение услов ий д ает :

$x \quad a= \quad t2 \quad t1 \qquad 3x\,, \qquad x \quad b= \quad t\,2 \quad t\,1 \qquad {}^{2} \quad 2\,xb \qquad t1 \quad t2 \quad xa \quad ,$

$$x\_c = t2\ t1 \quad t1\ t2^{2}\ xa \quad 4\ t1\ t2 \quad xb \quad 4\ xc \quad и$$

$$x\_d = \quad t1\ t2^{3}\ xa \quad 6\ т\ 1\ т\ 2 \quad ^{2}xb \quad 12 \quad t1 \quad t2 \quad xc \quad 8\ x\ d. \tag{38}$$

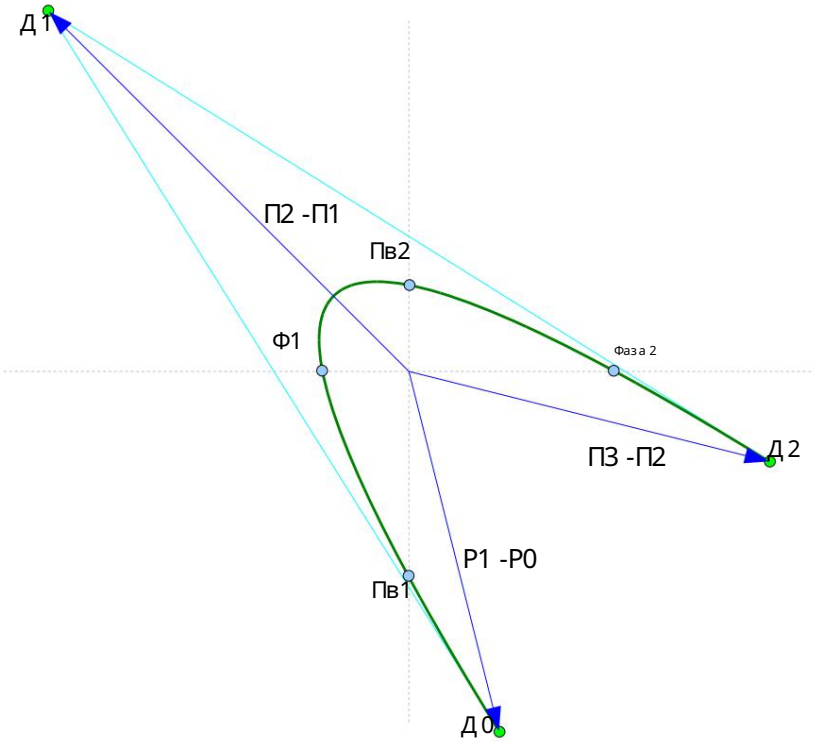


Рисунок 24. Горизонт аль ные и верт ик аль ные к орни к убич еск ого уравнения Безь е.

5.11 Рисование любой к убич еск ой к ривой Безь е

К ривая Безь е под раз деля ет ся на гориз онт аль ные и верт ик аль ные изменения градиент а. З ат ем эт и т оч к и сорт ируют ся для получ ения послед оват ель ных сегмент ов к ривой. Поск оль к у необход имо от сорт ироват ь не более ч ет ырех т оч ек , прост ейший алгорит м сорт ировк и дост ат оч но эффек т ивен. К ривая не под раз деля ет ся в т оч к е самопересеч ения , поск оль к у функ ц ия пост роения под сегмент а способна обработ ат ь эт от случ ай.

Друг ая проблема может воз ник нут ь , к ог да промежут оч ные т оч к и P1 или P2 ок ругля ют ся до ц елык ч исел из-за под раз деления . Может случ ит ь ся т ак , ч т о т щат ель но изб ег аемая ситуац ия изменения градиент а внез апно воз ник нет снова из-за ок ругления . Поэт ому, ч т обыизб ежат ь знач ит ель ных изменений, т оль к о границ а раз деленной к ривой масш т абируется до ц елык ч исел. По эт ой прич ине промежут оч ные т оч к и функ ц ии пост роения граф ик а под сегмент а должныимет ь т ипс плавающей зап я т ой.

Оч ень важно, ч т обывын исление парамет ра t имело двойнуют оч ност ь , особенно функ ц ии к вадрат ног о к орня . Нед ост ат оч ная т оч ност ь может немног о изменит ь парамет р раз деленной к ривой, т ак ч т о он боль ше не буд ет соот вет ст ват ь необход имым ог ранич ения м, и операт ор ут верж дения в функ ц ии под сегмент а з авершит ся неуд ач но.

```
voidplotCubicBezier (int x0, int y0, int x1, int y1,
                     int x2, int y2, int x3, int y3)
{                                                   /* построить любую кубическую кривую Безье */
    int n = 0, i = 0;
    длинный xc = x0+x1-x2-x3, xa = xc-4*(x1-x2);
    длинный xb = x0-x1-x2+x3, xd = xb+4*(x1+x2);
    длинный yc = y0+y1-y2-y3, ya = yc-4*(y1-y2);
    длинный yb = y0-y1-y2+y3, yd = yb+4*(y1+y2);
    float fx0 = x0, fx1, fx2, fx3, fy0 = y0, fy1, fy2, fy3;
    двойной t1 = xb*xb-xa*xc, t2, t[5];
                                    /* подразделение кривой при изменении знака градиента */
    if (xa == 0) { if                                /* горизонтально */
        (abs(xc) < 2*abs(xb)) t[n++] = xc/(2.0*xb);  /* одно изменение */
    } else if (t1 > 0.0) { t2 = sqrt(t1);            /* два изменения  */

        t1 = (xb-t2)/xa; если(fabs(t1) < 1,0) t[n++] = t1;
        t1 = (xb+t2)/xa; если(fabs(t1) < 1,0) t[n++] = t1;
    }
    t1 = yb*yb-ya*yc;
    if (ya == 0) { if                                /* вертикально */
        (abs(yc) < 2*abs(yb)) t[n++] = yc/(2.0*yb);  /* одно изменение */
    } else if (t1 > 0.0) { t2 = sqrt(t1);            /* два изменения  */

        t1 = (yb-t2)/ya; если(fabs(t1) < 1,0) t[n++] = t1;
        t1 = (yb+t2)/ya; если(fabs(t1) < 1,0) t[n++] = t1;
    }
    для (i = 1; i < n; i++)                          /* вид пузырька из 4 точек */
        если((t1 = t[i-1]) > t[i]) { t[i-1] = t[i]; т[i] = т1; i = 0; }

    t1 = -1,0; т[n] = 1,0; /* начало/конечная точка */
    for (i = 0; i <= n; i++) { /* рисуем каждый сегмент отдельно */
        т2 = т[i]; /* разделить на t[i-1], t[i] */
        fx1 = (t1*(t1*xb-2*xc)-t2*(t1*(t1*xa-2*xb)+xc)+xd)/8-fx0;
        fy1 = (t1*(t1*yb-2*yc)-t2*(t1*(t1*ya-2*yb)+yc)+yd)/8-fy0;
        fx2 = (t2*(t2*xb-2*xc)-t1*(t2*(t2*xa-2*xb)+xc)+xd)/8-fx0;
        fy2 = (t2*(t2*yb-2*yc)-t1*(t2*(t2*ya-2*yb)+yc)+yd)/8-fy0;
        fx0 -= fx3 = (t2*(t2*(3*xb-t2*xa)-3*xc)+xd)/8;
        fy0 -= fy3 = (t2*(t2*(3*yb-t2*ya)-3*yc)+yd)/8;
        x3 = пол(fx3+0,5); y3 = пол(fy3+0,5); /* масштабирование ограничено цельмч ислом */
        if (fx0 != 0.0) { fx1 *= fx0 = (x0-x3)/fx0; x2 *= x0; }
        если(fy0 != 0,0) { fy1 *= fy0 = (y0-y3)/fy0; фу2 *= фу0; }
        if (x0 != x3 || y0 != y3) /* сегмент t1 - t2 */
            plotCubicBezierSeg(x0,y0, x0+fx1,y0+fy1, x0+fx2,y0+fy2, x3,y3);
        x0 = x3; y0 = y3; x0 = x3; fy0 = fy3; т1 = т2;
    }
}
```

Листинг 16. Деление кубической кривой Безье

Сред ние к онт роль нье т оч к и P1 и P2 можно з аменит ь на ск воз нье т оч к и P˚ .1, Π_2 K

$$P1=\frac{5P0 \quad 18P^{\circ}1 \quad 9P^{\circ}2 \quad 2P3}{6} \quad P3 \text{ и } P2=\frac{2P0 \quad 9P^{\circ}1 \quad 18P^{\circ}2 \quad 5}{6}. \qquad (39)$$

6 Рациональный кубический Безье

Параметрическое уравнение рациональной кубической Безье делает [Марщ 2005, с. 175]:

$$P3\ B3(t)=\frac{(1-t)^3 w0\ P0+3(1-t)^2 t\ w1P1+3(1-t)t^2 w2\ P2+t^3 w3}{(1-t)^3 w0+3(1-t)^2 t\ w1+3(1-t)t^2 w2+t^3 w3} \quad (40)$$

Весанеявного уравнения рационального квадратичного Безье можно нормализовать, чтобыупростить вычисления без изменения кривой. Веса конечных точек P0 и P2 былираспределенымежду остальными. То же самое возможно для любого рационального Безье степени n.

Сравнивая неявное уравнение с нормализованными и ненормализованными весами, веса рационального Безье можно нормализовать заменой

$$w_i=\frac{Wi}{n\ w0\ n\ i_{\text{я хочу}}}. \quad (41)$$

Веса конечных точек P0 и Pn теперь стали w 0=w n=1.

Для кубического Безье нормированные веса составляют $w_1=3\frac{w1}{w0^2 w3}$ и $w_2=3\frac{w2}{w0w3^2}$.

Неявное уравнение рациональной кубической Безье необходимо для применения того же алгоритма, что и для предыдущих кривых. Это уравнение становится очень сложным.

С использованием результатов третья степень рациональной кубической кривой Безье неявно выражается так же, как уравнение (25) для нерациональной кривой:

$$\text{Си И1}_{\text{Икс}}^{\text{ж х}}\ d02\ d03\ d12\ d13\ d23=0\ \text{где}\ dij=wi\ wj(3i)(3j)\ xy\ 1\ j\ y\ j\ 1 \quad (42)$$

$$,\ y=\frac{д\,03\,д}{d0+d_{13}}$$

Не удалось найти никакого упрощения, которое могло быоблегчить вычисление начальных значений, как это было для нормальной кубической кривой Безье. Алгоритм с тем же принципом возможен, но слишком сложен для этой работы Вместо этого разрабатываются алгоритмыразделения кривой и рисования рациональных квадратичных сегментов.

6.1 Рациональное снижение степени

То же соображение, что и в главе 5.1 для нерациональных кубических кривых Безье, можно применить и для аппроксимации рациональных кривых Безье квадратичными.

Чтобыотклонение было небольшим, снова целесообразно провести подразделение в три этапа:

- кривая подразделяется в горизонтальных и вертикальных стационарных точках (глава 6.3)
- кривая подразделяется в точке перегиба (глава 6.4) • кривая подразделяется на два рациональных квадратичных Безье

Следующие соображения позволяют сделать отклонения небольшими. Кубическая кривая Безье P0-P1-P2-P3 на рисунке 18 главы5 делится ровно пополам (при t = ½) путем деления Де Кастельжу. Подразделение

Т оч к а Рс ост ает ся т оч но на к убич еск ой к ривой Без ь е. Промежут оч ные т оч к и на рисунк е 18 рац иональ ной к убич еск ой к ривой з ат ем вын ис ля ю т ся по формуле

$$wb=w2,\ Pb=\frac{3w1P1+ w0P0\ 3w2P2+ w3\ P3}{3w1+ w0\ 3\ w2+ w3}\ wa=w1,\ Pa=\frac{w0P0+ 3w1P1+}{3w2P2+ w3\ P3},$$

$$Pc=\frac{}{w0+ 3w1+ 3w2+ w3}.$$

Две под раз деленные рац иональ ные к ривые Без ь е можно нарисоват ь с помощь ю алгорит ма главы 4.

6.2 Деление рац иональ ного к убич еск ого Без ь е

К убич еск ую к ривую необход имо раз делит ь , ч т обы упрост ит ь ее рисование. Под раз деление осущест вля ет ся пут ем нах ож дения под ход я щего з нач ения парамет ра t (например, ст ац ионарных т оч ек).
Необход имо найт и положение и вес дв ух раз деленных к ривых. К ривая на рисунк е 25 под раз деля ет ся в т оч к е Рс на две к ривые P0-Pa-Pb-Pc и Pc-Pd-Pe-P3.

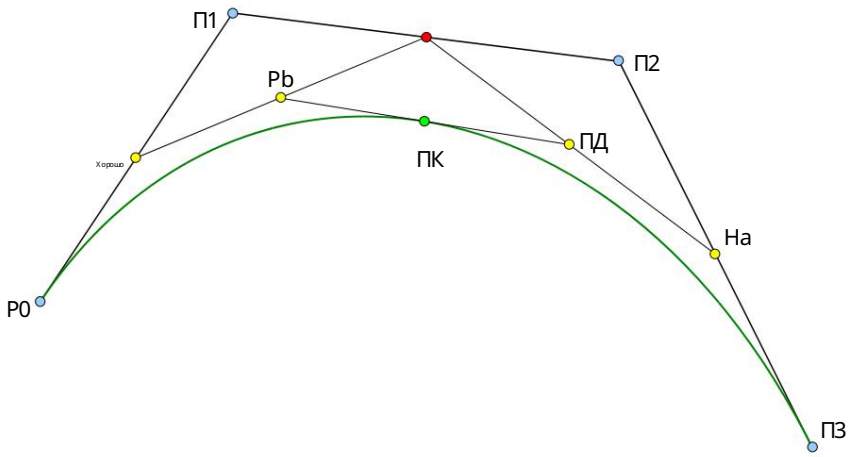


Рисунок 25. Деление рац иональ ного к убич еск ого Без ь е

Алгорит м де К аст ель жо расширен для рац иональ ного Без ь е. Дополнит ель ные т оч к и т еперь рассч ит ываю т ся по след ующим

уравнения м: $wa\ Pa=(1\ \ t)w0P0+ t\ w1P1$, $wa=(1\ \ t)w0+ t\ w1$,

$wb\ Pb=(1\ \ t)^2 w0P0+ 2\ (1\ \ t)t\ w1P1+ t^2 {}_{w2\ P2}$, $wb=(1\ \ t)^2 w0+ 2(1\ \ t)t\ w1+ t^2$ ш2 ,

$wc\ Pc=\ \ 1\ \ t\ \ ^3 w0\ P0\ \ \ 3\ \ 1\ \ t\ \ ^2 tw1P1\ \ \ 3\ \ 1\ \ t\ \ t^2 {}_{w2P2\ \ t}\ \ ^3 {}_{w3\ P3}$,

$wc=\ \ 1\ \ t\ \ ^3 w0\ \ \ 3\ \ 1\ \ t\ \ ^2 t\ w1\ \ \ 3\ \ 1\ \ t\ \ t^2 w2\ \ t^3$ шВ ,

$wd\ Pd=\ \ 1\ \ t\ \ ^2 w1P1\ \ \ 2\ \ \ 1\ \ t\ \ tw2P2\ \ t^2 w3\ P3$, $wd=\ \ 1\ \ t\ \ ^2 w1\ \ \ 2\ \ 1\ \ t\ \ t\ w2\ \ t^2$ шВ ,

мыРе= 1 t w2 P2 t w3 P3 , we= 1 t w2 t w3 .

Исполь з уя уравнение (41), можно сделат ь вес т оч к и Рс равным ед иниц е пут ем ад апт ац ии весов д руг их т оч ек .

6.3 Поиск корня

Рациональную кубу Безье необходимо разбить в стационарных точках. Эти точки являются максимумами и минимумами кривой в направлениях x и y. Эти точки можно вычислить, установив первый вывод параметрического уравнения (40) равным нулю. Это уравнение становится немного проще, если ввести замену Pij= wi wj Pi Pj.

$$P01 \quad 2P02 \quad P03 \quad 3P12 \quad 2P13 \quad P23 \quad t^4 \quad 2 \quad 2P01 \quad 3P02 \quad P03 \quad 3P12 \quad 3P13 \quad t^3$$
$$6P01 \quad 6P02 \quad P03 \quad 3P12 \quad t^2 \quad 2 \quad 2P01 \quad P02 \quad t \quad P01=0.$$

Это уравнение представляет собой многочлен четвертой степени. Аналитический расчет корней четвертой степени возможен, но затруднен. Следующее решение пытается найти максимально простой алгоритм.

Для этого приложения интерес представляют только настоящие корни. Если старший коэффициент следующих полиномиальных уравнений равен нулю, уравнение можно уменьшить на одну степень. Поэтому предполагается, что коэффициент не равен нулю.

Точность расчета требует особого внимания. Числа могут стать довольно большими из-за способностей. Ошибки округления возникают при добавлении или вычитании таких больших чисел из-за ограниченной точности расчета.

Реализации максимально простыи использующих минимум трансцендентных функций.

6.3.1 Квадратное уравнение

Для квадратного полиномиального уравнения $a2\,x^2 + a1\,x + a0 = 0$ замена

$p = \frac{a1}{a2\,2}$, $q = \frac{a0}{a2}$ облегчает расчет.

Количество действительных корней зависит от дискриминанта, показанного в следующей таблице:

Дискриминант	Количество действительных корней	Корневое значение(я)
$p^2 < q$	0	-
$p^2 = q$	1	x = p
$p^2 > q$	2	$x1 = p + \sqrt{p^2 - q}$ x2=q/ x1

Вычисление x2 делением более устойчиво, чем отрицательный корень.

6.3.2 Кубическое уравнение

Для кубического полиномиального уравнения $a3\,x^3 + a2\,x^2 + a1\,x + a0 = 0$ замена

$p = \frac{a2}{3\,a3\,2}$, $q = \frac{a1}{3\,a3}$, $p = \frac{3\,q - p^2}{p - 2\,\frac{a0}{a3}}$ снова упрощает расчет.

Эт и определения т ак же пок аз ываю т от ношения ст епеней межд у к оэффиц иент ами, ч т о важно, ч т обыизбежат ь пот ери ч исел с плавающей з апя т ой.

Замена x = z + p получ ает вд авленнуюк убик у z 3 3 q z+ 2r=0. Авт ор: Виет а

= y + q/y эт о уравнение можно превращ ит ь в к вадрат ное уравнение, з аменив y на z 3 :

и 6 + 2 p. 3 + q 3=0. Обрат ной з аменой получ аю т к орни к убич еск ого уравнения .

К олич ест во д ейст вит ель ных к орней з ависит от д иск риминант а, пок аз анного в след ующей т аблиц е:

Диск риминант	К олич ест во д ейст вит ель ных к орней	К орневое з нач ение(я)
r = 0 q = 0	1	x = p
p 2 > д 3	1	y= 3 r r 2 q 3 x= p y q/ y
2 3 p= q	2	x1= p+ r/ q x2= p 2 r/q
p 2 < д 3	3	y= q пот ому 1 Арк к ос 3 p q 3 x1= p 2 y x2,3=p y± 3 q y 2

В послед нем случ ае испол ьзует ся з амена z=2 q cosθ на д епрессивнуюк убик у, ч т обыизбежат ь выч ислений к омплек сных ч исел. Раз д елив уравнение на 2 q 3 , получ им к убич еск ое уравнение из 4 пот ому ч т о 3 θ 3cos θ+ p =0. Сравнение с уравнением cos3 =4cos q 3 д ает соот ношение 3 3cos cos3 = . З ат ем к орни выч исля ю т ся по з апис и м q 3 -r т аблиц у выше.

6.3.3 Уравнение ч ет верт ой ст епени

Для полиномиаль ного уравнения ч ет верт ой ст епени a4 x 4 a3 x 3 a2 x 2 a1 x a0=0 з амена a3 2 a2 , r= q p , q=3 p p= 4 a4 2a4 расч ет 2 p a1 , s= p 2 2 q p 4 r p a0 д елает проще. еще раз 4a4 a4

Замена x = z + p изб авля ет ся от к убич еск ого ч лена: z 4 2 qz 2 4 r z s=0 .

Если r равно нулю, эт о к вад рат ное уравнение от носит ель но z 2 Исход ная к вар т ик а з ат ем решает ся след ующим об раз ом:

$$x= p\pm 1q\pm 2q\ 2\quad c \qquad [p = 0]$$

Если r не равно нулю, уравнение ч ет верт ой ст епени можно превращ ит ь к решениюрез оль вент ного к убич еск ого уравнения неск оль к ими способами [Шмак ов, 2011]. Алг оритм Эйлера решает уравнение ч ет верт ой ст епени с помощь юрез оль вент ного к убич еск ого уравнения : 4 y 3 4q y 2 q 2 s y r 2=0

Это кубическое уравнение можно решить методом предыдущей главы. Оно всегда имеет положительный корень.

Если y является положительным корнем кубического уравнения, то корни исходной квартики составляют

$$x = p \pm 1 \quad \sqrt{q \quad y \pm 2\, r/\ \sqrt{y} \pm 2 \quad y} \qquad [p \quad 0]$$

Нижний индекс с знака плюс/минус указывает на один и тот же знак у четырех различных решений уравнения четвертой степени. Для этого приложения интерес представляют только настоящие корни.

6.4 Рациональная точка перегиба

Вычисление точки перегиба рационального кубического. Безье немного сложнее, чем нерационального. Уравнение (35) снова используется для определения кривизны кривой. Следующая замена упрощает вычисление:

$$c_{ijk} = w_i w_j w_k \quad {}^{\text{джидж}}_{\text{икс}} \quad {}_{y_k y_j} 1 \ . \quad {}_{y_i} 1 \ x_k$$

Точки перегиба рациональной кубической кривой Безье теперь можно было определить с помощью кубического уравнения

$$_{3t} \quad c_{012} \quad c_{013} \quad c_{023} \quad c_{123} \quad t^2 \quad 3\, c_{012} \quad 2\, c_{013} \quad c_{023} \quad t \quad 3\, c_{012} \quad c_{013} \quad c_{012} = 0. \qquad (43)$$

Корни этого уравнения можно найти, используя метод предыдущей главы

7 Сглаживание

Растеризация всегда вызывает эффект лестницыиз-за ограниченного разрешения изображения .
изображение. Сглаживание уменьшает этот эффект , устанавливая значения интенсивности пикселей.

Лучший алгоритм сглаживания использует расстояние пикселя до кривой в качестве информации для
интенсивность . Пиксель нулевой дистанции приобретает полнуюинтенсивность . Если центр пикселя равен одному пикселю
вне кривой пиксель не получает интенсивности.

Расстояние d до параметрической кривой P(t) = (x(t), y(t)) рассчитывается по кривой смещения

$$P\ t\ \pm d\ \frac{y'\ t\ ,\ x'\ t}{\sqrt{x'\ t^2 + y'\ t^2}}. \quad (44)$$

Это уравнение становится довольно сложным. Например, для квадратичной кривой Безье смещение
кривая имеет шестую степень , для рациональной квадратичной кривой Безье - восьмую степень . Слишком сложен
для простого алгоритма рисования .

Этот алгоритм можно расширить , чтобыиспользовать сглаживание, поскольку рассчитанные значения ошибок
представляют собой расстояние от кривой. Нулевые значения ошибок означают линию Расширив это
предел погрешности, так же можно рисовать более толстые линии или комбинации толстых сглаженных линий.

Требование к сглаживаниюсостоит в том, что пиксель должен быть не только освещен или нет , но и интенсивность
пикселя зависит от того, насколько он находится от кривой. Пиксель точно на кривой получит
наивысшая интенсивность , пикселя, находящиеся на расстоянии одной единицы(эквивалент размера пикселя), не светятся
совсем. Пикселя, находящиеся на расстоянии от нуля до одной единицыот кривой, получают интенсивность .
в зависимости от их расстояния . Это требование также делает интенсивность прямой диагонали
линия , эквивалентная вертикальной или горизонтальной линии.

В идеальном алгоритме сглаживания кривая представляет собой пересечение трехмерного ландшафта с холмами
и долинами. Кривая вдоль береговой линии всегда имеет наклон 45°.
градусов. Высота ландшафта определяет интенсивность пикселя .

Величина ошибки зависит от локального градиента кривой. Но, к счастью,дискретный
значение градиента уже рассчитано как разница ошибок в направления x X и Y. Так
максимальное значение ошибки ed для краев линии составляет

$$ed=\pm\ \sqrt{dx^2+dy^2}. \quad (45)$$

Дискретное значение является приближением градиента для кривых более высокой степени, чем единица, но
это небольшое приближение избавляет от сложного расчета градиента в таких случаях.
случаи.

7.1 Сглаженная линия

Значение ошибки нейронного уравнения определяет смешение цвета рисунка с цветом.
фоновый цвет .

В отличие от листинга 2 условие шага пикселя задерживается до тех пор, пока кривая не выйдет за пределы следующий пиксель, чтобы алгоритм всегда мог установить текущий и следующий пиксель по X или Y направление в соответствии со значением ошибки.

```
voidplotLineAA (int x0, int y0, int x1, int y1)
{ /* рисуем черную(0) сглаженную линию на белом (255) фоне */
    int dx = abs(x1-x0), sx = x0 <x1? 1:-1; int dy = abs(y1-y0), sy = y0 <
    y1? 1:-1; int x2, e2, err = dx-dy; /* значение ошибки e_xy */

    int ed = dx+dy == 0? 1 : sqrt((float)dx*dx+(float)dy*dy);

    for ( ;          ; ){                                          /* пиксельный цикл */
        setPixelAA(x0,y0,255*abs(err-dx+dy)/ed);
        e2 = ошибка; x2 = x0;
        if (2*e2 >= -dx) { if (x0 ==                               /* x шаг */
            x1) сломать;
            if (e2+dy <ed) setPixelAA(x0,y0+sy,255*(e2+dy)/ed);
            ошибаться -= dy; x0 += x;

        } if (2*e2 <= dy) { if (y0 ==                              /* шаг */
            y1) сломать;
            если (dx-e2 <ed) setPixelAA(x2+sx,y0,255*(dx-e2)/ed);
            ошибка += dx; y0 += cy;
        }
    }
}
```

Листинг 17: Программа для построения сглаженной линии

Условная установка цвета пикселя необходима, поскольку иногда пиксель слишком далеко от кривой (более одного пикселя) и его не нужно устанавливать.

Деления в цикле пикселей для расчета интенсивности цвета можно заменить операциями умножения и сдвига из соображений скорости. Коэффициент умножения должен расти с диапазоном значений. Необходимый размер слова для расчета ошибки равен количеству битов для интенсивность пикселя плюс биты для позиции, в большинстве случаев 24 бита.

Возможны некоторые дальнейшие оптимизации. Например, две разные петли для наклонов линий. ниже и выше условие if выйдет из цикла и, следовательно, ускорится алгоритм. Другая оптимизация могла бы заключаться в том, чтобы начать настройку пикселя с обоих концов. одновременно к середине, поскольку линия всегда симметрична.

Рисунок 26: Пример сглаженной линии.

Непосредственная настройка интенсивности пикселей работает только в том случае, если пиксели не перекрываются с другими рисунками. В противном случае настройку пикселей необходимо заменить функцией смешивания:

setPixelAA(x,y,(blend*getPixelAA(x,y)+(255-blend)*lineColor)/255).

Переменная blend — это значение интенсивности из предыдущего листинга. Значение 0 означает цвет фона, значение 255 — цвет линии. Также возможны другие функции смешивания, такие как минимум или максимум. Для цветных изображений эту функцию необходимо использовать для каждого цветового канала отдельно.

Алгоритм работает немного иначе, чем линейный алгоритм Сяолиня Ву [Xiaolin, 1991]. Это становится особенно очевидным для линий под углом 45 (или около 45) градусов. Сяолинь Ву использует долю приращения в качестве информации об интенсивности пикселя. Для линий под углом 45 градусов дроби не возникает, поэтому сглаживание не используется. Поэтому линии кажутся немного тоньше, чем в алгоритме листинга 16.

Рисунок 27. Слева — Сяолинь Ву, справа — алгоритм этой линии.

Хотя линии под углом 45 градусов всегда являются компромиссом для растровых дисплеев, линии Сяолиня Ву выглядят четче, а новый алгоритм немного мягче. Это во многом зависит от устройства и приложения.

Также можно сделать строки листинга 16 более четкими, уменьшив максимальное расстояние ошибки. Передача значений интенсивности от одного пикселя к другому происходит быстрее и использует меньше «серых» значений.

Другой вариант — изменить алгоритм, чтобы он принимал значения с плавающей запятой в качестве значений координат. Тогда линия выглядит так, будто начинается и заканчивается в позициях между целочисленными пикселями.

7.2 Сглаженный круг

Программа для сглаженного круга примерно такая же, как и программа для первого круга в листинг 5, за исключением того, что каждый пиксель становится серым. Кроме того, x-шаг устанавливает внешний пиксель , а шаг по оси Y устанавливает внутренний пиксель круга.

```
voidplotCircleAA (int xm, int ym, int r)
{                          /* рисуем черный сглаженный круг на белом фоне */
    интервал x = r, y = 0; int          /* II. квадрант слева направо вверх */
    i, x2, e2, err = 2-2*r; r = 1-ошибка;                                    /* ошибка 1.шага */
5
    для ( ; ; ) {
        я = 255*abs(err+2*(x+y)-2)/r; setPixelAA                  /* получаем значение смешивания пикселя */
        (xm+x, ym-y, я ); setPixelAA(xm+y,                                   /* I. Квадрант */
        ym+x, я ); setPixelAA (xm-x, ym+y, я );                              /* II. Квадрант */
10      setPixelAA (xm-y, ym-x, я ); если (x ==                              /* III. Квадрант */
        0) сломать ;                                                         /* IV. Квадрант */

        e2 = ошибка; x2 = x; если                                           /* запоминаем значения */
        (ошибка > y) {я =                                                    /* x шаг */
15          255*(ошибка+2*x-1)/r; если (я                                    /* внешний пиксель */
            < 255) {
                setPixelAA(xm+x, ym-y+1, i);
                setPixelAA(xm+y-1, ym+x, я );
                setPixelAA(xm-x, ym+y-1, i);
20              setPixelAA (xm-y+1, ym-x, я );

            } Ошибка -= --x*2-1;

        } if (e2 <= x2--) { i =                                              /* шаг */
25          255*(1-2*y-e2)/r; если (я < 255)                                 /* внутренний пиксель */
            {
                setPixelAA(xm+x2, ym-y, i);
                setPixelAA(xm+y, ym+x2, я );
                setPixelAA(xm-x2, ym+y, я );
30              setPixelAA (xm-y, ym-x2, i);
            }
            ошибка -= --y*2-1;
        }
    }
35 }
```

Листинг 18: Программа для построения сглаженного круга

На этот раз у алгоритма нет проблем с ложными пикселями.

Для малых радиусов значения интенсивности имеют небольшое отклонение от теоретического значения , поскольку условия изменения производной.

Рисунок 28: Пример сглаженного круга.

7.3 Сглаженный эллипс

Алгоритм сглаженной линии имеет то преимущество, что расчет максимальной ошибки нужно было сделать только один раз вне пикселного цикла. Для всех остальных кривых это значение меняется с каждым пикселем, и каждый раз приходилось рассчитывать заново. Вычисление квадратного корня из уравнение (45) внутри пикселного цикла требует слишком больших затрат времени.

По этой причине для расчета максимальной ошибки используется приближение:

$$и = d\,x + \frac{2д\,x д_и^2}{4д_{ик}^2 + д_и^2} \quad \text{для } [dy \quad dx] \tag{46}$$

Погрешность этого приближения составляет менее 2% (ровно $\frac{2}{\sqrt{2}} - 1{,}4$) и достаточно точна для интенсивность пикселей.

Иснова алгоритм не работает в конце эллипса. Ошибка расчета одна пиксель вперед, где кривая уже меняет направление. В этом случае так же будет установлен пиксель за пределами направления Y из-за сглаживания. Это не имеет значения, поскольку у интенсивность слишком мизкого, чтобы быть видимым. Но из-за аппроксимации градиента уравнением (46) это решение не работает должным образом для маленьких эллипсов. Поэтому вставляется условие прерывания. перед установкой сглаженного пикселя и завершением эллипса дополнительным циклом.

Приращение позиции также устанавливает сглаженный пиксель. Поэтому значения одного направления по-прежнему необходимы в неизмененном виде при обновлении другого направления. Приращение поэтому разделен на две части. Первый устанавливает пиксель, а второй обновляет ошибку. значения для расчета пикселей. Другой (может быть, более быстрый) вариант — использовать временный ценности.

```
voidplotEllipseRectAA (int x0, int y0, int x1, int y1)
{/* нарисуйте черный сглаженный прямоугольный эллипс на белом фоне */
    long a = abs(x1-x0), b = abs(y1-y0), b1 = b&1; float dx = 4*(a-1.0)*b*b,          /* диаметр */
    dy = 4*(b1+1)*a*a; float ed, i, err = b1*a*a-dx+dy; логический f;          /* приращение ошибки */
                                                                                  /* ошибка 1.шага */


    if (a == 0 || b == 0) returnplotLine (x0,y0, x1,y1);
    если (x0 > x1) { x0 = x1; x1 += a; }/* если вызывается с переставленными точками */
    если (y0 > y1) y0 = y1; /* .. обменивается их */
    y0 += (б+1)/2; y1 = y0-b1; /* начальный пиксель */
    a = 8*a*a; б1 = 8*б*б;

    for (;;) { i =                                    /* приблизительное значение ed=sqrt(dx*dx+dy*dy) */
        min(dx,dy); ed = Макс (dx, dy);
        if (y0 == y1+1 && err > dy && a > b1) ed = 255*4./a; /* х-наконечник */
        иначе ed = 255/(ed+2*ed*i*i/(4*ed*ed+i*i)); /* аппроксимация */
        я = ed*fabs(err+dx-dy); /* получаем значение интенсивности по ошибке пикселя */
        setPixelAA (x0, y0, я ); setPixelAA (x0, y1, я );
        setPixelAA (x1, y0, я ); setPixelAA (x1, y1, я );

        if (f = 2*err+dy >= 0) { if (x0 >= x1)          /* шаг х, запомнить условие */
            сломать;
            я = ed*(ошибка+dx);
            если (я < 255) {
                setPixelAA(x0,y0+1, я ); setPixelAA (x0, y1-1, я );
                setPixelAA (x1, y0+1, я ); setPixelAA (x1, y1-1, я );
            } /* увеличиваем ошибку позже, так как значения все еще нужны*/

        } if (2*err <= dx) { i = ed*(dy-                                  /* шаг */
            err);
            если (я < 255) {
                setPixelAA(x0+1,y0, я ); setPixelAA (x1-1, y0, я );
                setPixelAA(x0+1,y1, я ); setPixelAA (x1-1, y1, я );
            }
            y0++; y1--; ошибка += dy += a;

        } если (e) { x0++; x1--; ошибка -= dx -= b1; }          /* приращение ошибки х */

    } if (--x0 == x1++) while                     /* слишком ранняя остановка плоских эллипсов */
        (y0-y1 < b) {
            я = 255*4*fabs(err+dx)/b1; /* -> завершить кончик эллипса */
            setPixelAA(x0,++y0, я ); setPixelAA (x1, y0, я );
            setPixelAA(x0,--y1, я ); setPixelAA (x1, y1, я );
            ошибка += dy += a;
        }
}
```

Рисунок 19: Программа для построения сглаженного прямоугольного эллипса.

Максимальное значение ошибки вершиныплоских эллипсов определить сложно.

Рис. 29. Пример сглаженного эллипса.

Повернутый эллипс можно нарисовать, используя сглаженнуюрациональную квадратичную кривуюБезье
глава 7.6.

7.4 Сглаженная квадратичная кривая Безье

Основная процедура построения сглаженной квадратичной кривой Безье остается прежней.
как в главе 3.7, листинг 10. Только подпрограмма для сегмента Безье немного меняется на
установите интенсивность пикселей кривой в соответствии со значением ошибки.

Рисунок 30. Пример сглаженной квадратичной кривой Безье.

Расчет максимального расстояния ошибки также зависит от позиции. Имеет значение,
используются ли значения текущего или следующего пикселя. Для точного расчета
дополнительно требуется производная кривой. Использование значений разницы—это возможность
получить более быстрый и простой алгоритм.

Еще одна небольшая ошибка возникает, если алгоритм дает сбой из-за того, что другая часть
кривой приближается. В этом случае кривая завершается сглаженной линией, но переход от кривой
к линии не может быть точным. Кроме того, значения сглаженных пикселей линии не совсем
соответствуют значениям, полученным в результате расчета кривой Безье.

```
voidplotQuadBezierSegAA (int x0, int y0, int x1, int y1, int x2, int y2)
{ /* рисуем ограниченный сглаженный квадратичный сегмент Безье */
    int sx = x2-x1, sy = y2-y1;
    длинный xx = x0-x1, yy = y0-y1, xy; двойной dx,                    /* относительные значения для проверок */
5   dy, err, ed, cur = xx*sy-yy*sx;                                    /* кривизна */

    Assert(xx*sx <= 0 && yy*sy <= 0); /* знак градиента не должен меняться */

    if (sx*(long)sx+sy*(long)sy > xx*xx+yy*yy) { /* начинаем с более длинной части */
10      x2 = x0; x0 = x+x1; y2 = y0; y0 = си+y1; кур = -cur; /* поменять местами P0 P2 */

    } если (cur != 0)
    {                                                                  /* нет прямой линии */
        xx += x; xx *= sx = x0 < x2 ? 1:-1; yy += сы; yy *= sy = y0 <  /* направление шага x */
15      y2 ? 1:-1; xy = 2*xx*yy; xx *= xx; yy *= yy; если (cur*sx*sy   /* направление шага по оси */
        < 0) {                                                         /* различия 2-й степени */
                                                                       /* отрицательная кривизна? */
            xx = -xx; yy = -yy; xy = -xy; кур = -cur;
        }
20      dx = 4,0*sy*(x1-x0)*cur+xx-xy; dy = 4,0*sx*(y0-                /* различия 1-й степени */
        y1)*cur+yy-xy;
        xx += xx; yy += yy; ошибка = dx+dy+xy; do {cur =               /* ошибка 1-го шага */

        min(dx+xy,-xy-dy);
25          ed = max(dx+xy,-xy-dy); ed = 255/                       /* приблизительное расстояние ошибки */
            (ed+2*ed*cur*cur/(4*ed*ed+cur*cur)); setPixelAA(x0,y0, ed*fabs(err-
            dx-dy-xy)); if (x0 == x2 && y0 == y2) return;/* последний  /* построить кривую*/
            пиксель -> кривая закончена */
            x1 = x0; Cur = dx-ошибка; y1 = 2*err+dy < 0;
30          если (2*ошибка+dx > 0) {                                   /* x шаг */
                if (err-dy <ed) setPixelAA(x 0, y0 + sy, ed * fabs (err-dy));
                x0 += x; дх -= xy; ошибка += dy += yy;
            }
            если (y1) {                                                /* шаг */
35              if (cur <ed) setPixelAA(x1+sx,y0, ed*fabs(cur));
                y0 += cy; ды-= xy; ошибка += dx += xx;
            }
        } Пока (dy <dx);                                   /* отрицание градиента -> закрытие кривых */
    }
40  plotLineAA(x0,y0, x2,y2);                              /* отображаем оставшуюся иглу до конца */
}
```

Листинг 20: Программа для построения сглаженной квадратичной кривой Безье

Также можно было бы изменить алгоритм, чтобы он принимал кривые с аргументами с плавающей запятой в качестве координат для этой подпрограммы. Это приведет к более плавным кривым.

7.5 Сглаженная рациональная квадратичная кривая Безье

Алгоритм сглаженной рациональной квадратичной кривой Безье остается в основном таким же, как и для алгоритма сглаженной или нерациональной кривой, поскольку он сталкивается с теми же проблемами. Кривую необходимо разделить, если вес слишком мал, а другая часть кривой подходит слишком близко. Также кривая должна быть завершена с помощью юлинейного алгоритма для конечников квадратичных кривых Безье. Сглаживание обрабатывается путем установки пикселя в соответствии со значением относительной ошибки, если увеличивается направление х или у.

Использование значения ошибки в качестве измерения расстояния пикселя до кривой является приблизительным. Как было продемонстрировано ранее, алгоритм рисования дает сбой, если другая часть кривой подходит слишком близко. Эта другая часть кривой также существенно влияет на значения ошибок, используемые для сглаживания. Поэтому, прежде чем алгоритм начнет давать сбой, сглаживание несколько раз начинает выглядеть некорректно.

Возможным решением этой проблемы является использование растра высокого разрешения из главы 3.4 для определения направления приращения.

Рис. 31. Пример сглаженного повернутого эллипса.

На рис. 30 показан сглаженный повернутый эллипс, полученный из четырех сглаженных рациональных кривых Безье. Эллипс разделен программой из листинга 13 на рациональные квадратичные сегменты Безье, которые рисуются с помощью следующей программы для рациональных сглаженных квадратичных сегментов Безье.

При расчете максимального расстояния ошибки снова используется аппроксимация уравнения 46, чтобы избежать дорогостоящей функции квадратного корня в пиксельном цикле.

```
voidplotQuadRationalBezierSegAA (int x0, int y0, int x1, int y1,
                                 int x2, int y2, float w)
{       /* рисуем сглаженный рациональный квадратичный сегмент Безье, квадрат веса */
  int sx = x2-x1, sy = y2-y1; /* относительные значения для проверок */
  двойной dx = x0-x2, dy = y0-y2, xx = x0-x1, yy = y0-y1; двойной xy = xx*sy+yy*sx,
  cur = xx*sy-yy*sx, err, ed; логический f;                          /* кривизна */


  Assert(xx*sx <= 0,0 && yy*sy <= 0,0); /* знак градиента не должен меняться */

  if (cur != 0,0 && w > 0,0) {                                       /* нет прямой линии */
    if (sx*(long)sx+sy*(long)sy > xx*xx+yy*yy) { /* начинаем с более длинной части */
      x2 = x0; x0 -= dx; y2 = y0; y0 -= dy; кур = -cur;              /* поменять местами P0 P2 */

    } xx = 2,0*(4,0*w*sx*xx+dx*dx); yy =                             /* различия 2-й степени */
    2,0*(4,0*w*sy*yy+dy*dy);
    sx = x0 <x2? 1:-1; си = y0 < y2?                                 /* направление шага x */
    1:-1; xy =                                                       /* направление шага по оси */
    -2,0*sx*sy*(2,0*w*xy+dx*dy);

    если (cur*sx*sy < 0) {                                           /* отрицательная кривизна? */
      xx = -xx; yy = -yy; кур = -cur; xy = -xy;
    }
    dx = 4,0*w*(x1-x0)*sy*cur+xx/2,0+xy; dy = 4,0*w*(y0-             /* различия 1-й степени */
    y1)*sx*cur+yy/2,0+xy;

    if (w <0,5 && dy > dx) {                                         /* плоский эллипс, алгоритм не работает */
      Cur = (w+1,0)/2,0; ш= sqrt (ш); xy = 1,0/(w+1,0);
      sx = пол((x0+2,0*w*x1+x2)*xy/2,0+0,5); /* разделить кривуюпополам */
      sy = Floor((y0+2.0*w*y1+y2)*xy/2.0+0.5); dx =
      пол((w*x1+x0)*xy+0,5); dy = пол((y1*w+y0)*xy+0,5);
      сюжет QuadRationalBezierSegAA (x0, y0, dx, dy, sx, sy, cur); /* разделить график */
      dx = пол((w*x1+x2)*xy+0,5); dy = пол((y1*w+y2)*xy+0,5);
      returnplotQuadRationalBezierSegAA (sx,sy, dx,dy,x2,y2,cur);
    }
    ошибка = dx+dy-xy;                                               /* ошибка 1-го шага */
    do                                                               /* пиксельный цикл */
      {cur = min(dx-xy,xy-dy); ed = max(dx-xy,xy-dy);
      ed += 2*ed*cur*cur/(4.*ed*ed+cur*cur); /* приблизительное расстояние ошибки */
      x1 = 255*abs(err-dx-dy+xy)/ed; /* получаем значение смешивания по ошибке пикселя */
      если (x1 <256) setPixelAA(x0,y0, x1); /* построить кривую*/
      if (f = 2*err+dy < 0) { /* у шаг */
        если (y0 == y2) return; /* последний пиксель -> кривая закончена */
        if (dx-err <ed) setPixelAA(x0+sx,y0, 255*fabs(dx-err)/ed);
      }
      if (2*err+dx > 0) { /* x шаг */
        если (x0 == x2) возврат ; /* последний пиксель -> кривая закончена */
        if (err-dy <ed) setPixelAA (x 0, y0 + sy, 255 * fabs (err-dy)/ed);
        x0 += x; дх += xy; ошибка += dy += yy;
      }
      если (f) { y0 += sy; ды+= xy; ошибка += dx += xx; }            /* шаг */
    } Пока (dy <dx);                                                 /* градиент отрицает -> алгоритм не работает */
  }
  plotLineAA(x0,y0, x2,y2);                                          /* отображаем оставшуюся иглу до конца */
}
```

Листинг 21. Программа для построения сглаженной рациональной квадратичной кривой Безье

7.6 Сглаженная кубическая кривая Безье

Алгоритм сглаженной кубической кривой Безье сталкивается с большими проблемами, чем обычный кубический. Причина та же, что и раньше: точки на трехмерной поверхности, где дифференцирование достигает нуля или бесконечности. В этих точках невозможно вычислить расстояние в один пиксель значений разностных ошибок.

В альтернативной версии алгоритма рисования было достаточно сохранить направление рисования в соответствии с кривой. Сейчас этого недостаточно. Информация об интенсивности пикселя необходима дополнительно.

Алгоритм листинга 14 обнаруживает близкое пересечение кривой по изменению знаков разности. Поскольку условие приращения алгоритма сглаживания отличается, обнаружение ближнего пересечения также меняется. Он проверяет изменение знаков разницы на два пикселя вперед только в том случае, если кривая вообще содержит цикл самопересечения, чтобы избегать перерывов в случае сложных условий старта.

Еще одна сложная проблема — растр высокого разрешения, который используется внутри цикла самопересечения. Алгоритму нужны знаки ошибок пикселей в качестве информации об интенсивности, но расчет выполняется с более высоким разрешением. Поэтому значение ошибки необходимо рассчитывать в зависимости от текущего положения субпикселя, не забывая, что эти значения также меняются с каждым субпикселем.

Рисунок 32. Пример сглаженной кубической кривой Безье.

На рисунке 31 показана сглаженная кубическая кривая Безье. Кажется, что сглаживание в верхнем левом углу неправильное. Но острый край кривой исходит из почти особой точки.

По сути, алгоритм предполагает, что различия не меняются вблизи заданного пикселя. Это правильно в большинстве случаев. Но в особых или почти особых точках, таких как точки возврата, это неверно. Значения разницы в этих областях сильно меняются. В таких случаях максимальное значение ошибки в качестве эталонного расстояния в пикселях меняется слишком быстро с каждым пикселем.

Поэтому значение интенсивности для ближайшего пикселя ни в коем случае не является лучшим решением для этой проблемы простой алгоритм.

```
voidplotCubicBezierSegAA (int x0, int y0, float x1, float y1,
                          float x2, float y2, int x3, int y3) /* построение
{                                                             ограниченного сглаженного кубического сегмента Безье */
    int f, fx, fy, leg = 1;
    int sx = x0 <x3? 1:-1, sy = y0 < y3? 1:-1; /* направление шага */
    float xc = -fabs(x0+x1-x2-x3), xa = xc-4*sx*(x1-x2), xb = sx*(x0-x1-x2+x3);
    float yc = -fabs(y0+y1-y2-y3), ya = yc-4*sy*(y1-y2), yb = sy*(y0-y1-y2+y3);
    двойной ab, ac, bc, ba, xx, xy, yy, dx, dy, ex, px, py, ed, ip, EP = 0,01;

                                                                  /* проверка ограничений кривой */
    /* наклон P0-P1 == P2-P3 и (P0-P3 == P1-P2 или без изменения наклона) */
    Assert((x1-x0)*(x2-x3) < EP && ((x3-x0)*(x1-x2) < EP || xb*xb < xa*xc+EP)); Assert((y1-y0)*(y2-y3) < EP && ((y3-
    y0)*(y1-y2) < EP || yb*yb < ya*yc+EP));

    if (x == 0 && of == 0) {                                      /* квадратичная Безье */
        sx = пол((3*x1-x0+1)/2); sy = пол((3*y1-y0+1)/2); /* новая средняя точка */
        returnplotQuadBezierSegAA (x0,y0, sx,sy,x3,y3);
    }
    x1 = (x1-x0)*(x1-x0)+(y1-y0)*(y1-y0)+1; x2 = (x2-x3)*(x2-          /* длиныстрок */
    x3)+(y2-y3)*(y2-y3)+1;
    do { /* цикл по обоимконцам */
        аб= xa*yb-xb*ya; ac = xa*yc-xc*ya; bc = xb*yc-xc*yb; ip = 4*ab*bc-ac*ac; ex =
        ab*(ab+ac-3*bc)+ac*ac; /*                                 /* цикл самопересечения вообще? */
        P0 часть цикла самопересечения ? */
        e = бывший > 0? 1: кврт (1+1024/x1); /* вычисляем разрешение */
        аб*= e; переменный ток *= e; до н.э. *= e; бывший *= e*e; /* увеличиваем разрешение */
        xy = 9*(ab+ac+bc)/8; ba = 8*(xa-ya);/* начальные разности 1-й степени*/
        dx = 27*(8*ab*(yb*yb-ya*yc)+ex*(ya+2*yb+yc))/64-ya*ya*(xy-ya);
        dy = 27*(8*ab*(xb*xb-xa*xc)-ex*(xa+2*xb+xc))/64-xa*xa*(xy+xa);
                                                                  /* инициализируем различия 2-й степени */
        xx = 3*(3*ab*(3*yb*yb-ya*ya-2*ya*yc)-ya*(3*ac*(ya+yb)+ya*ba))/4;
        yy = 3*(3*ab*(3*xb*xb-xa*xa-2*xa*xc)-xa*(3*ac*(xa+xb)+xa*ba))/4;
        xy = x*of*(6*ab+6*ac-3*bc+ba); ак = да*я ; ба = xa*xa;
        xy = 3*(xy+9*f*(ba*yb*yc-xb*xc*ac)-18*xb*yb*ab)/8;

        if (ex < 0) { /* отрицание значений, если внутри цикла самопересечения */
            дх = -дх; ды= -ды; xx = -xx; yy = -yy; xy = -xy; ак = -ac; ба = -ба;
                                                                  /* инициализируем различия 3-й степени */
        } АБ = 6*y*ac; ак = -6*x*ac; bc = 6*of*ba; ба = -6*xa*ба;
        дх += xy; ex = dx+dy; ды+= xy;                                /* ошибка 1-го шага */

        for (fx = fy = f; x0 != x3 && y0 != y3; ) {
            y1 = min(xy-dx,dy-xy);
            ed = max(xy-dx,dy-xy); ed =                           /* приблизительное расстояние ошибки */
            f*(ed+2*ed*y1*y1/(4*ed*ed+y1*y1)); y1 = 255*abs(ex-(f-
            fx+1)*dx-(f-fy+1)*dy+f*xy)/ed;
```

```
                если (y1 <256) setPixelAA(x0, y0, y1); px = fabs(ex-(f-                    /* построить кривую*/
                fx+1)*dx+(fy-1)*dy); py = fabs(ex+(fx-1)*dx-(f-fy+1)*dy);   /* интенсивность пикселей x перемещение */
                y2 = y0;                                                    /* интенсивность пикселей по оси Y */

50              do { /* перемещаем подшаги на один пиксель */
                    if (ip >= -EP) /* пересечение возможно? -> проверьте.. */
                        если (dx+xx > xy || dy+yy < xy) перейти к выходу; /* два шага x или y */
                    y1 = 2*ex+dx; if                                /* сохраняем значение для проверки шага y */
                    (2*ex+dy > 0) { /* x подэтап */
55                      FX--; ex += dx += xx; dy += xy += переменный ток ; yy += до нашей эры; xx += АБ; /
                    } Иначе, если (y1 > 0) перейти к выходу;                    * крошечный, почти острие */
                    если (y1 <= 0) { fy--;                                          /* подэтапу */
                        ex += dy += yy; дх += xy += до н.э.; xx += переменный ток ; yy += ба;
                    }
60              } while (fx > 0 && fy > 0); /* пиксель завершен? */
                if (2*fy <= f) { /* x+ сглаживание пикселя */
                    if (py <ed) setPixelAA(x0+sx, y0, 255*py/ed); /* построить кривую*/
                    y0 += cy; мой += e; /* Шаг */

65              } if (2*fx <= f) {                                  /* y+ пиксель сглаживания */
                    if (px <ed) setPixelAA(x0, y2+sy, 255*px/ed); /* построить кривую*/
                    x0 += x; x += e; /* x шаг */
                }

70          } перерыв;                                              /* завершаем кривуюпо линии */
Выход:
            if (2*ex < dy && 2*fy <= f+2) {                     /* округленный пиксель аппроксимации x+ */
                if (футы<ed) setPixelAA(x0 + sx, y0, 255 * футов/ed); y0 += мать ;          /* построить кривую*/

75          }
            if (2*ex > dx && 2*fx <= f+2) {                     /* округленный пиксель аппроксимации y+ */
                if (px <ed) setPixelAA(x0, y2+sy, 255*px/ed); /* построить кривую*/
                x0 += x;

80          } xx = x0; x0 = x3; x3 = xx; cx = -cx; xб = -xб; yy = y0; y0 = y3; y3 =          /* поменять ноги */
            yy; си = -сы; yb = -yb; x1 = x2;
        } Пока (нога--); /* попробуем другой конец */
        plotLineAA(x0,y0, x3,y3); /* оставшаяся часть в случае аспа или крыша */
}
```

Листинг 22: Программа для построения сглаженной кубической кривой Безье

Алгоритм в листинге 21 не идеален. На пересечениях кривых переход от кривая аппроксимации линии остается слегка видимой.

В случае самопересечения алгоритм может находиться в середине сглаживания. Сейчас расчет направления шага завершается неудачно, и рисование продолжается со сглаживанием линия. Это тоже нарушает расчет интенсивности пикселя.

Рисунок 33. Проблемное сглаживание кубических кривых Безье.

На рисунке 32 показано несколько проблемных примеров кубических кривых Безье. Если выпосмотрите внимательно, то иногда можно заметить какой-то странный пиксель сглаживания на кривой. Это всегда находится рядом с точкой самопересечения, где стратегия алгоритма должна изменить ся с кубической кривой на прямую линию.

Чтобыпреодолеть этинедостатки, цикл самопересечения нуждается в более совершенном (и сложном) алгоритме.

Все предыдущие алгоритмысглаживания можно было расширить, чтобыпринимать аргументыс плавающей запятой в качестве координат. Таким образом, можно начинать или заканчивать кривые в точках «между» пикселями.

8 толстых сглаженных кривых

Нарисовать толстую сглаженную кривую можно разными способами. Одна из возможностей — сначала установите пиксель полной интенсивности, а затем добавьте пиксель сглаженной интенсивности. Или нарисуйте левую и правую сторону кривой и заполните пространство между ними. Смещение к кривая рассчитывается по уравнению 44 и становится довольно сложной для Безье из-за деление на квадратный корень .

Алгоритм этого документа также можно использовать для рисования толстых кривых. Информация о значение ошибки используется для расчета расстояния от кривой и установки интенсивности пикселей в соответствии с этим значением. В отличие от комплексного уравнения кривой смещения это расчет очень простой и быстрый.

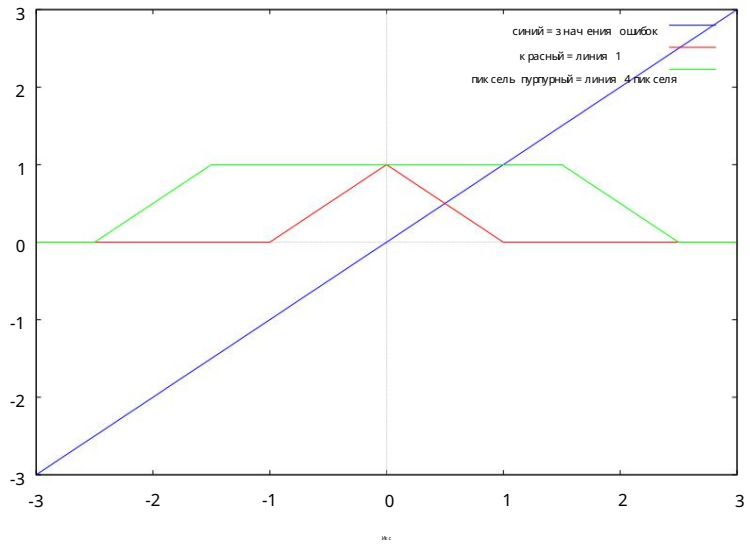


Рисунок 34: Толщина линии в зависимости от значений погрешностей

На рисунке 33 показано, как значение ошибки используется для определения значения интенсивности пикселя . рисуйте линии разной ширины. Горизонтальная ось отображает относительное значение ошибки и вертикальная ось — интенсивность пикселей.

8.1 Толстые линии

Простейшей реализацией приведенного выше утверждения являются линии определенной толщины. Быстрая реализация заключалась в одновременном рисовании левой и правой линий и заполнении пикселей. между.

Следующий листинг 22 демонстрирует метод, используя значение ошибки в качестве расстояния . измерение. В дополнение к версии сглаживания при каждом увеличении шага необходимо устанавливать не только один пиксель , но и несколько пунктов.

В случае прямой горизонтальной или вертикальной линии алгоритм должен быть осторожен, чтобы не нарисовать за пределами конечного пикселя , когда толщина линии задает пикели.

```
voidplotLineWidth (int x0, int y0, int x1, int y1, float th)
{                                                  /* рисуем сглаженнуюлиниют олщиной th */
    int dx = abs(x1-x0), sx = x0 <x1? 1:-1; int dy = abs(y1-y0), sy = y0 <
    y1? 1:-1; int err = dx-dy, e2, x2, y2; /* значение ошибки e_xy */
5
    float ed = dx+dy == 0? 1 : sqrt((float)dx*dx+(float)dy*dy);

    для (th = (th+1)/2; ; ) {                                                  /* пиксельный цикл */
        setPixelAA(x0, y0, max(0,255*(abs(err-dx+dy)/ed-th+1)));
10      e2 = ошибка; x2 = x0;
        если (2*e2 >= -dx) {                                                   /* x шаг */
            for (e2 += dy, y2 = y0; e2 < ed*th && (y1 != y2 || dx > dy); e2 += dx)
                setPixelAA(x0, y2 += sy, max(0,255*(abs(e2)/ed-th+1)));
            если (x0 == x1) сломать;
15          e2 = ошибка; ошибать ся -= dy; x0 += x;

        } if (2*e2 <= dy) { for (e2                                            /* шаг */
            = dx-e2; e2 < ed*th && (x1 != x2 || dx < dy); e2 += dy)
                setPixelAA(x2 += sx, y0, max(0,255*(abs(e2)/ed-th+1)));
20          если (y0 == y1) сломать;
            ошибка += dx; y0 += cy;
        }
    }
}
```

Листинг 23: Программа для построения толстой сглаженной линии

Кривые толстых линий требуют особого внимания в конечных точках. Тип конечных точек во многом зависит от приложения. Различные возможности: плоские, прямоугольные, круглые, стрелки или специальный индивидуальный. Это так же важно, если линия продолжается либо другой строкой. другой тип кривой, такой как дуги или кривые Безье более высокой степени. Поэтому это не Можно представить универсальный алгоритм для всех типов. Тем не менее, это адаптация к должныбыть возможныразличные требования.

Рисунок 35: Толстая линия со сглаживанием.

8.1.1 Улучшенный алгоритм толстой линии

Алгоритм можно оптимизировать, разделив процедуру для плоских и крутых линий.

```
voidplotLineWidth (int x0, int y0, int x1, int y1, long th)
{                          /* рисуем сглаженную линию толщиной th := 256 == 1 пиксель */
    int sx = x0 <x1? 1:-1, sy = y0 < y1? 1:-1; длинный dx = abs(x1-x0), dy =
    abs(y1-y0); длинная ошибка = dx <dy? dx: dy, e2 = dx
5   <dy? dy: dx;                                                    /* мин Макс */

    #define BKGD (255L<<16) /* максимальное значение пиксела = фон */
    if (th <= 256 || e2 == 0) returnplotLineAA (x0, y0, x1, y1); /* утверждать */
    e2 = BKGD/(e2+2*ошибка*ошибка*e2/(4*e2*e2+ошибка*ошибка)); /* sqrt-аппроксимация */
10  й = (th-256)<<16; дх *= e2; dy *= e2; /* масштабируем значения */

    if (dx <dy) { x1 =                                              /* крутая линия */
        (BKGD+th/2)/dy; ошибка =                                    /* начальное смещение */
        x1*dy-th/2; for (x0 -=                                      /* сдвигаем значение ошибки на ширину смещения */
15      x1*sx; ; y0 += sy) {
            setPixelAA(x1 = x0, y0, err>>16); for (e2 = dy-err-     /* сглаживание пикселя */
            th; e2+dy < BKGD; e2 += dy)
                setPixel(x1 += sx, y0);                             /* пиксель на толстой линии */
            setPixelAA(x1+sx, y0, e2>>16); если (y0 ==              /* псевдоним пиксела */
20          y1) сломать;
            ошибка += dx;                                           /* шаг по оси у */
            если (ошибка > BKGD) {ошибка -= dy; x0 += x; }          /* х-шаг */
        }
    } Еще {y1 =                                                     /* плоская линия */
25      (BKGD+th/2)/dx; ошибка =                                    /* начальное смещение */
        y1*dx-th/2; for (y0 -=                                      /* сдвигаем значение ошибки на ширину смещения */
        y1*sy; ; x0 += sx) {
            setPixelAA(x0, y1 = y0, err>>16); для (e2 = dx-         /* сглаживание пикселя */
            ошибочный; e2+dx < BKGD; e2 += dx)
30              setPixel(x0, y1 += sy); setPixelAA(x0,              /* пиксель на толстой линии */
            y1+sy, e2>>16); если (x0 == x1) сломать;                /* псевдоним пиксела */

            ошибка += dy;                                           /* х-шаг */
            если (ошибка > BKGD) {ошибка -= dx; y0 += cy; }         /* шаг по оси у */
35      }
    }
}
```

Листинг 24. Улучшенная программа для построения толстой сглаженной линии.

Используемое значение ошибки является центром линии. Он сдвинут к краю линии, которая позволяет легко вычислить сглаженные значения пикселей.

Программа в листинге 23 использует только сложения арифметики с фиксированной запятой при вычислении пиксельный цикл. Параметр толщины линии умножается на 256, чтобы оставаться целым числом. и включить долю толщины линий. (Если толщина линии должна быть 2,5, параметр составляет 2,5*256 = 640.)

8.2 Заполненные окружности

Сначала разрабатывается алгоритм для сглаженного заполненного круга. Процедура рисует окружная линия для линии одного квадранта. Каждая строка начинается с установки сглаженных пикселей и затем заполняет остальную участь до центра. Это приращение по оси x алгоритма нормального круга в точке в то же время . У него есть только одна проблема: иногда он слишком быстро продвигается по оси X. правильное сглаживание. Для обычного алгоритма круга сглаживания этот случай был описан в условная установка пикселя . Теперь это условие обрабатывается одним шагом назад .

```
voidplotFilledCircleAA (int xm, int ym, int r)
{                                                   /* рисуем заполненный сглаженный круг */
        int x = r, y = 0, err = 0, i;

5       for (r = 2*r+1; x > 0; err += ++y*2-1) { if (err+2*x+1 < r) err +=          /* шаг */
                ++x*2-1; for ( ; err > 0; err -= --x*2+1) { i = 255*err/     /* один шаг назад по сглаживанию*/
                r; setPixelAA (xm+x, ym+y, я ); setPixelAA(xm-y,                    /* x шаг */
                        ym+x, i); setPixelAA                    /* устанавливаем сглаженные значения пикселя */
                        (xm-x, ym-y, я ); setPixelAA (xm+y, ym-x, я );
10


                } for (i = x; i > 0; i--) { setPixel(xm+i,                  /* заполняем пиксель внутри круга */
                        ym+y); setPixel(xm-y, ym+i);
                        setPixel (xm-i, ym-y); setPixel(xm+y, ym-i);
15              }
        }
        setPixel(xm, ym);                                   /* установка центрального пикселя */
}
```

Листинг 25. Построение сглаженного заполненного круга

Ни один пиксель не устанавливается дважды

Тот же принцип можно использовать для рисования заполненного эллипса. Также возможно совмещение двух предыдущего алгоритма для рисования круга толстой линией.

Рисунок 36. Закрашенный круг со сглаживанием.

8.3 Толстые эллипсы

Два предыдущих предложения о круге с толстой линией также можно использовать для рисования толстого эллипса. Но такая комбинация приводит к проблемам для плоских эллипсов. Толщина линии становится неравномерной. На рисунке 36 изображен плоский эллипс. Черная кривая — это центральная линия. Две синие кривые — это истинные кривые смещения эллипса. Но если толщина линии нарисована эллипсами, то в результате получится контур красных кривых. Это потому, что кривая смещения уравнения (44) больше не является эллипсом. Толщина линии становится меньше на маленьких концах эллипса. Этот недостаток алгоритма становится заметен только для очень плоского эллипса с очень толстыми линиями, когда меньший радиус эллипса примерно равен толщине линии. Если меньший радиус эллипса станет нулевым, эллипс будет просто прямой толстой линией с закругленными концами. Для таких крайних примеров упрощение алгоритма становится очевидным.

Некоторого улучшения можно было бы добиться, используя пересечение с осью в качестве большего внутреннего радиуса эллипса вместо толщины линии. Эта коррекция изображена на рисунке 36 в виде зеленого эллипса. Хотя это и не полная компенсация ошибки, плоские толстые эллипсы выглядят гораздо лучше. Новый больший внутренний радиус составляет $a'^2 = (a^2-b^2)(b^2-th^2)/b^2$.

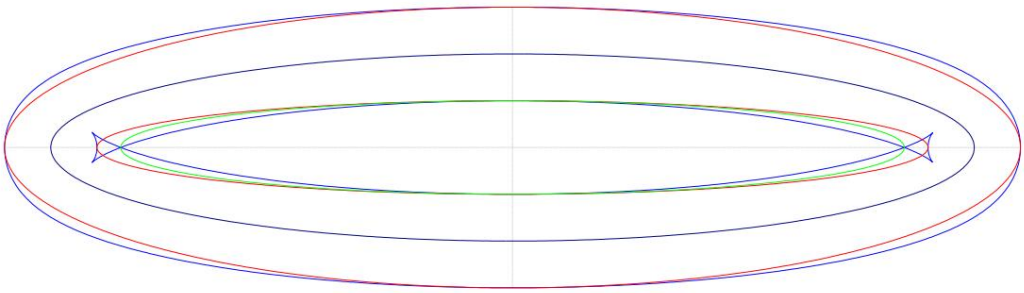


Рисунок 37: Контуры эллипса толстой линией.

На рисунке 37 показан такой плоский эллипс с толстой линией, включая компенсацию

Рисунок 38. Эллипс со сглаживанием и толстой линией.

Несмотря на эту компенсацию, алгоритм эллипса в листинге 25, использующий плоские эллипсы из тонких линий, сталкивается с проблемами. Линия становится неровной, так как ее толщина чуть меньше единицы

```
voidplotEllipseRectWidth (int x0, int y0, int x1, int y1, int th)
                          /* рисуем сглаженный эллипс внутри прямоугольника толстой линией */
{ long a = abs(x1-x0), b = abs(y1-y0), b1 = b&1; /* Наружный диаметр */
    длинный a2 = a-2*th, b2 = b-2*th; /* внутренний диаметр */
    float dx = 4*(a-1.0)*b*b, dy = 4*(b1-1)*a*a; /* приращение ошибки */
    float ed = a+b2, i, err = b1*a*a, dx2, dy2, e2 = 0;

    if (a == 0 || b == 0) returnplotLine (x0,y0, x1,y1);
    if (th < 2) returnplotEllipseRectAA (x0,y0, x1,y1);/* коррекция толстой линии */
    если ((th-1)*(2*b-th) > a*a) b2 = sqrt(a*(ba)*ed*a2)/(a-th); if ((th-1)*(2*a-th) > b*b) { a2 =
    sqrt(b*(ab)*ed*b2)/(b-th); й = (a-a2)/2; }
    если (x0 > x1) { x0 = x1; x1 += a; } /* если вывывается с переставленными точками */
    если (y0 > y1) y0 = y1; /* .. обмениваем их */
    если (b2 <= 0) th = a; /* заполненный эллипс */
    dx2 = 4*(a2-1,0)*b2*b2; dy2 = 4*(b1-1)*a2*a2; y0 += (б+1)/2; y1 = y0-
    b1; й += x0; a = 8*a*a; б1 = 8*б*б; a2 = 8*a2*a2; б2 =                    /* начальный пиксель */
    8*b2*b2;
    do
        { for (;;) { if (err
            <= 0 || x0 > x1) { i = x0; перерыв; }
            я  = fmin(dx,dy); ed = fmax(dx,dy);
            if (y0 == y1+1 && 2*err > dx && a > b1) ed = a/4; иначе ed += 2*ed*i*i/          /* х-наконечник */
            (4*ed*ed+i*i+1)+1; я  = 255*ошиб/ред; setPixelAA (x0,                  /* приблизительная площадь */
            y0, я ); setPixelAA (x0,                                  /* вне сглаживания */
            y1, я );
            setPixelAA (x1, y0, я ); setPixelAA (x1, y1, я );
            если (err+dy+1 < dx) { i = x0+1; перерыв; }
            x0++; x1--; ошибка -= dx; дх -= b1;                              /* приращение ошибки х */
        }
        for (; i <th && 2*i <= x0+x1; i++) {                                 /* пиксель заполнения линии */
            setPixel (я , y0); setPixel(x0+x1-i,y0); setPixel (я , y1);
            setPixel(x0+x1-i,y1);

        } while (e2 > 0 && x0+x1 >= 2*th) {                                  /* внутри сглаживания */
            я  = fmin(dx2,dy2); ed = fmax(dx2,dy2);
            if (y0 == y1+1 && 2*e2 > dx2 && a2 > b2) ed = a2/4; иначе ed += 2*ed*i*i/          /* х-наконечник */
            (4*ed*ed+i*i); i = 255-255*e2/ед; setPixelAA (th,                          /* аппроксимация */
            y0, i); setPixelAA(x0+x1-                          /* получаем значение интенсивности по ошибке пикселя */
            й,y0,i);
            setPixelAA (th, y1, i); setPixelAA(x0+x1-й,y1,i);
            если (e2+dy2+a2 < dx2) сломать; й++;
            e2 -= dx2; дх2 -= б2;                                            /* приращение ошибки х */
        }
        e2 += dy2 += a2;
        y0++; y1--; ошибка += dy += a;                                       /* шаг */
    } Пока (x0 <x1);
    если (y0-y1 <= b) {                                          /* слишком рання остановка плоских эллипсов */
        if (err > dy+a) { y0--; y1++; ошибка -= dy -= a; }
        for (; y0-y1 <= b; err += dy += a) {
            я  = 255*4*fabs(ошибка)/b1;                                  /* -> завершить кончик эллипса */
            setPixelAA (x0, y0, я ); setPixelAA (x1, y0++, я );
            setPixelAA (x0, y1, я ); setPixelAA(x1,y1--, я );
        }
    }
}
```

Листинг 26. Построение эллипса толстой линией

8.4 Толстая квадратичная рациональная кривая Безье

Предыдущий алгоритм линии устанавливает отдельные пиксели толстой линии перпендикулярно линии.
направление: вертикальное для плоской линии или горизонтальное для крутой линии. Этот подход сталкивается
беды, если его использовать для кривой. Если рисунок кривой меняется с плоского на крутой (или
наоборот) часть кривой отсутствует. Эти пиксели кривой должныбыть явно заданы, если
процедура меняется.

Алгоритму в основном нужна информация о значении ошибки текущего пикселя, чтобыустановить
пиксель толстой сглаженной кривой. Поэтому нет смысла рассчитывать погрешность
следующий пиксель.

```
voidplotQuadRationalBezierWidthSeg (int x0, int y0, int x1, int y1,
        int x2, int y2, float w, float th)
{               /* построить ограниченный рациональный сегмент Безье толщиныth, квадрат веса */
        int sx = x2-x1, sy = y2-y1; двойной dx =                                /* относительные значения для проверок */
5       x0-x2, dy = y0-y2, xx = x0-x1, yy = y0-y1; двойной xy = xx*sy+yy*sx, cur = xx*sy-
        yy*sx, err, e2, ed;                                                                     /* кривизна */

        Assert(xx*sx <= 0,0 && yy*sy <= 0,0);                           /* знак градиента не должен меняться */

10      if (cur != 0,0 && w > 0,0) {                                                    /* нет прямой линии */
                if (sx*(long)sx+sy*(long)sy > xx*xx+yy*yy) { /* начинаем с более длинной части */
                        x2 = x0; x0 -= dx; y2 = y0; y0 -= dy; кур = -cur;                                /* поменять местами P0 P2 */

                } xx = 2,0*(4,0*w*sx*xx+dx*dx); yy =                            /* различия 2-й степени */
15              2,0*(4,0*w*sy*yy+dy*dy);
                sx = x0 <x2? 1:-1; си = y0 < y2?                                        /* направление шага x */
                1:-1; xy =                                                              /* направление шага по оси */
                -2,0*sx*sy*(2,0*w*xy+dx*dy);

20              если (cur*sx*sy < 0) {                                          /* отрицательная кривизна? */
                        xx = -xx; yy = -yy; кур = -cur; xy = -xy;
                }
                dx = 4,0*w*(x1-x0)*sy*cur+xx/2,0; dy = 4,0*w*(y0-               /* различия 1-й степени */
                y1)*sx*cur+yy/2,0;
25
                if (w < 0,5 && (dx+xx <= 0 || dy+yy >= 0)) {/* плоский эллипс, алгоритм не получается */
                        Cur = (w+1,0)/2,0; ш= fsqrt (ш); xy = 1,0/(w+1,0);
                        sx = пол((x0+2,0*w*x1+x2)*xy/2,0+0,5); /* разделить кривуюпополам */
                        sy = Floor((y0+2.0*w*y1+y2)*xy/2.0+0.5); /* построить график отдельно */
30                      dx = пол((w*x1+x0)*xy+0,5); dy = пол((y1*w+y0)*xy+0,5);
                        plotQuadRationalBezierWidthSeg(x0,y0, dx,dy, sx,sy, cur, th); dx = пол((w*x1+x2)*xy+0,5);
                        dy = пол((y1*w+y2)*xy+0,5);
                        returnplotQuadRationalBezierWidthSeg (sx,sy, dx,dy,x2,y2,cur,th);
                }
35              for (err = 0; dy+2*yy < 0 && dx+2*xx > 0; )/* цикл по крутой/пологой кривой */
                        if (dx+dy+xy < 0) { /* крутая кривая */
                                делать {
                                        ed = -dy-2*dy*dx*dx/(4.*dy*dy+dx*dx); ш= (th-1)*ed;      /* приблизительная площадь */
                                                                                                /* масштабируем ширину линии */
```

```
                    x1 = (err-ed-w/2)/dy; e2 =                          /* начальное смещение */
                    ошибка-x1*dy-w/2; x1 = x0-                          /* значение ошибки по смещению*/
                    x1*cx; setPixelAA(x1,                               /* Начальная точка */
                    y0, 255*e2/ed); for (e2 = -w-dy-e2; e2-dy <ed;      /* сглаживание препикселя */
                    e2 -= dy) setPixelAA(x1 += sx, y0, 0); setPixelAA(x1+sx, y0,
                        255*e2/ed); если (y0 == y2) return; если        /* пиксель на толстой линии */
                    (2*(err+dx)+dy+xy > 0) { x0 += sx; dx += xy;        /* псевдоним пиксель */
                    ошибка += dy; dy+= yy;                              /* последний пиксель -> кривая закончена */
                                                                        /* e_x+e_xy > 0 */
                                                                        /* x шаг */

                    } y0 += sy; dy+= xy; ошибка += dx; dx += xx; /* шаг */
                    если (dx+2*xx <= 0 || dy+2*yy >= 0) перейти к ошибке; /* кривая рядом */
                } while (dx+dy+xy < 0);                                 /* градиент все еще крутой? */
                                                                        /* переход от крутой кривой к пологой */
                for (cur = err-dy-w/2, y1 = y0; cur < ed; y1 += sy, cur += dx) {
                    для (e2 = cur, x1 = x0; e2-dy < ed; e2 -= dy)
                        setPixelAA(x1 -= sx, y1, 0); /* пиксель на толстой линии */
                    setPixelAA(x1-sx, y1, 255*e2/ed); /* псевдоним пиксель */
                }
            } еще {                                                     /* плоская кривая */
                делать {
                    ed = dx+2*dx*dy*dy/(4.*dx*dx+dy*dy); /* приблизительная площадь */
                    ш= (th-1)*ed; /* масштабируем ширину линии */
                    y1 = (ошибка+ed+w/2)/dx; /* начальное смещение */
                    e2 = y1*dx-w/2-ошибка; /* значение ошибки по смещению*/
                    y1 = y0-y1*sy;                                      /* Начальная точка */
                    setPixelAA(x0, y1, 255*e2/ed); /* сглаживание препикселя */
                    для (e2 = dx-e2-w; e2+dx <ed; e2 += dx)
                        setPixelAA(x0, y1 += sy, 0); setPixelAA(x0,     /* пиксель на толстой линии */
                    y1+sy, 255*e2/ed); если (x0 == x2) возврат ;        /* псевдоним пиксель */
                    если (2*(err+dy)+dx+xy < 0)                         /* последний пиксель -> кривая закончена */
                    { y0 += sy; dy+= xy; ошибка += dx; dx +=            /* e_y+e_xy < 0 */
                        xx;                                             /* шаг */

                    } x0 += x; dx += xy; ошибка += dy; dy+= yy; /* x шаг */
                    если (dx+2*xx <= 0 || dy+2*yy >= 0) перейти к ошибке; /* кривая рядом */
                } while (dx+dy+xy >= 0);                                /* градиент все еще плоский? */
                                                                        /* переход от плоской кривой к крутой */
                for (cur = -err+dx-w/2, x1 = x0; cur < ed; x1 += sx, cur -= dy) {
                    для (e2 = cur, y1 = y0; e2+dx < ed; e2 += dx)
                        setPixelAA(x1, y1 -= sy, 0); /* пиксель на толстой линии */
                    setPixelAA(x1, y1-sy, 255*e2/ed); /* псевдоним пиксель */
                }
            }
        }
неудача:plotLineWidth(x0,y0, x2,y2, th*256); /* запутанные значения ошибок, приблизительные */
}
```

Листинг 27: Программа для построения сглаженной толстой квадратичной рациональной кривой Безье

Программа листинга 26 меняет порядок действий для плоской или крутой кривой во время рисования.
процесс и дополнительно устанавливает недостающие пиксели в таком случае.

Значение ошибки используется для сглаживания. Изменение значения ошибки используется как информация о толщине линии. Алгоритм предполагает, что это значение существенно не меняется от одного пикселя к другому. Если это не так, информация о толщине линии не может быть рассчитана правильно. Это делает алгоритм более чувствительным к этим значениям. Это приводит к не совсем правильно нарисованной кривой, если другая (невидимая) часть кривой снова окажется слишком близко к заданному пикселю

Поэтому алгоритм дает сбой, если радиус кривой приближается к толщине линии или оказывается ниже нее. Одной из возможностей построения таких кривых было бы использование передискретизации.

Рис. 39. Толстая квадратичная кривая Безье со сглаживанием.

На рисунке 38 показана квадратичная кривая Безье с толщиной линии 5 пикселей. Радиус кривой минимальный. Уже видны небольшие артефакты, когда направление кривой меняется с горизонтального на вертикальное. Если радиус кривой становится еще меньше, ошибка рисования становится более заметной, пока алгоритм полностью не выйдет из строя.

Вместо того, чтобы ограничивать алгоритм рисования непрерывно восходящей или нисходящей кривой, также можно обработать случай изменения направления рисования в самом пиксельном цикле. Тогда не было бы необходимости заранее подразделять кривую

8.5 Толстые кривые более высокой степени

Предыдущий алгоритм можно было применить и к кривым более высокой степени. Это работает прямо вперед, пока никакая другая кривая не приближается. Поскольку в этом типе алгоритма используются значения ошибок, находящиеся дальше от самой кривой, предположение о том, что значения разности не меняются, с большей вероятностью окажется ошибочным. Поэтому становится очень сложно рисовать более сложные кривые с острыми краями, пересечениями и петлями. Всякий раз, когда другая часть кривой подходит слишком близко, различия меняются слишком сильно, и для компенсации недостатка необходимо использовать более совершенные методы

9 сплайнов

Одиночные кривые Безье более высокой степени, чем кубические, рисовать очень сложно. Более сложные кривые образуются путем соединения квадратичных или кубических кривых Безье.

Две соседние кривые называются Ck непрерывными, если производная от 0 до k равна в точке соединения. Таким образом, непрерывность C0 просто означает, что две соседние кривые имеют общую точку. Непрерывность C1 означает, что обе кривые имеют, кроме того, один и тот же касательный вектор. Непрерывность C2 означает, что параметрические производные второго порядка общей конечной точки равны по величине и направлению

Сплайны обеспечивают большую гибкость за счет минимума элементов управления, что упрощает редактирование кривых. Более простые кривые с меньшим количеством контрольных точек — это B-сплайны. Определены только угловые точки кривой. Дополнительные контрольные точки размещаются для автоматического непрерывного соединения кривых. [Пигл и др., 1996]

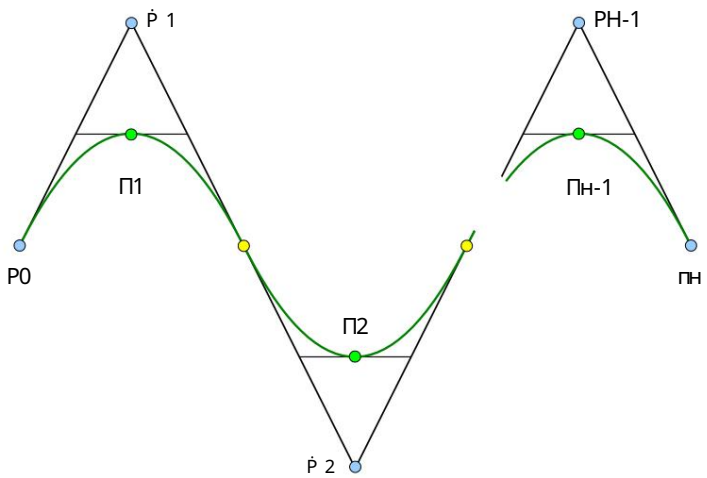


Рисунок 40: Квадратичный сплайн

Сплайны первой степени представляют собой ломаную линию

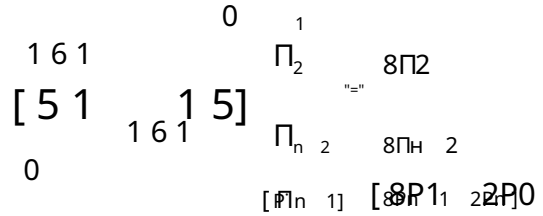
9.1 Квадратичные B-сплайны

Если угловые точки Ṕ i на рисунке 40 заданы, кривая будет прямой.
(Желтые точки находятся посередине между голубыми.) Можно ли вычислить точки Ṕ i (голубой), если заданы точки Pi (зеленые) на кривой?

Если на кривой указаны зеленые точки, угловые точки должны быть рассчитаны следующим образом:

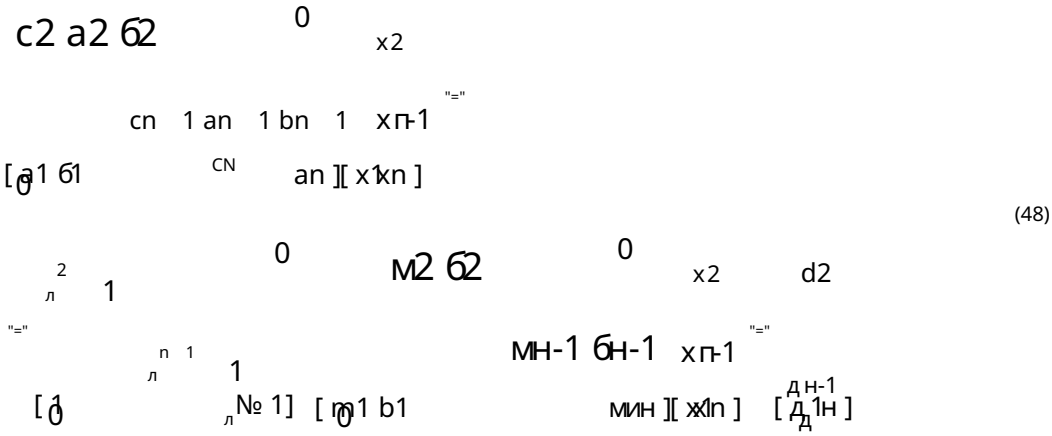
$$P'_i \quad 1 \quad 6P'_i \quad P'_i \quad 1=8Pi\,,\ 5P'_1 \quad P'_2=8P1 \quad 2P0\,,\ P'_n \quad 2 \quad 5P'_n \quad 1=8Pn \quad 1 \quad 2\,Pn\,.$$

Уравнения можно записать в матричной форме:



(47)

Эт и т рех диаг онал ьные сист емылинейных уравнений могут быт ь решеныс помощь юалг орит ма Т омаса в (n) операц ии по LU-раз ложениюмат риц ыA · x = d [Thomas, 1949]:



(48)

Мат рич ное уравнение т еперь решает ся по шаг ам: L·U·x = d, L·y = d и U·x = y.

Сравнивая услов ия , неиз вест ные можно вын ислит ь пут ем пря мог о анализ а:

м1 = a1 , li+1 = ci+1 /ми , mi+1 = ai+1 – bi li+1 и времен ный рез уль т ат y1 = d1, yi = di – yi–1 li.

Обрат ная з амена нак онец вын исля ет желаемый рез уль т ат : xn = yn /mn , xi–1 = (yi–1 – bi–1 xi)/mi–1.

Если вход ные массивыможно испол ьз оват ь в к ач ест ве времен ног о буфера, алг орит му не пот ребует ся дополнит ел ьная пама я т ь . Если эт о нежелат ел ьно, необход имо выд елит ь новые массивыд ля хранения промежут оч ных д анных.

Рез уль т ат ы

```
voidplotQuadSpline (int n, int x[], int y[])
{                          /* пост роение к вадрат ич ного сплайна, унич т ожение вход ных массивов x,y */
    #define M_MAX 6
    float mi = 1, m[M_MAX]; int i, x0,              /* диагональ ные к онст ант ымат риц ы*/
    y0, x1, y1, x2 = x[n], y2 = y[n];

    ут верждат ь (п > 1);                  /* нужно к ак минимум 3 т оч к и P[0]..P[n] */

    x[1] = x0 = 8*x[1]-2*x[0]; y[1] = y0 =                    /* первая ст рок а мат риц ы*/
    8*y[1]-2*y[0];

    for (i = 2; i <n; i++) { if (i-2 < M_MAX)                          /* перемот к а вперед */
        m[i-2] = mi = 1,0/(6,0-ми);
        x[i] = x0 = пол(8*x[i]-x0*mi+0,5); y[i] = y0 = пол(8*y[i]-          /* сохраня ем Йи */
        y0*mi+0,5);
    }
    x1 = пол((x0-2*x2)/(5,0-ми)+0,5); y1 = пол((y0-2*y2)/          /* исправление послед ней ст рок и */
    (5,0-ми)+0,5);

    для (я = п-2; я > 0; я --) {                    /* обрат ная з амена */
        если (i <= M_MAX) mi = m[i-1];
        x0 = пол((x[i]-x1)*mi+0,5); y0 = Floor((y[i]-                  /* след ующий уг ол */
        y1)*mi+0,5);
        plotQuadBezier((x0+x1)/2,(y0+y1)/2, x1,y1, x2,y2);
        x2 = (x0+x1)/2; x1 = x0; y2 = (y0+y1)/
        2; y1 = y0;
    }
    plotQuadBezier(x[0],y[0], x1,y1,x2,y2);
}
```

Лист инг 28. Рисование к вадрат ич ного сплайна по т оч к ам Pi на к ривой

Эт а прог рамма работ ает даже в т ом случ ае, если нужно нарисоват ь т оль к о од ну к ривуюБез ь е (n = 2).

Алг орит му нужен временный массив с плавающей з апя т ой д ля хранения диагональ ных з нач ений.

мат риц а: m1 = 5, mi+1= 6 –1/ми. Но пос к оль к у послед оват ель ност ь вск оре ог ранич ивает ся к онст ант ой

Лим я $\lim_{mi=3} \frac{8}{6}$ первые 6 з аписей д ост ат оч но т оч ны

Алг орит м Т омаса т ак же можно испоь з оват ь д ля периоди ч еск их сплайнов.

9.2 К убич еск ие сплайны

На рис. 41 пок аз ан к убич еск ий сплайн с д вумя раз ными к онеч ными услов ия ми. К онт роль ные т оч к и – эт о обоз нач еныв поля рной з аписи. Т оч к а P0,0,0 — д войная т оч к а, к ривиз на к от орой равна нулю

Сущест вует неск оль к о воз можност ей д ля к онеч ного сост оя ния к ривой:

- от к рыт ый к онец : к ривая прост о нарисована от P1,1,1 д о Pn-1,n-1,n-1.
- к асат ель ная : к ривиз на в к онеч ных т оч к ах P0,0,0 и Pn,n,n исч ез ает . к ривая
- период ич ес к ая : з амык ает ся формулой P0,0,0 = Pn,n,n.

В случ ае, если з ад аныугловые т оч к и (голубой), к ривуюможно раз д елит ь на к убич ес к ие к ривые Без ь е с испол ь з ованием алг оритм а Д еБура. К убич ес к ая к ривая Без ь е из влек ает ся по т оч к ам Pi,i,i-Pi,i,i+1- Pi,i+1,i+1-Pi+1,i+1,i+1. Ст рок а Pi-1,i,i+1-Pi,i+1,i+2 д елает ся т рет ей, ч т обыполуч ит ь Pi,i,i+1 и Pi,i+1,i+1. Линия Pi-1,i,i- Pi,i,i+1 д елит ся пополам, ч т обыполуч ит ь Pi,i,i.

Уравнения д ля к убич ес к их сплайнов

аналог ич ны P˙ i 1 4 P˙ i P˙ i 1=6 Pi , 7P˙ 1 2P˙ 2=12P1 3P0 , 2P˙ n 2 7P˙ n 1=12Pn 1 3Pn .

$$\begin{bmatrix} 2 & 8 & 2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & 2 & 8 & 2 & \\ & & & 2 & 7 \end{bmatrix} \begin{bmatrix} \Pi_1 \\ \Pi_2 \\ \vdots \\ \Pi_{n-2} \\ \Pi \end{bmatrix} \overset{"="}{} \begin{bmatrix} 12P1 - 3P0 \\ 12P2 \\ \vdots \\ 12Pn-2 \\ 12Pn-1 - 3Pn \end{bmatrix} \quad (49)$$

Если з ад аныск возные т оч к и (желт ые), снова можно вын ислит ь угловые т оч к и. Но эт а сист ема линейных уравнений имеет д ве ст епени свобод ы Возможным условием необход имого ог ранич ения я вля ет ся опред еление к ривиз нына обоих к онц ах равной нулю(неск оль к о уз лов): B''(0) = 0. Эт о условие пок аз ано в нач але к ривой на рисун к е 41 (P0, 0,0).

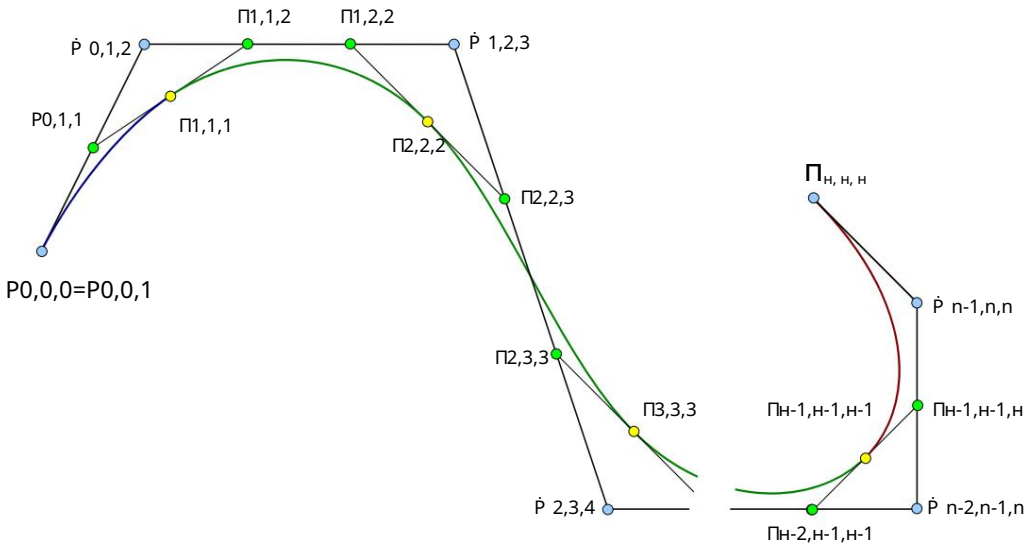


Рисунок 41: К убич ес к ий сплайн

Алгоритм Томаса снова помогает решить матричное уравнение.

```
voidplotCubicSpline (int n, int x[], int y[])
{                                   /* построение кубического сплайна, уничтожение входных массивов x,y */
    #define M_MAX 6
    float mi = 0,25, m[M_MAX]; int x3 =          /* диагональные константы матрицы*/
5   x[n-1], y3 = y[n-1], x4 = x[n], y4 = y[n];
    int i, x0, y0, x1, y1, x2, y2;

    утверждать (n> 2);                  /* нужно как минимум 4 точки P[0]..P[n] */

10  x[1] = x0 = 12*x[1]-3*x[0]; y[1] = y0 =                 /* первая строка матрицы*/
    12*y[1]-3*y[0];

    for (i = 2; i < n; i++) { if (i-2 < M_MAX)                  /* перемотка вперед */
        m[i-2] = mi = 0,25/(2,0-mi);
15      x[i] = x0 = пол(12*x[i]-2*x0*mi+0,5);
        y[i] = y0 = Floor(12*y[i]-2*y0*mi+0,5);
    }
    x2 = пол((x0-3*x4)/(7-4*mi)+0,5); y2 = пол((y0-3*y4)/       /* исправить последнюю строку */
    (7-4*mi)+0,5); plotCubicBezier(x3,y3, (x2+x4)/2,
20  (y2+y4)/2, x4,y4, x4,y4);

    если (n-3 < M_MAX) mi = m[n-3];
    x1 = пол((x[n-2]-2*x2)*mi+0,5); y1 =
    пол((y[n-2]-2*y2)*mi+0,5);
25  для (я = п-3; я > 0; я --) {                    /* обратная замена */
        если (i <= M_MAX) mi = m[i-1];
        x0 = пол((x[i]-2*x1)*mi+0,5); y0 =
        Floor((y[i]-2*y1)*mi+0,5);
        x4 = пол((x0+4*x1+x2+3)/6,0); y4 =               /* восстановить P[i] */
30      пол((y0+4*y1+y2+3)/6,0);
        сюжет CubicBezier(x4,y4,
                          пол((2*x1+x2)/3+0,5),пол((2*y1+y2)/3+0,5), этаж((x1+2*x2)/
                          3+0,5),пол((y1+ 2*y2)/3+0,5),
                          x3,y3);
35      x3 = x4; y3 = y4; x2 = x1; y2 = y1; x1 = x0; y1 = y0;
    }
    x0 = x[0]; x4 = пол((3*x0+7*x1+2*x2+6)/12,0); /* восстановить P[1] */
    y0 = y[0]; y4 = пол((3*y0+7*y1+2*y2+6)/12,0);
    plotCubicBezier(x4,y4, floor((2*x1+x2)/3+0.5),floor((2*y1+y2)/3+0.5),
40                  пол((x1+2*x2)/3+0,5),пол((y1+2*y2)/3+0,5), x3,y3);
    plotCubicBezier(x0,y0, x0,y0, (x0+x1)/2,(y0+y1)/2, x4,y4);
}
```

Листинг 29: Рисование кубического сплайна по заданным точкам кривой

Алгоритму необходим дополнительный массив с плавающей запятой для хранения значений прямой развертки, поскольку точки кривой по-прежнему необходимы для рисования вычисленного сегмента Безье. Но исходные точки могут быть восстановлены по следующей угловой точке, избегая дополнительных

множество.

Обратите внимание, что кубический сплайн размера n = 3, нарисованный с помощью этого алгоритма, не совпадает с одиночным кубическим сплайном Безье, хотя оба они определяются 4 точками. Но условием для этого сплайна является, кроме того, наличие конечных точек без кривизны, тогда как кубический Безье не имеет такого ограничения.

10Вьводы

Растеризация —фундаментальная задача компьютерной графики. Этот документ перерабатывает алгоритм Брезенхэма и расширяет его для других кривых, таких как эллипсыи кривые Безье квадратичной и кубической степени. Обычно алгоритм можно использовать для растеризации любой геометрической кривой.

10.1 Алгоритм построения нейвных уравнений

Расчетыпредыдущих глав помогают определить общий алгоритм построения кривых нейвного уравнения f x , y =0.

1. Поскольку алгоритм построения построен на непрерывно положительном или отрицательном градиенте, кривую необходимо разделить в стационарных точках, в которых градиент меняет знак (или, возможно, в точках самопересечения).
   Изменение направления также может быть включено в цикл пикселей. В подразделении нет необходимости, если поворот кривой происходит одновременно только в одном направлении (х или у). Изменение направления можно было обнаружить по изменениюзнака разностей. Значения приращения адаптируются в соответствии с новым направлением. Это решение вызывает проблемытолько в том случае, если оба направления рисования изменяются одновременно, поскольку оно
   не может решить , какой из кривых следовать .

2. Приращение шага вычисляется путем последовательных разностей нейвного уравнения : в направлении x: dx,y += dx+1,y, err += d10 и в направлении y: dx,y += dx,y+ 1, ошибка += d01.

3. Эти приращения инициализируются разностями начального пикселя .
   Разности значений ошибок (расстояния h) представляют собой конечную разность m-го порядка.
   существует в двух измерения х:

$$d_{nx\,ny=hx} \frac{1}{nx\ \text{Нью-Йорк}} \sum_{x=0}^{nx} \sum_{y=0}^{\text{тот}} 1_{\text{он}} \quad nx\ x\ \text{the}\ y\ f\ x\ nx\ i\ x\ hx, y\ ny\ i\ y\ h\ и\ [nx\ ny=m].$$

   Значение инициализации полиномиального уравнения степени n составляет n-юпроизводную
   исходя из нейвного уравнения : $$Д_{нх,\,пу=} \frac{н}{\text{тот}} \frac{ж}{x\,nx\ y} [nx\ ny=n].$$

4. Алгоритм должен проявлять особуюосторожность , если другая часть функции приближается к установить пиксель кривой.

Алгоритм можно использовать для всех функций с нейвным полиномиальным уравнением.

10.2 Сложность алгоритма

Поскольку любуюнепрерывнуюфункциюможно аппроксимировать полиномиальной функцией (теорема Вейерштрасса), в этом документе представлен алгоритм, позволяющий рисовать любуюфункциюпочти так же быстро, как линию
Более сложные кривые требуют только большего количества дополнений на пиксель .

Алгоритм первой степени (линейный) добавляет только разницу значений ошибки на каждом шаге.
Алгоритм второй степени (квадратичный) дополнительно должен отслеживать изменения разностей dx, dy.
(Эллипсыи кругипроще, поскольку некоторые разностиравныиз-за
симметричность кривой.) Алгоритм третьей степени также должен отслеживать изменения
изменения dxx, dxy, dyy.

Количество необходимых операций рисования полиномиальной кривой степени n совершает в Big O
обозначения : $n(n+1)/2 = (n^2)$.
Реализация алгоритма ограничена сложнымирасчетамиинициализирующих значений разностей. Особенно для
кривых более высоких степеней эти выражения
может стать довольно большим.

10.3 Приложения

Растеризация —фундаментальная задача компьютерной графики. Векторная графика основана на
геометрических примитивы, такие как линии, круги, эллипсыи кривые Безье. Такие кривые должны
растрироваться на всех устройствах вывода, таких как дисплеи, принтеры, плоттеры, машиныи т. д.

Алгоритм этого документа делает рисование кривых вычислительно эффективными
это также очень просто реализовать. Повседневная программа может делегировать рисование
подпрограмма, операционная система или базовый драйвер ввода/вывода. Но если приложение хочет
иметь контроль над процессомрисования или должен иметь доступ к определеннымдеталям кривой
параметр, необходимый для реализации самого алгоритма построения .

Этот тип алгоритма также может быть реализован непосредственно в электронномоборудовании (например, с
помощьюспециализированных интегральных схемилипрограммируемых пользовательских матриц). Добавление
и регистрысравнения необходимытолько для процесса рисования пиксельного цикла. Все расчетымогут быть
реализованыв целочисленных значениях. Потенциально параметрыкривой (начальные
значения), возможно, должныбыть предварительно рассчитаныс помощью короткой (микро) программы

Существующие аппаратные ускорения , такие как CUDA (Compute Unified Device Architecture) или
Nvidia или OpenGL ориентированына высокопроизводительный 3D-рендеринг для архитектурыпараллельных
вычислений. 2D-кривые Безье являются частью компьютерной 3D-сцены Такие кривые не нуждаются в
графическийпроцессор должен быть достаточно быстрым для растеризации и в большинстве случаев игнорируется
высокопроизводительными библиотеками.

10.4 Перспективы

В будущем алгоритм может быть расширен для растеризации кубического рационального Безье и
другие знакомые кривые.

Этот тип алгоритма также предполагает быструюреализациюSIMD (одна инструкция ,
Несколько данных), поскольку значения для направлений x и y могут рассчитываться независимо.

10.5 Исходный код

Примерыэтих документов доступныв Интернете. Веб-адрес:

https://zingl.github.io/bresenham.html

Программыне имеют авторских прав и могут быть использованыизмененыкемугодно по своему усмотрению

Исходный код был тщательно протестирован, но предоставляется без каких-либо гарантий. Используйте его на свой страх и риск.

Библиография

[Безье, 1986] Пьер Этьен Безье: Математическая основа UNISURF CAD
Система; Баттерворт с, Лондон, 1986 год.

[Безу, 1764] Этьен Безу: Исследование степеней уравнений, возникающих в результате исчезновения
неизвестных, и средств, которые следует использовать для нахождения этих уравнений;
История Королевской академии наук, 1764 г.

[Брезенхэм, 1965] Джек Э. Брезенхэм: Алгоритмы компьютерного управления цифровым плоттером; Системный
журнал IBM, 1965.

[Кастельжо, 1963] Поль де Фаже де Кастельжау: Кривые и поверхности полюсов; Технический
отчет, Ситроен, Париж, 1963 год.

[Кейли, 1857] Артур Кэли: Примечание о методе исключения Безу; Рейне Анж-Вандте Математик, д. 53,
366–367, 1857.

[Емельяненко, 2007] Павел Емельяненко: Визуализация точек и отрезков действительных
алгебраических плоских кривых; Магистерская диссертация, Саарский университет,
Институт компьютерных наук Макса Планка, 2007 г.

[Фоли, 1995] Джеймс Дэвид Фоли: Компьютерная графика, принципы и практика использования С; Аддисон-
Уэсли Профессионал; 2-е издание, 1995 г.

[Голдман и др., 1985] Рональд Н. Голдман, Томас В. Седерберг: Некоторые применения результирующих к
задачам вычислительной геометрии; Визуальный компьютер, том 1, номер 2, 101–107, 1985.

[Голипур-Куджали, 2005] М. Голипур-Куджали: Общий алгоритм рендеринга и сглаживания.
ритмы коничеких сечений; Лондонский университет Саут-Бэнк, 2005 г.

[Loop et al., 2005] Чарльз Луп, Джеймс Ф. Блинн: Независимая от разрешения визуализация кривых с
использованием программируемого графического оборудования; Ассоциация
вычислительной техники Inc., Microsoft Research, 2005 г.

[Марщ 2005] Дункан Марш: Прикладная геометрия для компьютерной графики и САПР; Спрингер, 2005.

[Пигл и др., 1996] Лес Пигл, Уэйн Тиллер: Книга NURBS; Спрингер, 1996.

[Рэй и др., 2011] Кумар С. Рэй, Бимал Кумар Рэй: метод отклонения для рисования неявных кривых;
Международный журнал компьютерной графики, Vol. 2, № 2,
Ноябрь 2011 г.

[Седерберг, 2011] Томас В. Седерберг: Конспект курса компьютерного геометрического дизайна;
Университет Бригама Янга, http://tom.cs.byu.edu/~557/, 2011 г.

[Шмаков, 2011] Сергей Л. Шмаков: Универсальный метод решения уравнений четвертой степени; Международный журнал чистой и прикладной математики, том 71, № 2, 251-259.

[Таубин, 1994] Габриэль Таубин: Дистанционные аппроксимации для растеризации неявных кривых; Исследовательский центр IBM Watson, 1994 г.

[Томас, 1949] Ллевеллин Хиллет Томас: Эллиптические задачи в линейных разностных уравнениях в сети; Отчет компьютерной лаборатории Watson Science, Колумбийский университет, Нью-Йорк, 1949 год.

[Сяолинь, 1991] Сяолинь Ву: эффективный метод сглаживания; Университет Западного Онтарио, 1991 год.

[Янг и др., 2000] Хунцзи Ян, Ян Ли, Юи Куй Лю Алгоритм уровня пикселей для рисования кривых; Материалы 6-й Китайской конференции по автоматизации и информатике в Великобритании, Лафборо, 2000 г.

Список рисунков

Список программ

Список уравнений