Министерство образования Российской Федерации
Санкт-Петербургский государственный институт
точной механики и оптики (технический университет)

Кафедра компьютерных технологий

С.В. Чириков

АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ГРАФИКИ

(Методы растрирования кривых)

Санкт-Петербург
2001

УДК 681.327.11: 681.3.06 : 76

Чириков С.В. Алгоритмы компьютерной графики (методы растрирования кривых). Учебное пособие – СПб : СПб ГИТМО(ТУ), 2001. – 120 с.

В пособии рассмотрены основные алгоритмы растровой графики. Основное внимание уделено различным методикам синтеза инкрементных целочисленных алгоритмов растрирования кривых, ориентированных на аппаратную и микропрограммную реализацию. Эти и подобные им алгоритмы лежат в основе программного обеспечения графических ускорителей.

Рекомендовано советом факультета Информационных Технологий и Программирования, протокол N 2 от 14.12.2000.

ОГЛАВЛЕНИЕ

Предисловие

Компьютерная графика – это раздел компьютерной математики, занимающийся синтезом изображений на экране дисплея. Практический аспект применения компьютерной графики (использование графических библиотек OpenGL, Direct3D, Java3D и т.д.) достаточно освещен во многих публикациях. Однако математика и алгоритмы, лежащие в основе синтеза изображений, описаны лишь в журнальных статьях и материалах международных конференций, которые студентам по большей части недоступны. К тому же последние монографии по компьютерной графике были опубликованы в России 10-15 лет назад, и к настоящему времени они до некоторой степени устарели.

В пособии рассмотрен круг вопросов, связанных с представлением на экране дисплея различных линий набором смежных пикселей. Такое представление называется растрированием линий, а вычислительное устройство, реализующее алгоритм растрирования, называют генератором примитивов вывода. В данном пособии рассматривается ряд алгоритмов растрирования, широко используемых в компьютерной и инженерной графике, причем описаны как классические алгоритмы, так и самые последние достижения в области растрирования линий.

Рассматриваются также методы представления кусков поверхностей. Тонирование поверхностей (закраска и текстурирование) может быть реализовано методом растрирования семейства параметрических линий при условии, что мы располагаем генератором параметрических линий.

Основное внимание уделено обоснованию математических алгоритмов растрирования, а также возможности их аппаратной реализации в виде СБИС. В основе критериев синтеза всех рассматриваемых в пособии алгоритмов лежит убеждение, что быстрый вычислительный процесс можно реализовать только на основе применения элементарных вычислительных операций - сложения, вычитания и сдвига. Применение операций умножения и деления, а также представление чисел в формате с плавающей точкой рассматриваются как нежелательные, поскольку эти операции заметно уменьшают быстродействие вычислительного устройства и резко увеличивают аппаратные затраты на его реализацию.

В качестве универсального примитива вывода рассматривается коническая дуга, вписанная в треугольник общего положения. Подробно исследованы свойства конических дуг, а также свойства конического интерполятора – вычислительного элемента, на основе которого реализуются интерполяционные вычислительные структу-

ры, составляющие основу алгоритма растрирования как конических дуг, так и их обобщений – лекальных кривых.

В пособии подробно описан генератор кривых, основанный на вычислительной структуре, состоящей из элементов двух типов – конического и линейного интерполяторов. Этот генератор растрирует линии, называемые L-кривыми или лекальными кривыми. Показано, что при определенном выборе управляющих параметров генератор лекальных кривых может быть использован для растрирования кривых Безье, которые широко применяются в инженерной графике и системах САПР.

В пособии изложены элементы теории сплайнов. Основное внимание уделено свойству локальности сплайнов. Сплайн рассматривается в данном пособии как набор сопряженных кусочно-полиномиальных кривых, представленных в форме Безье. Предложены алгоритмы пересчета управляющих параметров сплайна в управляющие параметры составляющих его кривых Безье. Там, где это возможно, алгоритмы представляются в виде матричного преобразования, причем все элементы матриц являются целочисленными.

Приведенный в данном пособии материал не охватывает весь круг вопросов, рассматриваемых в курсе компьютерной графики, который читается на кафедре компьютерных технологий ИТМО студентам программистских специальностей. В дальнейшем автор планирует создать серию пособий, которые и составят полный курс компьютерной графики.

1. Методы растрирования линий

1.1. Цепочное кодирование линии на растровой сетке

Процесс генерации растрового образа геометрического объекта принято называть растрированием (rasterization). Для отображения на экране геометрический объект представляется в виде набора типовых элементов, растрирование которых может быть выполнено с высокой скоростью встроенными аппаратными средствами графического процессора. Такими элементами, называемыми графическими примитивами вывода, в зависимости от архитектуры процессора могут быть отрезок прямой, дуга кривой второго порядка, параметрические или алгебраические сплайны, куски поверхности Кунса (Coons's patch) и некоторые другие виды кривых и поверхностей.

Для представления на экране растрового дисплея любой кривой единичной толщины необходимо найти ее цепочный код или, что одно и то же, найти координаты близких к линии смежных точек на целочисленной сетке. Этот код иногда называют кодом Фримена, по имени ученого, который впервые предложил такой способ кодирования линий. Цепочный код содержит массив единичных приращений, который используется, в частности, для управления движением пера планшетного графопостроителя.

На рис. 1 приведен пример растрирования эллипса, а на рис. 2 – возможные варианты выбора шага приращения. Непосредственно из рис. 1 можно записать цепочный код растрированного эллипса в виде $<11000077554534433> = <1^2 0^4 7^2 5^2 4534^2 3^2>$.

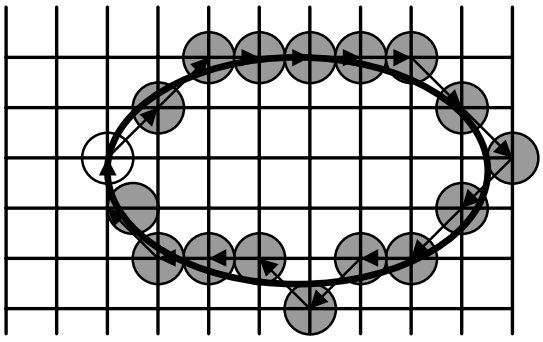


Рис. 1. Растрирование эллипса

Для цепочного кодирования можно использовать несколько стратегий выбора приращений. На рис. 2 приведены две наиболее популярные схемы: 8-точечная и 4-точечная схемы.

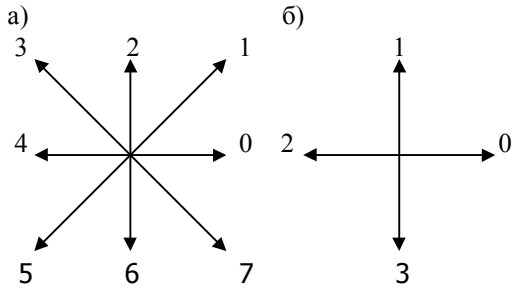


Рис. 2. Схемы выбора шага приращения:
а – восьми точечная; б – четырех точечная

Эти схемы применимы для линий, которые предназначены для отображения на растровом дисплее.

Растровый образ кривой, полученный на основе 8-точечной схемы выбора шага, выглядит несколько более гладким, чем та же кривая, растрированная с применением 4-точечной схемы. Отметим, однако, что при высоком разрешении (более 1024×756), кривые, растрированные на основе различных схем выбора шага, визуально неразличимы.

Линии, для которых существуют алгоритмы, допускающие аппаратную реализацию, называются графическими примитивами. Общепринятыми графическими примитивами считаются векторы (отрезки прямых линий), окружности (дуги окружностей) и эллипсы. Рассмотрим алгоритмы растрирования этих кривых.

1.2. Растрирование векторов методом Брезенхема

Проблема растрирования отрезков прямых линий возникла в начале 60-х годов и тогда же получила эффективное решение в работе Брезенхема [1].

Отрезок прямой, задаваемый координатами его концов, называют вектором, и долгое время он был единственным аппаратно-поддерживаемым графическим примитивом вывода. Первое решение задачи растрирования векторов было предложено Брезенхемом [1] и до самого последнего времени считалось оптимальным как с точки зрения точности и быстродействия, так и простоты аппарат-

ной реализации алгоритма. Методика Брезенхема в дальнейшем была обобщена на случай алгебраических кривых второго и третьего порядков [2,3].

Расчетная схема алгоритма приведена на рис. 3. Предположим, что координаты предшествующего пикселя P_i нам известны. Для того, чтобы решить, в каком направлении находится следующий пиксель, необходимо найти расстояния от прямой до двух пикселей-кандидатов T_i и S_i. Собственно, найти требуется не сами расстояния, а их разность $d_i = s_i - t_i$. Знак этой разности и определяет, какой из пикселей T_i или S_i будет выбран для подсветки.

В основе алгоритма лежит наблюдение, что целевая функция d_i линейно зависит от координат пикселя и может быть вычислена рекурсивно, причем для расчета положения одного пикселя требуется выполнить всего одну операцию сложения целых чисел. Алгоритм Брезенхема считается одним из наиболее быстродействующих, и он реализован практически во всех видеокартах.

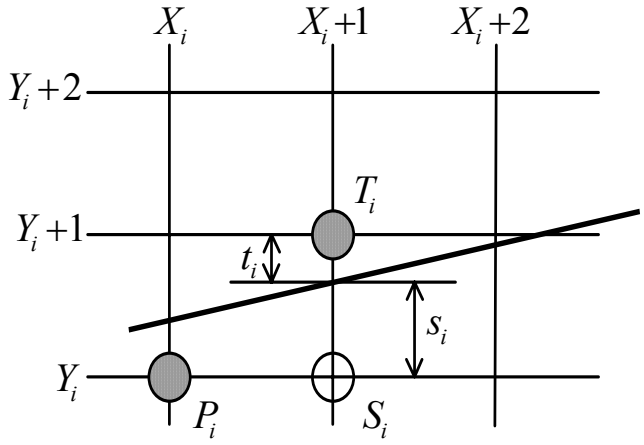


Рис. 3. Расчетная схема алгоритма Брезенхема для векторов

Пусть (X_H, Y_H) – целые координаты начала вектора, (X_K, Y_K) – целые координаты конца вектора. Будем считать, что вектор лежит в первом октанте, и тангенс наклона кривой находится в диапазоне [0,1]. Уравнение линии, проходящей через две точки, имеет вид

$$y = \frac{Y_K - Y_H}{X_K - X_H}(x - X_H), \qquad Y = \frac{V}{U}X, \qquad U \geq V,$$

где

$$X = x - X_H, \qquad x \subseteq [X_H, X_H],$$

$$Y = y - Y_H, \qquad y \subseteq [Y_H, Y_K],$$
$$U = X_K - X_H, \qquad X \subseteq [0, U],$$
$$V = Y_K - Y_H, \qquad Y \subseteq [0, V].$$

Предположим, что ближайшая к прямой узловая точка - это P_i. Необходимо выбрать следующую узловую точку. Очевидно, что претендентов будет два: точки T_i и S_i. Для того, чтобы определить, какая же из этих точек лежит ближе к прямой, найдем разницу их расстояний до прямой вдоль оси Y.

Расстояние от прямой до узла S_i составляет

$$s_i = \frac{V}{U}(X_i + 1) - Y_i,$$

а расстояние от прямой до узла T_i равно

$$t_i = Y_i + 1 - \frac{V}{U}(X_i + 1).$$

Разницу расстояний до прямой $d_i = s_i - t_i$ можно использовать в качестве целевой функции алгоритма:

$$d_i = 2\frac{V}{U}(X_i + 1) - 2Y_i - 1.$$

Очевидно, что знак d_i не изменится, если d_i умножить на U.

$$d_i = 2VX_i - 2UY_i + 2V - U.$$

Целевая функция принимает только целые значения. Принимая во внимание полученные соотношения, получим алгоритм растрирования вектора:

если $d_i < 0$, то следует выбрать узел S_i, и тогда

$$X_{i+1} = X_i + 1, \qquad d_{i+1} = d_i + 2V;$$

если же $d_i \geq 0$, то следует выбрать узел T_i, и тогда

$$X_{i+1} = X_i + 1, \qquad Y_{i+1} = Y_i + 1, \qquad d_{i+1} = d_i + 2V - 2U.$$

Таким образом, алгоритм поточечной генерации отрезка прямой (алгоритм Брезенхема [1]) можно представить в виде следующей блок-схемы (рис. 4):

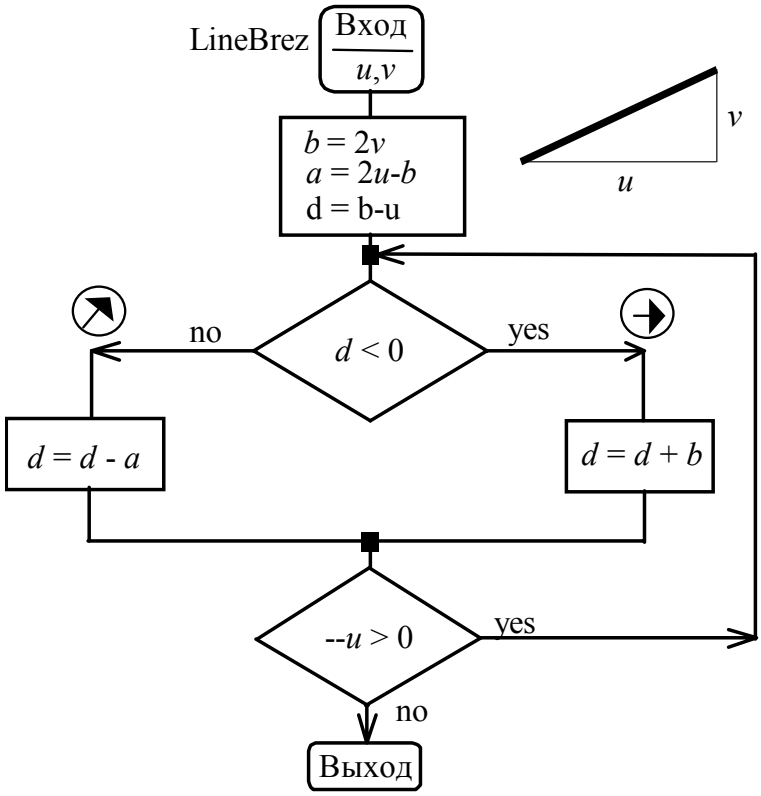


Рис. 4. Блок-схема алгоритма Брезенхема для векторов

Достоинством данного алгоритма является его целочисленность и простота аппаратной реализации. Как правило, именно этот алгоритм является основой генератора векторов в большинстве видеокарт. К его недостаткам следует отнести необходимость задания координат концов отрезка целыми числами. Далее мы покажем, что этот недостаток можно устранить, не усложняя динамическую часть алгоритма.

Для дальнейшего анализа будет удобно функционально представить генератор векторов как линейный интерполятор, имеющий один вход и один выход. На вход подаются единичные управляющие сигналы, инициирующие выполнение одной итерации алгоритма, а с выхода снимаются единичные сигналы, суммируя которые, мы получим текущую координату *Y*. Функциональная схема линейного интерполятора приведена на рис. 5.

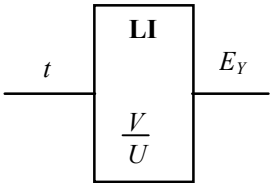


Рис. 5. Функциональная схема линейного интерполятора

1.3. Циклический характер алгоритма Брехенхема.

Приведенный на рис.4 алгоритм, выгдятит достаточно элементарным, однако он также допускает некоторые упрощения. Некоторым недостатком алогоритма яляется сравнение и ветвление. Покажем, что от этого можно избавиться.
Добавим к целевой функции `d` величину `2u`. Теперь алгоритм можно представить в следующем виде.

```
Void BrezLine1(int u, v)
{
      int U = 2*u;
      int V = 2*v;
      int D = u + v;

      for(int X=0, Y=0,; X<u; X++)
      {
            if(d > U)
            {
                  D -= U; Y++;
            }

            D +=  DrawPixel(X,Y);
      }
}
```

Этот код содержит в себе реализацию операции деления по модулю `D = D mod U` или `D %= U;` При этом, приращение `Y` – координаты происходит только когда целая часть отношения `D / U` равна `1`.
Учитывая это, алгоритм можно представить в следующем виде.

```
void BrezLine2(int u, v)
{
      int U = 2*u;
      int V = 2*v;
      int D = u;
```

```
      for(int X=0, Y=0, dY=0; X<u; X++)
      {
            D += V;
            dY = D / U;
            D %= U;
            Y += dY;
            DrawPixrl(X, Y)
      }
}
```

Теперь алгоритм стал линейным, но появилась операция деления, от которой целесообразно избавиться. Принимая во внимаеие, что все управляющие переменные можно умножить на любое неотрицательное число , и это не приведет к изменению функцтонирования адгоритма, выберем коффициент массштабирования таким, чтобы переменная `U` стала равной степени 2. В этом случае операцию деления можно будет заменить операцинй сдвига вправо, а операцию деления по модулю – маскированием. Теперь алгоритм можно представить в следующем виде.

```
void BrezLine2(int u, v)
{
      int U = 2*u;
      int V = 2*v;
      int n = (int)floor(log((dounle)U / log(2.0)) + 0.5);
      V = (V << n) / U;
      U = 1 << n;
      UINT mask = 0xFFFFFFFF << n;
      int D = U >> 1;


      for(int X=0, Y=0, dY=0l X<u; X++)
      {
            D += V;
            dY = D >> n;
            D &= mask;
            Y += dY;
            DrawPixrl(X, Y)
      }
}
```

Теперь мы получили алгоритм, который не использует медленную операцию деления. При реализации его в микрокомандах, он эквивалентен исходному алгоритму Брезенхема, хотя и не содержит операцию сравнения. Отсутствие этой операции в некоторых микропроцессорах позврляет заметно ускорить вычислительный процесс. Для нахожднгия числа сдвигов `n`, в некоторых процессорах преддусмотрена специальная микрокоманда и нахождение логарифма не требуется. Т.о., мы получили чисто целочисленный линейный алгоритм.

1.4. Растрирование векторов методом DDA

Метод DDA (Digital Differential Analyzer) или метод ЦДА (цифровой дифференциальный анализатор) основан на итеративном алгоритме последовательного вычисления точек. Свое название он получил по названию вычислительного устройства, выполняющего суммирование целых чисел и применявшегося некогда для аппаратной реализации генератора векторов. В отличие от метода Брезенхема, метод DDA допускает задание координат концов вектора в формате `double`. Приведем С-код данного алгоритма.

```
Void LineDDA(double XH, double YH, double XK, double YK,Color)
{
      double X = XH+0.5, Y = YH+0.5;
      int iXH = (int)foor(X);     // Координаты первого пикселя
                                  //     вектора (iXH, iYH)
      int iYH = (int)foor(Y);
      int iXK = (int)foor(XK+.5);// Коорд. последнего пикселя
                                  //     вектора (iXK, iYK)
      int iYK = (int)foor(YK+.5);
      int iLX = abs(iXK - iXH);
      int iLY = abs(iYK - iYH);
      int L   = max(iLX, iLY);    // Количество итераций,
                                  // необходимое для генерации
      double dX = (double)(XK - XH) / L;
      double dY = (double)(YK - YH) / L;
      while(L--)                  // Динамический цикл алгоритма
      {
            X += dX;   Y += dY;
            putpixel((int)floor(X), (int)floor(Y), Color);
      }
}
```

Более простого алгоритма (его динамический цикл) и представить себе трудно, однако алгоритм имеет очевидный серьезный недостаток: он использует операции сложения величин в плавающем формате. Хотя для современных компьютеров это не имеет такого большого значения, как несколько лет назад, при аппаратной реализации сумматор плавающих величин значительно сложнее и дороже, чем сумматор целых величин.

Алгоритм DDA можно реализовать и в целых числах. Для этого следует промасштабировать все величины `X, Y, dX, dY` и реализовать этот алгоритм в арифметике чисел с фиксированной точкой. Можно, например, старшие 16 разрядов отвести под целую часть числа, а младшие 16 разрядов отвести под дробную часть:

```
long iX  = (long)floor(X  * (1L << 16));
long iY  = (long)floor(Y  * (1L << 16));
```

```
long idX = (long)floor(dX * (1L << 16));
long idY = (long)floor(dY * (1L << 16));
```

Теперь динамическая часть алгоритма примет следующий вид:

```
while(L--)            // Динамический цикл алгоритма.
{
      iX += idX;
      iY += idY;
      putpixel((iX >> 16), (iY >> 16), Color);
}
```

Операции сдвига выполняются за один такт процессора и не замедляют заметно работу алгоритма. Отметим, что разрядность переменных целочисленной версии метода DDA вдвое превышает разрядность переменных в алгоритме Брезенхема [1].

1.5. Растрирование окружностей методом Брезенхема

Метод Брезенхема для окружностей основан на том математическом факте (рис. 6), что уравнение окружности

$$F(X,Y) = X^2 + Y^2 - R^2 = 0$$

разбивает плоскость на три части: внутреннюю $F(X,Y) < 0$, граничную $F(X,Y) = 0$ и внешнюю $F(X,Y) > 0$. Кроме того, принимается во внимание выпуклость растрируемой фигуры и ее центральная симметрия, а также считается, что координаты центра окружности и ее радиус задаются целыми числами.

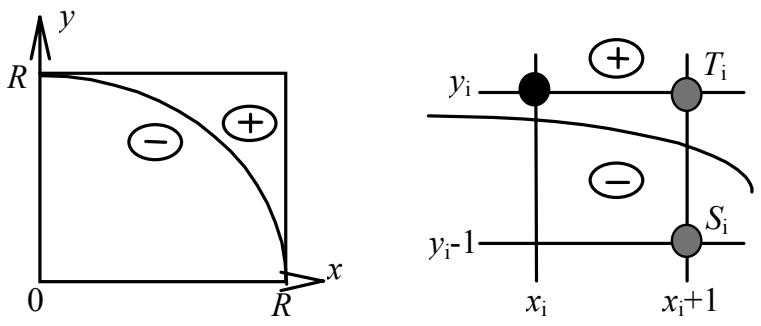


Рис. 6. Расчетная схема алгоритма Брезенхема для окружностей

Функция $F(X,Y)$ рассматривается в качестве целевой функции некоторой процедуры поиска. В качестве критерия близости узловых точек к линии окружности берется сумма значений целевой функции в двух узловых точках. Поскольку узлы - кандидаты (S_i, T_i) будут лежать по разные стороны линии $F(X,Y)=0$, то значения $F(X,Y)$ в этих узлах будут иметь противоположные знаки. В данном алгоритме сумма $F(S_i)+F(T_i)$ играет ту же роль, которую в алгоритме Брезенхема для векторов играла разница расстояний *s*-*t* до прямой. Найдем значения функции $F(X,Y)$ в узлах S_i и T_i:

$$F_S^{i+1} = (x_i+1)^2 + (y_i-1)^2 - R^2,$$
$$F_T^{i+1} = (x_i+1)^2 + y_i^{\,2} - R^2,$$

и определим целевую функцию алгоритма как

$$F_{ST}^{i+1} = \left|F_S^{i+1}\right| - \left|F_T^{i+1}\right| = F_S^{i+1} + F_T^{i+1} = 2x_i^2 + 2y_i^2 + 4x_i - 2y_i - 2R^2 + 3.$$

В начальной точке ($x_0 = 0,\ y_0 = R$) целевая функция имеет значение

$$F_{ST}^1 = 3 - 2R.$$

Таким образом, мы показали, что целевая функция F_{ST}^i является квадратичной формой, следовательно, для нее можно предложить пошаговый инкрементный алгоритм вычисления. Наша цель - выбирать направление перемещения таким образом, чтобы на каждом шаге уменьшать абсолютное значение целевой функции. Пока целевая функция отрицательна, следует продвигаться вдоль оси *X* и выбрать узел T_i. Если целевая функция положительна, то следует выбрать узел S_i:

T: if($F_{ST}^i \le 0$) then {$X = X+1$; }
S: if($F_{ST}^i > 0$) then {$X = X+1$; $Y = Y-1$; }

В алгебраической форме этот алгоритм можно записать следующим образом:

T: $u = 4x_i + 6$; $F_{i+1} = F_i + u$;
S: $v = 4(x_i - y_i) + 10$; $F_{i+1} = F_i + v$;

Представим алгоритм в виде блок-схемы (рис. 7):

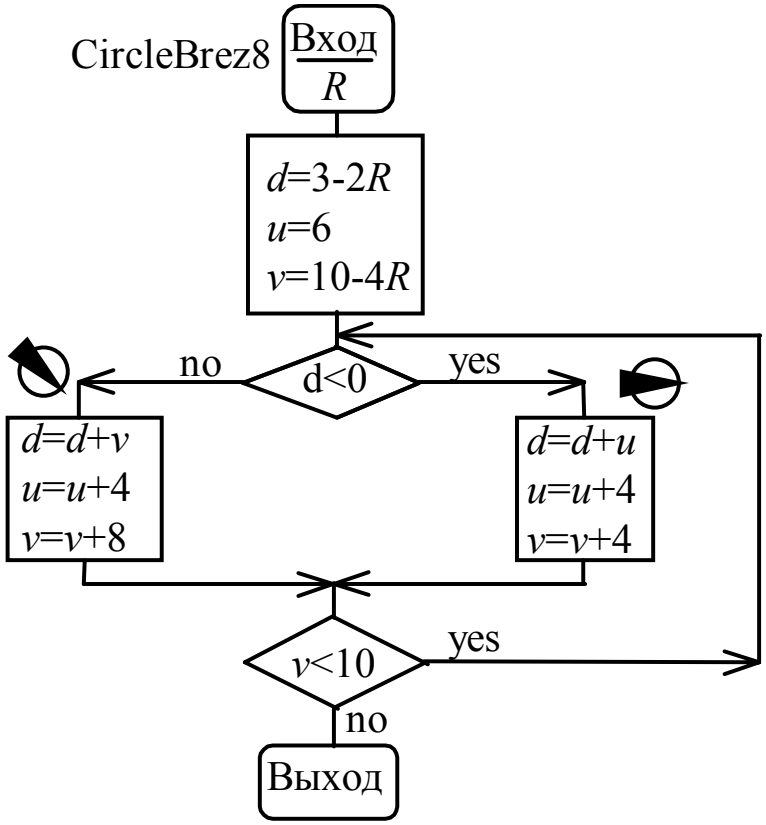


Рис. 7. Блок-схема алгоритма Брезенхема для окружностей

Достоинством данного алгоритма является его целочисленность и простота аппаратной реализации. К его недостаткам следует отнести необходимость задания координат центра окружности и ее радиуса целыми числами.

1.6. Достоинства и недостатки алгоритма Брезенхема

Как уже говорилось в разделе 1.1, исходной посылкой при выводе алгоритма Брезенхема было допущение о том, что координаты концов вектора являются целыми числами. Это допущение оправдано, если иметь в виду высокую разрешающую способность планшетных графопостроителей, для управления которыми и разрабатывался первоначально данный алгоритм. Однако при отображении векторов на экране видеотерминала обнаружился ряд визуальных

дефектов, которые, как будет показано ниже, являются следствием недостаточно точного задания положения векторов.

Негативное восприятие некоторых из этих дефектов можно заметно ослабить, если использовать описываемую в данном пособии модификацию алгоритма Брезенхема, работающую с субпиксельной (subpixel) точностью.

Основным достоинством алгоритма Брезенхема является его целочисленность и простота. Для расчета одной точки дискретного вектора требуется выполнить всего одну операцию сложения целых чисел, что обеспечивает высокую скорость растрирования вектора. При моделировании достаточно сложных геометрических объектов их координаты обычно задаются в мировой системе координат и представляются в формате с плавающей точкой. При этом все геометрические преобразования объекта также выполняются над действительными числами.

Однако для отображения на экране видеотерминала необходимо преобразовать координаты визуализируемого объекта из мировой в систему координат видеотерминала. При этом все действительные величины округляются до ближайшего целого значения и преобразуются в целые числа. Именно в момент преобразования действительных величин в целые вносится погрешность, которая является источником ряда нежелательных визуальных эффектов.

На рис. 8, а показано семейство параллельных векторов, повернутых на 20^{o}. Несложно заметить, что после поворота векторы подверглись некоторой деформации: параллельные линии перестали быть таковыми. Это особенно заметно вследствие наличия ступенчатого эффекта, причем, чем ближе находятся параллельные векторы, тем заметнее становится их “непараллельность”.

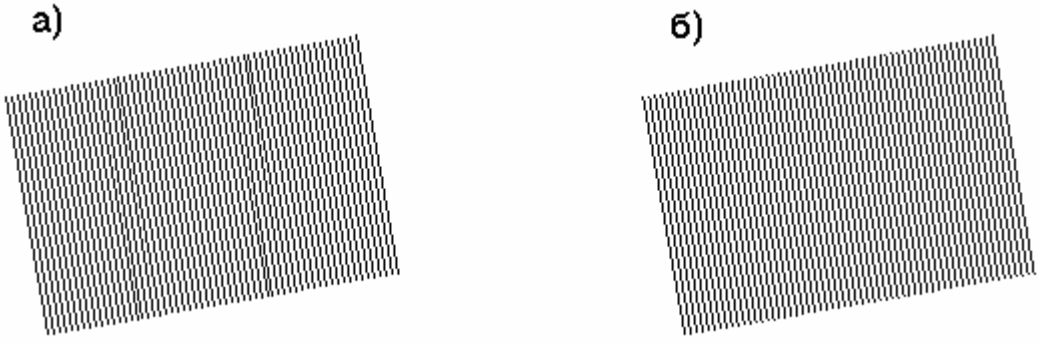


Рис. 8. Сравнительный пример представления семейства параллельных векторов, построенных с обычной (а) и субпиксельной (б) точностью

Следует отметить, что алгоритм Брезенхема растрирует векторы абсолютно точно, но поскольку перед инициализацией генератора векторов координаты концов вектора преобразуются в целые числа, то в результате выполняется растрирование близкого к исходному, но не эквивалентного ему вектора с целочисленными координатами его концов.

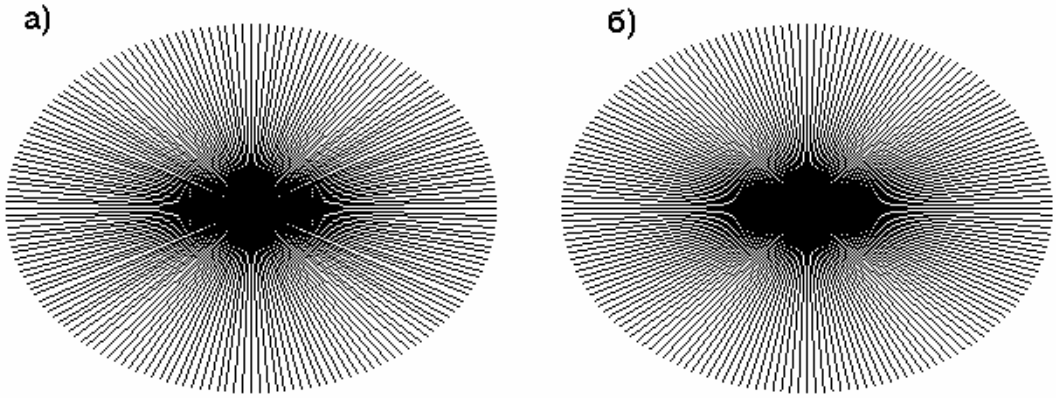


Рис. 9. Сравнительный пример представления семейства векторов, построенных методом Брезенхема с обычной (а) и субпиксельной (б) точностью

Другая иллюстрация данного эффекта приведена на рис. 9, а. На этом рисунке показано семейство векторов, проходящих через общую точку и повернутых относительно друг друга на 4^{o}. На рисунке отчетливо виден нерегулярный характер муарового эффекта. Появление муарового узора является неизбежным следствием дискретности телевизионного растра, однако "рваный" характер пятна является результатом недостаточно точного задания координат векторов при их преобразовании из вещественных величин в целые. Такое преобразование неизбежно требует округления и, как следствие, приводит к искажению исходных данных. Муаровый эффект невозможно устранить, но можно заметно ослабить его визуальное восприятие, как это показано на рис. 8, б и 9, б. На этих рисунках приведены векторы, построенные описываемым в данном пособии алгоритмом растрирования с субпиксельной точностью.

Отметим, что указанные визуальные эффекты наиболее заметно проявляются при отображении семейства близких линий и, как правило, не привлекают внимания при отображении отдельно расположенных линий.

1.7. Растрирование векторов с субпиксельной точностью

Отрезок прямой линии однозначно задается положением его концов. Представим координаты концов с субпиксельной точностью:

$$X_H = \widetilde{X}_H + \overline{X}_H 2^{-m}, \qquad Y_H = \widetilde{Y}_H + \overline{Y}_H 2^{-m},$$

$$X_K = \widetilde{X}_K + \overline{X}_K 2^{-m}, \qquad Y_K = \widetilde{Y}_K + \overline{Y}_K 2^{-m},$$

где

(X_H, Y_H) - координаты начала вектора;

(X_K, Y_K) - координаты конца вектора,

$$\widetilde{X}_H = ent(X_H), \qquad \widetilde{Y}_H = ent(Y_H),$$
$$\widetilde{X}_K = ent(X_K), \qquad \widetilde{Y}_K = ent(Y_K),$$

ent(X) - операция взятия целой части действительного числа *X*; значком "~" помечены целые значения координат; значком "-" помечены дробные значения координат, сдвинутые на *m* разрядов влево:

$$\overline{X}_H = ent(X_H \cdot 2^m) - ent(X_H) \cdot 2^m;$$
$$\overline{Y}_H = ent(Y_H \cdot 2^m) - ent(Y_H) \cdot 2^m,$$
$$\overline{X}_K = ent(X_K \cdot 2^m) - ent(X_K) \cdot 2^m;$$
$$\overline{Y}_K = ent(Y_K \cdot 2^m) - ent(Y_K) \cdot 2^m,$$

m - разрядность дробной части.

Из приведенных выше соотношений видно, что субпиксельная точность задания положения вектора, в сущности, означает представление его координат в формате с фиксированной точкой. При этом координаты кодируются целыми числами, и единственным ограничением такого представления чисел является требование, чтобы сумма разрядностей целой и дробной частей координаты не превышала разрядности представления целого числа в графическом процессоре. Для ПЭВМ эта величина равна 16 или 32. Принимая во внимание, что разрешение лучших растровых видеотерминалов не превышает 2048×1600, можно заключить, что субпиксельная точ-

ность может быть достигнута даже при использовании 16-разрядного представления целых чисел.

Уравнение прямой, проходящей через две точки с координатами (X_H, Y_H) и (X_K, Y_K), имеет следующий вид:

$$Y(X) = Y_H + \frac{Y_K - Y_H}{X_K - X_K}\left(X - Y_H\right),$$

где $X \subseteq [X_H, X_H]$, $Y \subseteq [Y_H, Y_K]$. В дальнейшем для определенности примем, что

$$(X_K \geq X_H)\,\&\,(Y_K \geq Y_H)\,\&\,(X_K - X_H \geq Y_K - Y_H).$$

Если положить, что $ent(X_H + 1/2) = ent(X_H)$ и $ent(Y_H + 1/2) = ent(Y_H)$, то, перенося центр координат в точку $(\widetilde{X}_H, \widetilde{Y}_H)$, приведем выражение *Y(X)* к виду

$$Y(X) = Y_0 + \frac{V}{U}X,$$

где

$$X \subseteq [0, \widetilde{X}_K - \widetilde{X}_H], \qquad Y \subseteq [0, \widetilde{Y}_K - \widetilde{Y}_H],$$

$$U = ent((X_K - X_H)\cdot 2^m), \qquad V = ent((Y_K - Y_H)\cdot 2^m),$$

$$Y_0 = \frac{1}{U}(\overline{Y}_H U - \overline{X}_H V)\cdot 2^{-m}.$$

Наличие ненулевого члена Y_0 в уравнении линии составляет отличие предлагаемой расчетной модели от рассмотренной Брезенхемом [1].

Для вывода алгоритма растрирования вектора рассмотрим рис. 10. Зная положение текущего пикселя (X_i, Y_i) дискретного вектора, необходимо найти положение смежного ему пикселя с абсциссой $X_i + 1$. Из рис. 10 видно, что, если расстояние $T_i = Y_i + 1 - Y(X_i)$ от правого верхнего пикселя $(X_i + 1, Y_i + 1)$ до прямой вдоль оси Y больше, чем расстояние $S_i = Y(X_i) - Y_i$ от прямой до правого пикселя $(X_i + 1, Y_i)$, то следует выбрать правый пиксель. В противном случае выбирается правый верхний пиксель.

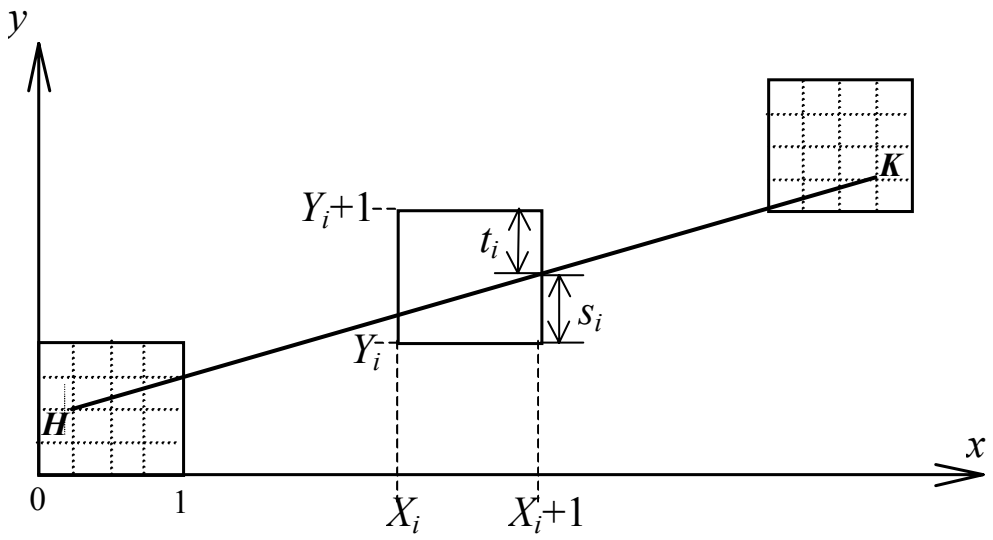


Рис. 10. Иллюстрация к выводу алгоритма растрирования векторов с субпиксельной точностью

В качестве целевой функции берется разность расстояний от прямой до смежных пикселей $D_i = S_i - T_i$:

$$D_i = 2Y(X_i + 1) - 2Y_i - 1.$$

Принимая во внимание уравнение прямой, приведем это выражение к виду

$$D_i = D_0 + 2VX_i - 2UY_i,$$

где $D_0 = 2V - U + 2UY_0$. Значение целевой функции D_i меняется при переходе к каждому следующему пикселю. Из проведенного выше анализа следует, что при выборе правого пикселя координата X получит единичное приращение, а значение целевой функции D_i увеличится на $2V$. Если был выбран правый верхний пиксель, то единичное приращение получат обе координаты X и Y, а D_i уменьшится на величину $2(U - V)$. Рекуррентный алгоритм вычисления целевой функции D_i и текущих целочисленных координат дискретного вектора имеет следующий вид:

$D = 2V - U$; // Начальная инициализация
$D_e = (\overline{Y}_H U - \overline{X}_H V) \cdot 2^{-m}$; // алгоритма
$D = D + 2D_e$;

$B = 2V$;
$A = 2U - B$;
$N = \widetilde{X}_K - \widetilde{X}_H$;
$X = \widetilde{X}_H$;
$Y = \widetilde{Y}_H$;

```
for (i=0; i<N; i++)   // Внутренний цикл алгоритма
 {
     putpixel(X,Y);
     if (D<0)
     {
          D=D+B; X++;
     }
     else
     {
          D=D-A; X++; Y++;
     }
}
```

Нетрудно заметить, что внутренний цикл алгоритма полностью идентичен классическому алгоритму Брезенхема [1]. Введение субпиксельного представления координат привело лишь к коррекции начального значения целевой функции на величину $2D_e$.

При выводе данного алгоритма были сделаны следующие допущения:

$$X_K > X_H,\ Y_K > Y_H,\ X_K > X_H,\ Y_K > Y_H .$$

Эти допущения соответствуют лишь одной из 16 возможных ситуаций, приведенных на рис. 11. Ситуации различаются по двум факторам. Во-первых, при выводе алгоритма начало системы координат переносится в точку, соответствующую положению первого пикселя растрируемого вектора. Это положение зависит от результата округления координат начала вектора. Кроме этого, приращения координат вектора могут быть как положительными, так и отрицательными. Оба эти фактора должны быть учтены при коррекции начального значения целевой функции. Для получения общего алгоритма необходимо рассмотреть все 16 вариантов. Методика вывода рекуррентных соотношений алгоритма при этом остается неизменной, поэтому мы опускаем громоздкий процесс анализа всех 16-ти вариантов и приводим на рис. 11 лишь сводку полученных результатов.

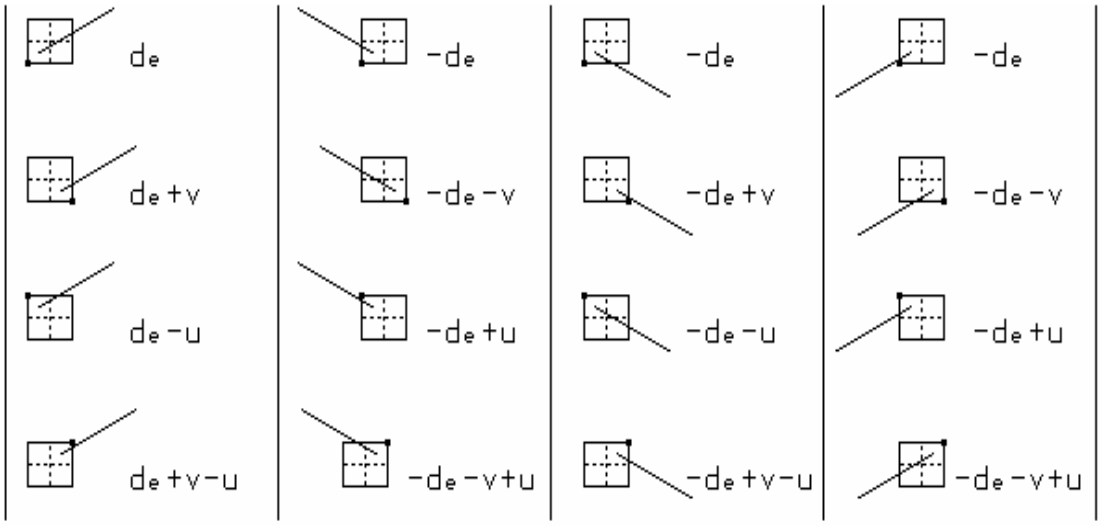


Рис. 11. Возможные варианты положения начала вектора и его направления и соответствующие этим вариантам поправки начального значения целевой функции $d_e = (Y_H u - X_H v)2^{-m}$

На рис. 11 показана связь между положением начала вектора внутри единичной ячейки растра, его направлением и величиной корректирующей поправки начального значения целевой функции алгоритма Брезенхема. Отметим, что внутренний цикл алгоритма для всех рассмотренных вариантов остался неизменным.

Если объединить все полученные результаты, то инициализация предлагаемого алгоритма будет задаваться следующими соотношениями:

$$U = ent((X_K - X_H)\cdot 2^m), \qquad V = ent((Y_K - Y_H)\cdot 2^m),$$
$$S_X = ent(X_H + 0.5) - ent(X_H),$$
$$S_Y = ent(Y_H + 0.5) - ent(Y_H),$$
$$D_e = (Y_H U - X_H V)\cdot 2^{-m},$$
$$D_s = S_X V - S_Y U,$$
$$D_0 = 2V - U,$$
$$D = D_0 + 2(D_e + D_s)\cdot sign(U)\cdot sign(V),$$
$$X = ent(X_H) + S_X,$$
$$Y = ent(Y_H) + S_Y.$$

Эти соотношения соответствуют вектору, для которого справедливо неравенство $X_K - X_H > Y_K - Y_H$. Обобщение полученных ре-

зультатов на случай $X_K - X_H \le Y_K - Y_H$ элементарно. Ниже приведена распечатка программы на языке C++, которая реализует полный алгоритм растрирования вектора с субпиксельной точностью. Изображения, приведенные на рис. 9, б и 10, б, получены с помощью этой программы.

```
#include <math.h>
#include <conio.h>

#define max(a,b)      (((a) > (b)) ? (a) : (b))
#define sign(a)       (((a) > 0)   ?  1  : (((a) < 0) ? -1:0))
#define abs(a)        (((a) > 0)   ? (a) : -(a))
#define labs(a)       (((a) > 0L)  ? (a) : -(a))

class NEWLINE
{
   private:
      long u , v , d , De , Ds , a , b;
      int x , y , sgnX , sgnY , XY , r;

   public:
      void MoveX(void)  { d += b; x += sgnX; r--; };
      void MoveY(void)  { d += b; y += sgnY; r--; };
      void MoveXY(void) { d -= a; y += sgnY; x += sgnX; r--; };
      void Init(double XH, double YH, double XK, double YK,
           int pr);
      void subLine(void);
      NEWLINE(void)  {};
      ~NEWLINE(void) {};
};

void NEWLINE::InitLine(double XH,double YH,
                       double XK,double YK, int m)
{
      int xn , yn , xk , yk , Sx , Sy , n;
      long xH , xK , yH , yK , _xH , _yH , mask;

      n=1 << m; mask = n - 1;
      Sx = (int)floor(XH + 0.5 - floor(XH));
      Sy = (int)floor(YH + 0.5 - floor(YH));
      xn = (int)floor(XH + 0.5); yn = (int)floor(YH + 0.5);
      xk = (int)floor(XK + 0.5); yk = (int)floor(YK + 0.5);
      sgnX = (int)(XK - XH);     sgnY = (int)(YK - YH);
      xH = (long)(XH * n);       yH = (long)(YH * n);
      xK = (long)(XK * n);       yK = (long)(YK * n);
      x  = xn;                   y  = yn;
      r  = (int)max(labs(xk - xn) , labs(yk - yn));

      if (abs(sgnX) >= abs(sgnY))
      {
           XY  = 1; u = xK - xH;  v = yK - yH;
           _xH = xH & mask; _yH = yH & mask;
           De  = ((_yH * u) - (_xH * v)) >> m;
           Ds  = Sx * v - Sy * u;
```

```
      }
      else
      {
            XY = 0; u = yK - yH;  v = xK - xH;
            _xH = xH & mask; _yH = yH & mask;
            De  = ((_xH * u) - (_yH * v)) >> m;
            Ds = Sy * v - Sx * u;
      }
      De  = De  << 1;  Ds = Ds << 1;
      d = 2L * labs(v) - labs(u);

      if ( ((u >= 0L) && (v >= 0L)) || ((u < 0L) && (v < 0L)) )
            d += (De + Ds);
      else
            d -= (De + Ds);

      b = labs(v + v);      a = labs(u + u) - b;
      sgnX = sign(sgnX);    sgnY = sign(sgnY);
}

void NEWLINE::subLine(void)
{
      if ( XY )
      {
            while(r)
            {
                  PutPixel(x,y);
                  if (d < 0) MoveX();
                  else       MoveXY();
            }
      }

      else
      {
            while(r)
            {
                  PutPixel(x,y);
                  if (d < 0) MoveY();
                  else       MoveXY();
            }
      }
}
void main(void)
{
      double xn,yn,xk,yk;         // координаты концов вектора
      double xc=320.2,yc=175.7,rad=150.3;
      NEWLINE NewLine;
                                  // Демонстрационный пример
      for(int i=0; i<360; i++)    // использования программы
      {
            xn=xc; yn=yc;
            xk=xc+rad*cos(M_PI*i/180);
            yk=yc+rad*sin(M_PI*i/180);
            NewLine.Init(xn, yn, xk, yk,8);// Инициализация
            NewLine.subLine();             // Генерация вектора
      }
```

```
}
```

1.7. Растрирование эллипса методом средней точки (8-точечная схема)

Для растрирования эллипса можно применить несколько иную стратегию выбора узла сетки для дискретизации кривой. Эллипс задается следующим уравнением:

$$\frac{x^2}{a^2}+\frac{y^2}{b^2}=1.$$

Представим это уравнение в виде

$$F(x,y)=y^2b^2+x^2a^2-a^2b^2=0.$$

Для упрощения математических выкладок введем следующие обозначения:

$$B=b^2, \qquad A=a^2, \qquad R^2=a^2b^2.$$

Теперь уравнение примет более простой вид:

$$F(x,y)=y^2B+x^2A-R^2=0.$$

В качестве целевой функции мы выберем $F(x,y)$, но вычислять ее значение будем в точке Q_i, лежащей посередине между узлами S_i и T_i. Стратегия выбора заключается в том, чтобы выбирать узел T_i, если значение целевой функции в средней точке отрицательно, и выбирать узел S_i в противном случае.

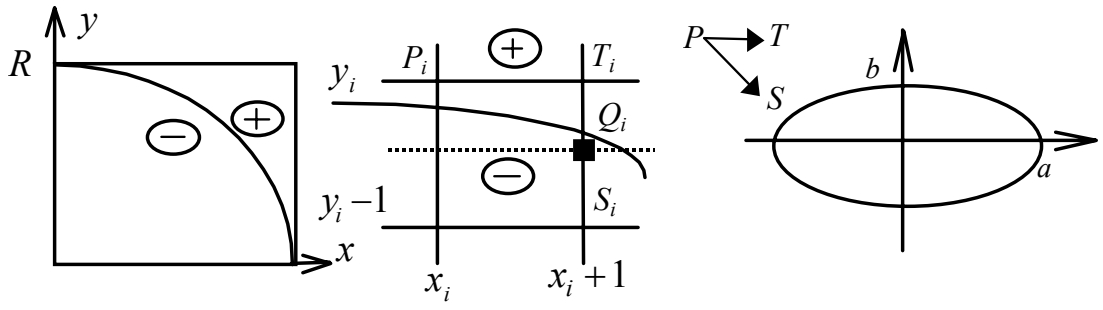


Рис. 12. Расчетная схема для метода средней точки

Легко видеть, что $F_{i+1} = \left(\frac{2y_i - 1}{2}\right)^2 B + (x_i + 1)^2 A - 4R^2$ - значение целевой функции F в средней точке (x_i+0.5, y_i+0.5). Тогда алгоритм принимает следующий вид:

T: $F_{i+1} = F_i + 8Bx_i + 12B$, $x_{i+1} = x_i + 1$,

S: $F_{i+1} = F_i + 8Bx_i - 8Ay_i + 12B + 8A$, $x_{i+1} = x + 1$; $y_{i+1} = y_i - 1$.

Введем дополнительные переменные:

$$u = 8Bx_i + 12B, \quad v = 8Bx_i - 8Ay_i + 12B + 8A.$$

И, принимая во внимание начальные значения

$$F_0 = 0, \quad u_0 = 12B, \quad v_0 = 12B+8A,$$

получаем законченный алгоритм. Заметим, что все переменные и константы кратны 4. При выводе алгоритма мы предположили, что весь анализ будем выполнять только для сегмента, касательная к каждой точке которого имеет тангенс угла наклона, лежащий в диапазоне [0,1].

В случае вектора и окружности критерий окончания работы алгоритма был очевиден. В случае эллипса это уже не так. Чтобы получить критерий окончания работы алгоритма, необходимо рассмотреть рис. 13.

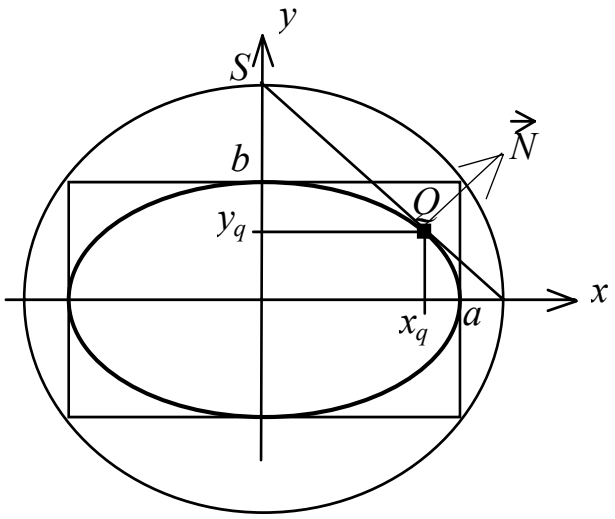


Рис. 13. Определение момента завершения работы алгоритма

Точка Q на этом рисунке соответствует последней точке, сгенерированной алгоритмом. В этой точке $\frac{dy}{dx} = -1$. Найдем координаты точки, в которой выполняется это условие:

$$x_q = \frac{a^2}{\sqrt{a^2 + b^2}},$$

$$y_q = \frac{b^2}{\sqrt{a^2 + b^2}}.$$

Из этих соотношений следует, что в точке Q имеет место отношение

$$\frac{y_q}{x_q} = \frac{B}{A}.$$

Рассмотрим следующую линейную функцию: $L(x,y) = Ay - Bx$. Очевидно, что в точке $(0,b)$ эта функция принимает положительное значение $L(0,b) = Ab > 0$. В точке $(a,0)$ $L(a.0) = -Ba < 0$, а в точке $L(X_q, Y_q) = 0$. Таким образом, мы показали, что условие $L(X,Y) < 0$ является критерием завершения работы алгоритма. На рис. 14 приведена его блок-схема.

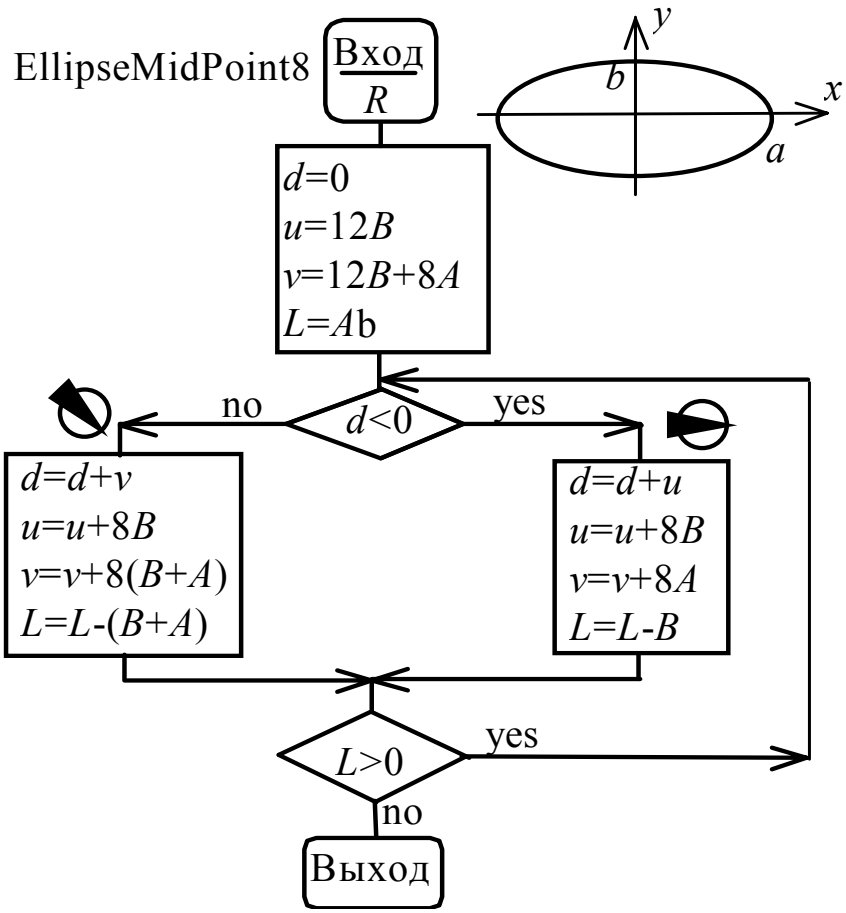


Рис. 14. Блок-схема алгоритма генерации эллипса

Описанный алгоритм с точки зрения производительности близок к алгоритму Брезенхема, однако следует отметить, что для его правильной работы необходимо представлять его переменные с достаточной разрядностью. Из условия $a^2b^2 < 2^{31}$ следует, например, что окружность, генерируемая этим алгоритмом, может иметь радиус не более 512, что является серьезным ограничением. Конечно, мы можем демасштабировать переменные, однако это внесет погрешность в работу алгоритма. В результате сгенерируется эллипс, близкий исходному, но не идентичный ему.

1.8. Растрирование эллипса методом средней точки (4-точечная схема)

Предшествующий алгоритм можно несколько упростить, если применить 4-точечную схему выбора смежных пикселей. В качестве целевой функции мы выберем $F(x, y)$, а вычислять ее значение бу-

дем в точке Q_i, лежащей посередине между узлами S_i и T_i. Стратегия выбора заключается в том, чтобы выбирать узел T_i, если значение целевой функции в средней точке отрицательно, и выбирать узел S_i, в противном случае.

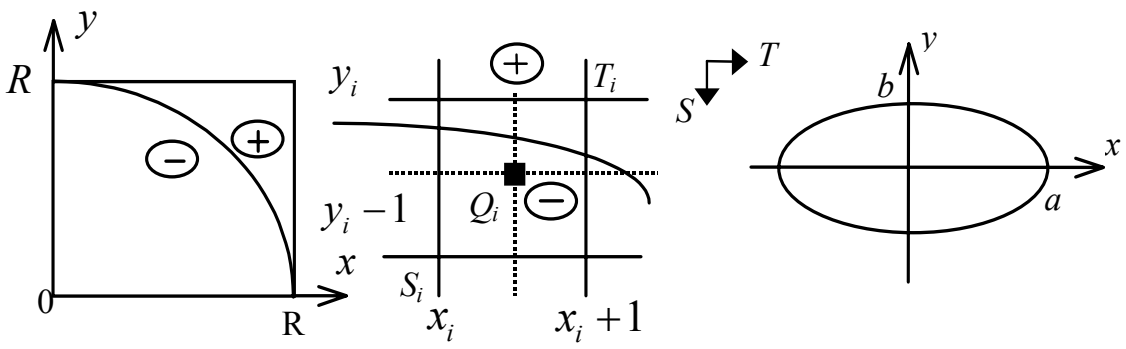


Рис. 15. Расчетная схема для метода средней точки

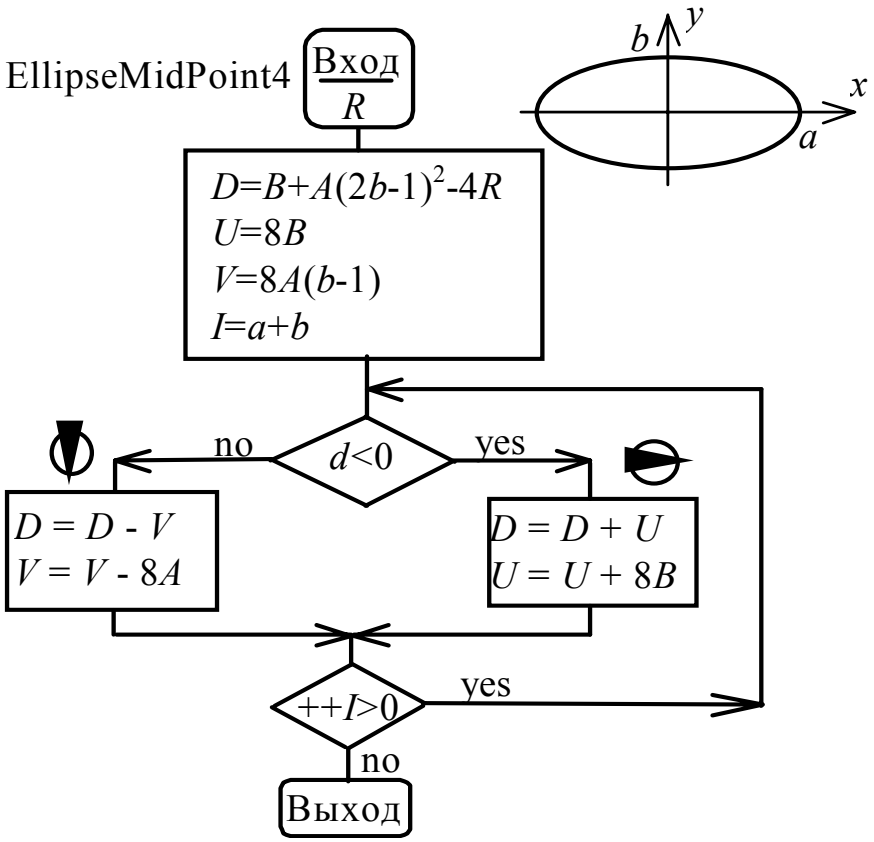


Рис. 16. Блок-схема алгоритма растрирования эллипса методом средней точки при 8-точечной стратегии выбора шага

Очевидно, что значение целевой функции в средней точке Q_i составляет

$$F_{i+1} = \left(\frac{2y_i - 1}{2}\right)^2 B + \left(\frac{2x_i + 1}{2}\right)^2 A - 4R^2.$$

Методика получения рекуррентных соотношений идентична рассмотренным выше алгоритмам, поэтому, опуская детали, приведем на рис. 16 блок-схему данного алгоритма.

Нетрудно заметить, что 4-точечный алгоритм проще, чем его 8-точечный аналог. Данный алгоритм работает в первом октанте и при любых соотношениях своих полуосей генерирует 1/4 эллипса за $a+b$ итераций.

Для дальнейшего анализа, нам потребуется также 4-точечная версия алгоритма генерации окружностей.

1.9. Растрирование окружности методом средней точки (4-точечная схема выбора шага)

Если принять, что $R=a=b$, и выполнить очевидные алгебраические упрощения, то алгоритм можно представить блок-схемой, изображенной на рис. 17.

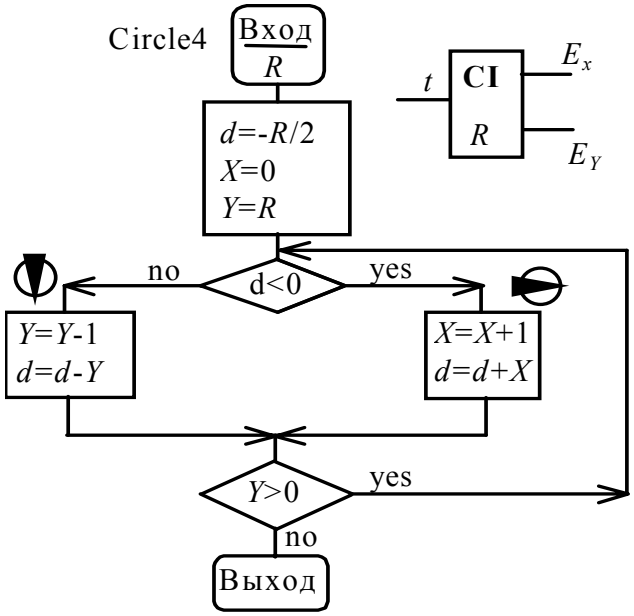


Рис. 17. Блок-схема алгоритма растрирования окружности методом средней точки при 4-точечной стратегии выбора шага

Этот алгоритм в дальнейшем будем называть алгоритмом кругового интерполятора. Функционально его можно представить как объект, имеющий один вход T и два выхода X и Y. На вход подаются единичные управляющие сигналы, каждый из которых инициирует выполнение одной итерации алгоритма. С выхода интерполятора снимаются единичные управляющие сигналы E_X и E_Y. Если просуммировать эти единичные приращения, то можно получить целочисленные координаты окружности. Четверть окружности генерируется за $2R$ итераций алгоритма.

1.10. Модификация метода Брезенхема | для генерации эллипса

Предшествующие алгоритмы растрирования эллипса основывались на задании эллипса алгебраическом уравнением, и полученные соотношения оказались весьма громоздки. Однако, если принять во внимание, что эллипс можно получить, выполнив сжатие окружности вдоль одной из осей, то можно попытаться изменить известный алгоритм растрирования окружности таким образом, чтобы он игнорировал часть приращений координат вдоль оси, соответствующей меньшей полуоси эллипса. На рис. 18 показаны оба варианта $a>$b и $a<b$:

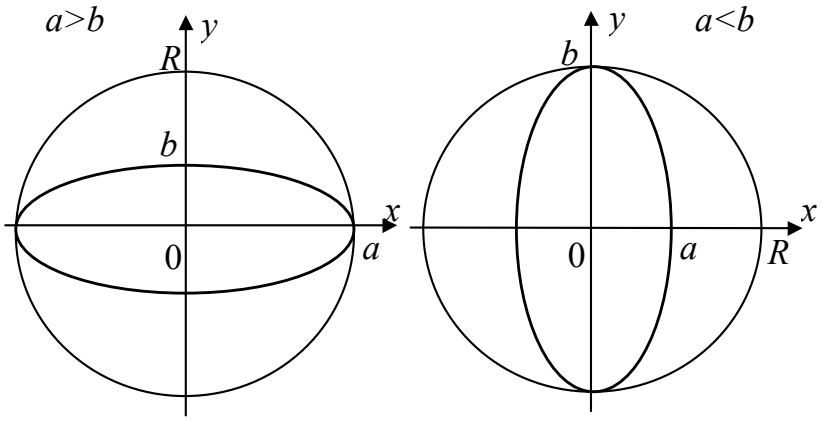


Рис. 18. Варианты ориентации коаксиальных эллипсов

Рассмотренный выше алгоритм Брезенхема для векторов выполняет поточечную генерацию вектора с квази-постоянной скоростью. При этом последовательность обеих инкрементных команд →↗ выдается с постоянными частотами таким образом, что y-

приращения генерируются в $\frac{v}{u}$ раз медленнее, чем x- приращения. Из этого наблюдения следует, что алгоритм Брезенхема для векторов может использоваться в качестве своеобразного фильтра для задания соотношения между частотами повторения инкрементных команд. В данном случае, если $a>b$, то игнорируется часть y-приращений. Если $a<b$, то уменьшается количество x-приращений. Функциональная блок-схема алгоритма, реализующего эту стратегию, приведена на рис. 19.

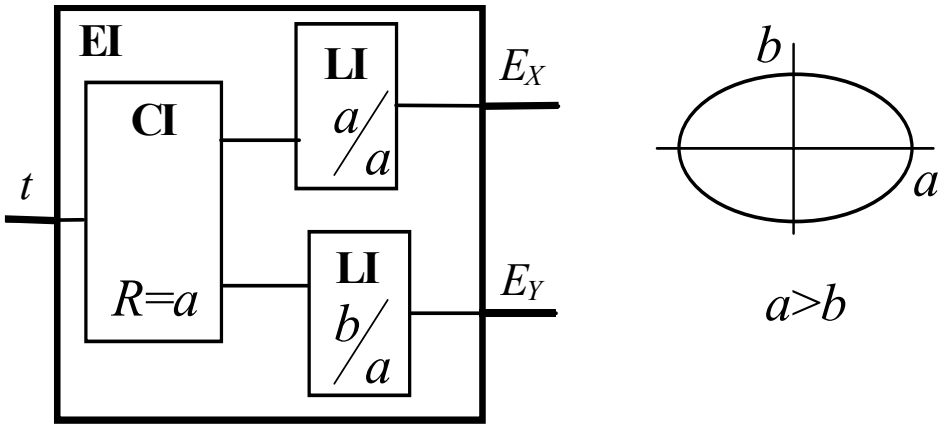


Рис. 19. Функциональная схема генератора коаксиальных эллипсов

Этот алгоритм выглядит несколько громоздко, но фактически он состоит из двух линейных и одного кругового интерполяторов, и аппаратные затраты на их реализацию не чрезмерны.

1.11. Растрирование цветных векторов

Мы рассмотрели методы растрирования линий, обращая основное внимание на степень близости пиксельного образа к геометрическому положению линии. Однако концы линии могут быть заданы не только их положением в пространстве, строго говоря, не только в 2D пространстве. Начало и конец линии могут быть окрашены разными цветами. Таким образом, необходимо растрировать 2D линию в 5D пространстве, а координаты пикселя должны включать цвет (X,Y,R,G,B).

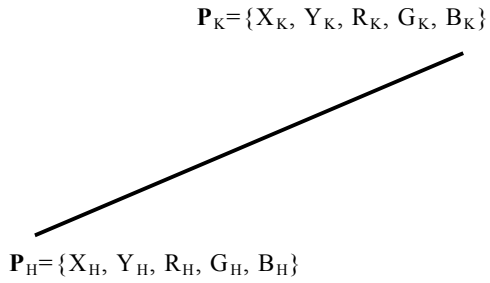


Рис. 20. Задание вектора в 5D пространстве положения-цвета

Все составляющие 5-мерного вектора должны генерироваться с постоянной скоростью. Мы имеем устройство, которое может это сделать - линейный интерполятор, реализованный по методу Брезенхема или по методу DDA. Теперь генератор цветных векторов можно представить в виде набора 1D линейных интерполяторов, как это показано на рис. 21.

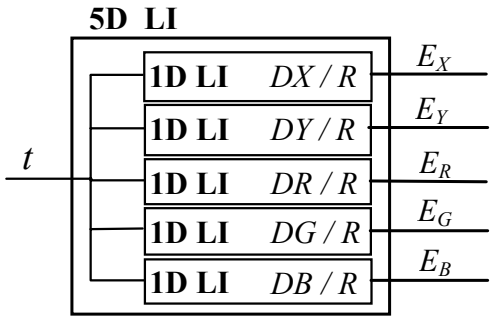


Рис. 21. Функциональная схема генератора многомерных векторов

Из этой функциональной схемы видно, что генератор векторов представляет собой многомерный линейный интерполятор. Для инициализации этого интерполятора следует рассчитать следующие величины:

$DX = abs(X_K - X_H)$, $DY = abs(Y_K - Y_H)$,
$DR = abs(R_K - R_H)$, $DG = abs(G_K - G_H)$, $DB = abs(B_K - B_H)$,
$R = max(DX, DY, DR, DG, DB)$ - количество итераций.

Если R определяется величинами DX или DY, то обсуждать больше нечего. Если же приращение цветовых координат (DR,DG,DB) превышает DX и DY, то расчет цвета пикселя методом Брезенхема становится более сложным. Не вникая в детали реализации, отметим,

что в этом случае, чтобы определить цвет пикселя, достаточно найти среднее значение цветов «ступеньки».

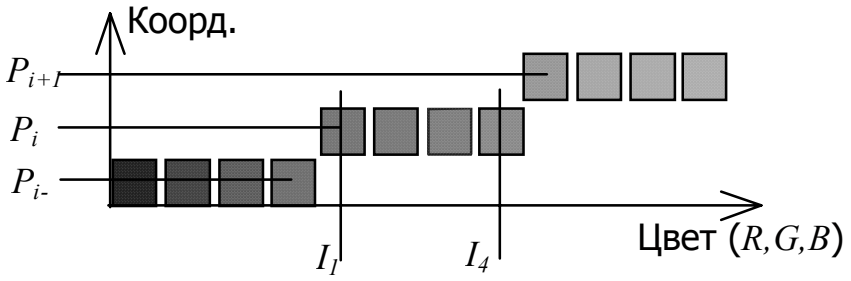


Рис. 22. Интерполяция в пространстве положения и цвета

Например, для пикселя P_i на рис. 22 интенсивность его цветовой компоненты будет равна $I_P=(I_1+I_4)/2$; Достаточно взять крайние точки ступеньки, так как интерполяция линейная.

Из приведенных рассуждений видно, что необходимо модифицировать алгоритм Брезенхема таким образом, чтобы он запоминал первую и последнюю точку ступеньки. Программно это реализуется элементарно, и аппаратное решение также возможно.

Отметим, что метод ЦДА в этом случае не требует никакой модификации, однако количество пикселей находится только из геометрических координат вектора, и приращение цвета может быть в этом случае неединичным.

2. Ступенчатый эффект и методы его устранения

При отображении на растровом дисплее линии постоянного цвета мы видим, что эта линия состоит из последовательности горизонтальных или вертикальных прямых, образующих своего рода ступени. Этот эффект называется ступенчатым (лестничным, aliasing) эффектом. Он не зависит от точности представления гладкой линии последовательностью пикселей и в этом смысле является неустранимым. В наибольшей степени лестничный эффект проявляется при рисовании линий, наклон которых лишь незначительно отличается от направления координатных осей. В этом случае линия визуально выглядит как несколько последовательных параллельных ступеней. Пример ступенчатого эффекта можно видеть на рис. 23.

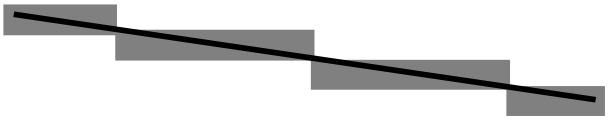


Рис. 23. Появление ступенчатого эффекта при растрировании векторов

Ступенчатый эффект нельзя полностью устранить, но его можно существенно ослабить, если для каждого пикселя варьировать его цвет от максимального значения (если линия проходит через центр пикселя) до цвета фона (если линия проходит вдали от пикселя).

Для упрощения рассуждений будем считать, что цвет фона – черный, и ему соответствует нулевая интенсивность свечения пикселя. Цвет линии будем считать белым, и ему будет соответствовать единичное значение интенсивности свечения. В дальнейшем мы обобщим предлагаемую методику на общий случай цветной линии. Таким образом, интенсивность свечения пикселей, иначе говоря, оттенки серого цвета, будут принимать значения в диапазоне от 0 до 1. Рассмотрим ситуацию, приведенную на рис. 24.

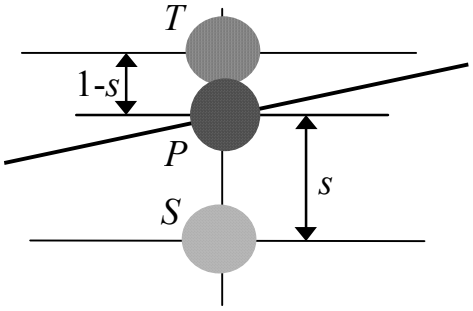


Рис. 24. Расчетная схема устранения ступенчатого эффекта методом линейной интерполяции интенсивностей светимости смежных пикселей

В этом случае истинное положение пикселя *P*, понимаемое как точка пересечения линии с вертикальной прямой, проходящей через узлы растровой сетки, лежит между двумя узлами *S* и *T*. Согласно ранее рассмотренным методикам дискретизации кривых, мы должны были бы выбрать узел *T* и подсветить его цветом линии. Однако в этом случае мы получили бы все те визуальные дефекты, от которых мы собирается избавиться.

Один из способов устранения лестничного эффекта заключается в том, чтобы представить пиксель *P* двумя пикселями *T* и *S*, причем интенсивность этих пикселей должна быть такой, чтобы суммарный вклад светимостей этой пары был эквивалентен светимости исходного пикселя *P*.

Из рис. 24 видно, что это условие будет выполнено, если принять следующие соотношения:

$$I_T = I_P s, \qquad I_S = I_P (1 - s).$$

Из этих соотношений видно, что при $s = 0$ пиксель *P* совпадает с узлом *S* и имеет максимальную интенсивность. То же имеет место при $s = 1$ относительно узла *T*. Очевидно, что в данном методе величина s равна расстоянию между пикселем P и узлом сетки *S*. Отметим, что если s не будет равна собственно расстоянию, а будет некоторой монотонной функцией от него, приведенные энергетические соотношения все равно останутся справедливыми.

Таким образом, задача свелась к определению расстояния от пикселя до узла сетки. Покажем, что для случаев вектора и окружности эта задача разрешима элементарными средствами, не усложняющими существенно исходные алгоритмы растрирования.

2.1. Генерация векторов с устранением лестничного эффекта

В рассмотренном ранее методе Брезенхема для растрирования векторов мы вычисляли целевую функцию, которая являлась кусочно-линейной и изменялась в диапазоне [*0, -2u*]. Очевидно, что расстояние S также будет периодичной кусочно-линейной функцией с тем же периодом, но изменяться оно должно от *-u* до *u*. Этого можно достичь, если ввести дополнительную переменную *E* с обнуленным начальным значением. Для того, чтобы пронормировать переменную *E*, ее следует демасштабировать, разделив на *u*. Если принять, что интенсивность пикселя изменяется от 0 до 255, то все переменные следует умножить на 255.

На рис. 25 вертикальный скачок имеет место при изменении знака целевой функции d. Блок-схема рис. 26 реализует алгоритм растрирования вектора с устранением лестничного эффекта (antialiasing).

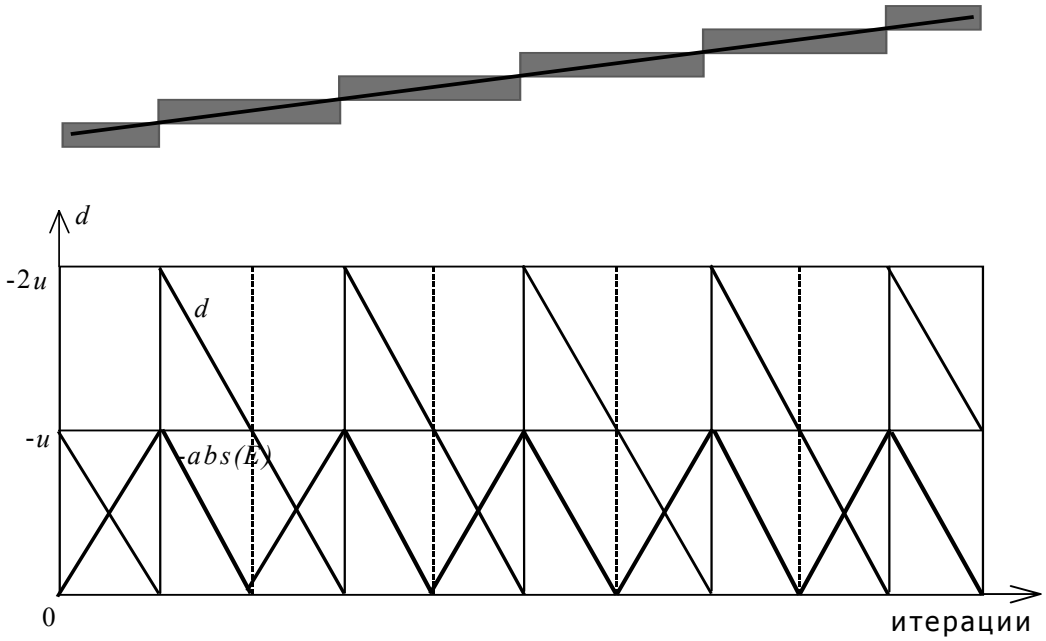


Рис. 25. Периодическая кусочно-линейная зависимость целевой функции от числа итераций алгоритма и наклона растрируемой прямой

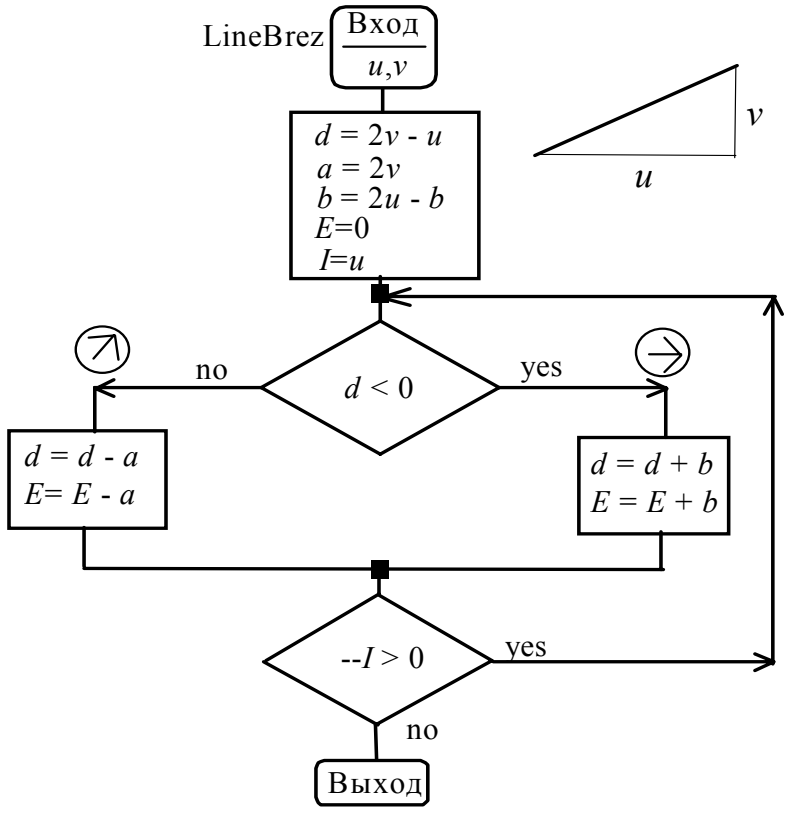


Рис. 26. Блок-схема алгоритма Брезенхема для векторов с устранением ступенчатого эффекта

Алгоритм динамически вычисляет новую переменную *E*, однако в данной блок-схеме не отражено ее использование. Эта переменная используется при подсветке пикселя. Для большей наглядности приведем С-код процедуры, вычисляющей интенсивности пары пикселей.

```
void _PUT_PIXEL(CDC *pDC, int X, int Y, int sx, int sy,
                long E, BYTE Color)
{
      long C1, C2;
      int  S1=0, S2=0;
      if(Color)
      {
            //  C2 = (long)Color*(       abs(E))/(2u);
            C2 = labs(E)>>12;
            //    C1 = (long)Color*(2u - abs(E))/(2u);
            C1 = (long)Color - C2;

            S1 = TabCol[Color][C1];     // Beta - correction
            S2 = Color - S1;
      }

      pDC -> SetPixelV(X, Y, RGB(S1, S1, S1));
                                        // C1 -  first pixel T
      if(E<=0)
            pDC -> SetPixelV(X-sx, Y-sy, RGB(S2, S2, S2));
                                        // C2 - second pixel S
      else
             pDC -> SetPixelV(X+sx, Y+sy, RGB(S2, S2, S2));
                                        // C2 - second pixel S
}
```

В этой процедуре таблица `TabCol[Color][C]` содержит все возможные значения интенсивностей пикселя, подвергнутые Beta – коррекции, назначение которой мы рассмотрим позже. Приведем С-код процедуры, генерирующей вектор с устранением лестничного эффекта.

```
void CDrawEllipse4View::Line_XgtY(CDC *pDC, int x0, int y0,
                                  int x1, int y1, BYTE Color)
{
      long A, B, D, E, U, V, X, Y, Sx, Sy, I, Col;
      X = (long)x0; Y = (long)y0; Col = (long)Color << 12;

      Sx = Sy = 0L;
      if(x1 > x0)       Sx= 1;
      else if(x1 < x0)  Sx=-1;

      if(y1 >= y0)      Sy= 1;
      else if(y1 < y0)  Sy=-1;
```

```
        U = abs(x0 - x1); V = abs(y0 - y1); I=U;
        B = 2*V; A = 2*(U-V); D = B-U;
        U *= 2;  E = 0;

        if(Color)
        {
              Col /= U; A *= Col; B *= Col; D *= Col;
        }

        while(I--)
        {
              _PUT_PIXEL_(pDC, X, Y, 0, Sy, E, Color);
              if(D<0)
              {
                    D +=B; E +=B;
              }
              else
              {
                    D -= A; E -= A; Y += Sy;
              }
              X += Sx;
        }
}
```

2.2. Gamma - коррекция

Мы нашли алгоритм, позволяющий рассчитать расстояния от истинного положения пикселя до ближайших узлов сетки, и можем теперь рассчитать значения интенсивностей пикселей в этих узлах. Однако, если мы сгенерируем линию по предложенной методике, то обнаружим, что рисуется не сплошная линия, а штриховая. Этот визуальный эффект является следствием изменения интенсивностей смежных пикселей. Собственно говоря, этого мы и добивались, однако эффект оказался не тем, на который мы рассчитывали. Это связано с тем, что человеческий глаз - прибор нелинейный. Он воспринимает не столько абсолютное значение светимости пикселя, сколько контраст между соседними пикселями.

Если задать непрерывное изменение интенсивности светимости на некоторой прямоугольной полосе, то мы увидим явно нелинейно освещенную полосу. Для того, чтобы распределение светимости стало изменяться линейно (с субъективной точки зрения), каждое значение светимости необходимо подвергнуть нелинейному искажению. Именно такое искажение и называется Gamma - коррекцией. Gamma - коррекция задается следующей формулой:

$$I' = I_m \left(\frac{I}{I_m} \right)^{1/\gamma},$$

где I - величина интенсивности светимости пикселя (цвет пикселя) $I \subseteq [0, I_m]$, I_m - величина интенсивности светимости линии (цвет линии), γ - коэффициент искажения, определяющий степень коррекции, $\gamma \subseteq [2.2 \div 2.5]$. В этой формуле цвет фона считается черным, а его светимость - равной нулю. Если фон не черный, а серый, то это можно учесть следующим образом:

$$I' = I_{bkgr} + (I_m - I_{bkgr}) \left(\frac{I - I_{bkgr}}{I_m - I_{bkgr}} \right)^{1/\gamma},$$

где I_{bkgr} - цвет фона, на котором рисуется линия.

В приведенных выше рассуждениях предполагалось, что яркая линия рисуется на тусклом фоне *($I > I_{bkgr}$)*, но возможно и противное, например, в Windows в большинстве случаев фон - белый, а графика, по умолчанию, - черная. Для случая *($I < I_{bkgr}$)* формула принимает вид

$$I' = I_{bkgr} - (I_{bkgr} - I_m) \left(\frac{I_{bkgr} - I}{I_{bkgr} - I_m} \right)^{1/\gamma}.$$

Сравнивая формулы коррекции для обоих случаев, находим, что исходная формула учитывает и второй случай тоже. Таким образом, приведенная выше формула, являясь верной, в то же время оказалось излишней. Этот факт пришлось доказывать, поскольку изначально он был не очевиден. Заметим, что приведенный выше фрагмент С-кода предполагает, что фон - черный.

Мы ставили перед собой задачу - найти линейную комбинацию интенсивностей двух соседних узловых пикселей, которая могла бы заменить один пиксель, имеющий нецелую координату *Y*. И мы решили эту задачу, модифицировав известный алгоритм растрирования векторов. Мы правильно рассчитали интенсивности светимости пикселей, т.е. их энергию излучения. Но для того, чтобы получить эффект непрерывности, линейно интерполироваться должна не энергия излучения, а энергия восприятия. Это можно учесть, если рассчитывать интенсивности пикселей с учетом Beta – коррекции по формулам

$$I_S = I_P(1-s),$$

$$I'_S = I_{bkgr} + (I_P - I_{bkgr})\left(\frac{I_S - I_{bkgr}}{I_P - I_{bkgr}}\right)^{1/\gamma},$$

где I_S - интенсивность свечения пикселя S, I'_S - интенсивность восприятия глазом свечения пикселя S, $I'_T = I_P - I'_S$ находится из требования сохранения энергетического баланса.

В приведенных выше программах для учета *Beta* - коррекции при запуске программы рассчитывалась таблица пересчета интенсивностей в их скорректированные значения. Использование этой таблицы позволило избежать медленных операций возведения в степень:

```
BYTE TabCol[256][256];
const double alf = 1./2.3;       // Gamma = 2.3;

for(int Color=0; Color<256; Color++)// Color - line color IP
for(int C=0; C<=Color; C++)     // C - current color:
                                // Is = [0, Color]; Is=(1-s)Ip;
{
      TabCol[Color][C] =        //   I'= I^alf * Ip^(1-alf)
                  (BYTE) (pow(C, alf) * pow(Color, 1.0-
            alf)+0.5);
}
```

3. Генерация конических дуг общего положения

Мы рассмотрели уже достаточное число алгоритмов растрирования, чтобы заметить, что все они имеют много общего, а именно, несмотря на различие в инициализации все алгоритмы имеют практически одну и ту же динамическую петлю. Это сходство не случайно, поскольку все рассмотренные нами до сих пор кривые были алгебраическими кривыми второго порядка.

В научной литературе описан алгоритм растрирования квадратичной формы. Этот алгоритм Питтвея-Боттинга (Pittway-Botting) [2] обобщает методику Брезенхема для векторов на случай квадратичных форм. Полученные соотношения оказались весьма громоздкими. При работе алгоритм может потерять устойчивость, если касательные в соседних пикселях имеют разность углов наклона более 45^o, то есть данный алгоритм не позволяет растрировать кривые, близкие к вырожденным (сильно сжатые эллипсы и т.п.). Но главным недостатком этого алгоритма является требование задавать входные параметры в виде коэффициентов квадратичной формы.

Мы уже рассмотрели практически все частные случаи, когда такое задание возможно и имеет очевидный геометрический смысл - это окружности и эллипсы частного положения.

Пересчет геометрически заданной дуги в эквивалентную ей квадратичную форму – относительно трудоемкая задача, а полученные коэффициенты часто невозможно представить в целом виде из-за чрезмерных требований к разрядности переменных. Вспомним хотя бы соотношения, полученные нами для эллипса частного положения, и сделанную нами оценку его допустимых размеров. Далее ситуация еще ухудшается, и становится очевидным, что возможности такого подхода к синтезу алгоритмов растрирования ограничены.

С другой стороны, нельзя не отметить простоту алгоритмов растрирования квадратичных форм. Это безусловное достоинство методики Брезенхема, и мы постарается сохранить его при синтезе алгоритмов растрирования более общих кривых, чем те, которые были нами уже рассмотрены.

3.1. Геометрический способ задания конической дуги

Возможен иной подход к организации растрирования кривых. Он основан на известном из геометрии свойстве конических кривых сохранять тип кривой при аффинном ее преобразовании. Другими словами, эллипс переходит в эллипс, парабола - в параболу, гипербола - в гиперболу. Из этого следует, например, что коника частного положения, вписанная в квадрат, после аффинного преобразования будет вписана в параллелограмм, как это показано на рис. 27.

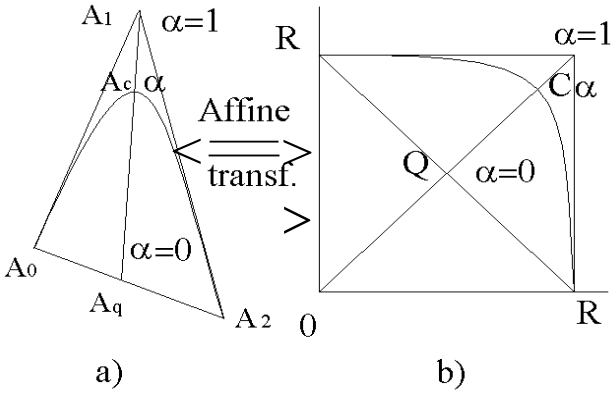


Рис. 27. Отношение трех точек α - инвариант аффинного преобразования

Вписанная в квадрат кривая второго порядка описывается следующим уравнением:

$$X^2 + Y^2 - 2ZXY + 2ZR(X + Y) - (2Z + 1)R^2 = 0.$$

Это уравнение удовлетворяет краевым условиям:

$\frac{dY}{dX} = 0$ - в начальной точке (0,*R*),

$\frac{dX}{dY} = 0$ - в конечной точке (*R*,0).

Параметр *Z* определяет тип коники и является ее эксцентриситетом. Этот факт доказывается в курсе аналитической геометрии вычислением инвариантов квадратичной формы. Эксцентриситет коники однозначно определяет ее вид, Так для эллипса $Z \in (-1,1)$, для окружности $Z = 0$, для параболы $Z = 1$, для гиперболы $Z > 0$. Поскольку мы намерены реализовать целочисленный алгоритм, то условимся представлять *Z* рациональной дробью $Z = m/n$, где m, n - целые числа, причем $n > 0$.

Для задания вписанной в треугольник конической дуги достаточно задать положение любой ее точки, не совпадающей с началом и концом дуги. Этот факт доказывается в курсе аналитической геометрии. Для удобства и наглядности задания дуги примем, что вписанная в треугольник коническая дуга будет определяться положением точки пересечения дуги с медианой треугольника, как это показано на рис. 27. Положение этой точки $\vec{A}_C$ на медиане $\vec{A}_Q\vec{A}_1$ можно задать коэффициентом α : $\vec{A}_C = \vec{A}_Q(1-\alpha) + \vec{A}_1\alpha$. Из геометрии известно, что отношение трех точек является инвариантом аффинного преобразования, а так как формула

$$\vec{A}(\tau) = \vec{A}_0 + \vec{A}_{01}\frac{X(\tau)}{R} + \vec{A}_{12}\left(1 - \frac{Y(\tau)}{R}\right),$$

где $\vec{A}_{01} = \vec{A}_1 - \vec{A}_0$, $\vec{A}_{12} = \vec{A}_2 - \vec{A}_1$, задает аффинное преобразование 2D коники $X(\tau), Y(\tau)$ в многомерное пространство, то точка $\vec{A}_C$ пересечения кривой $\vec{A}(t)$ с медианой $\vec{A}_Q\vec{A}_1$ треугольника $\vec{A}_2\vec{A}_2\vec{A}_2$ однозначно задает эксцентриситет Z коники $X(\tau), Y(\tau)$.

Принимая во внимание симметрию коники $X(\tau), Y(\tau)$, выразим значение ее эксцентриситета Z через положение точки $\vec{A}_C$, задаваемой коэффициентом α,

$$\frac{m}{n} = \frac{\alpha^2 + 2\alpha - 1}{(1-\alpha)^2},$$

где $\vec{A}_C = \vec{A}_Q(1-\alpha) + \vec{A}_1\alpha$, $\vec{A}_Q = (\vec{A}_0 + \vec{A}_2)/2$, $Z = m/n$; $\alpha \in [0,1]$.

Из данной формулы видно, что вид конической дуги однозначно задается положением ее средней точки $\vec{A}_C$, в частности, для эллипса $\alpha \in (0,1/2)$, для параболы $\alpha = 1/2$, для гиперболы $\alpha \in (1/2,1)$.

3.2. Конический интерполятор

Для синтеза алгоритма растрирования квадратичной формы

$$nX^2 + nY^2 - 2mXY + 2mR(X+Y) - (2m+n)R^2 = 0$$

с 4-точечной схемой выбора шага можно применить метод средней точки. Полученный по этой методике алгоритм приведен в виде блок-схемы на рис.28. Инициализация данного алгоритма выполняется присвоением его переменным следующих начальных значений:

$u = 4n + 2m$,
$v = u - 4R(n+m)$,
$D = (n+m)(1-2R)$,
$k_1 = 4n$,
$k_2 = 4m$.

На рис. 28 приведена только динамическая часть алгоритма, которая при аппаратной реализации должна быть реализована в виде самостоятельного функционального элемента. Этот элемент в дальнейшем будем называть коническим интерполятором и обозначать CI. Конический интерполятор имеет один вход *T*, на который подаются единичные управляющие сигналы, и два выхода *X* и *Y*, с которых снимаются единичные управляющие сигналы.

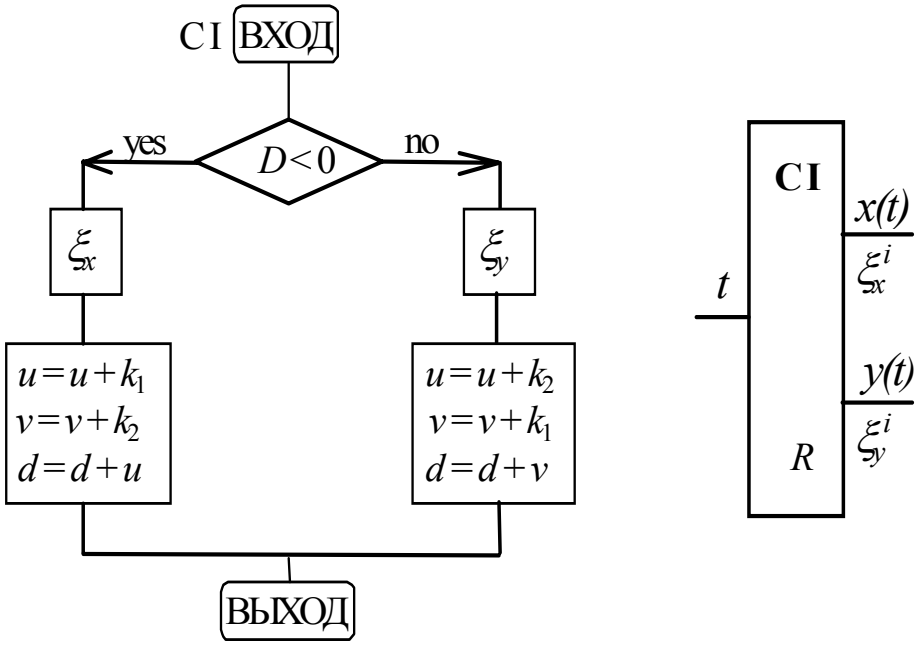


Рис. 28. Блок-схема и функциональная схема конического интерполятора

Если просуммировать все выходные сигналы X и Y по отдельности, то мы получим растровый образ вписанной в квадрат коники $X(t)$, $Y(t)$. Для генерации коники любого вида необходимо выполнить $2R$ итераций алгоритма, так как при выводе описываемого алгоритма использовалась 4-точечная схема выбора смежного пикселя. Этот момент принципиален для дальнейшего анализа.

Рассмотрим следующее уравнение:

$$\vec{A}(\tau) = \vec{A}_0 + \vec{A}_{01}\frac{X(\tau)}{R} + \vec{A}_{01}(1 - \frac{Y(\tau)}{R}).$$

Несложно заметить, что данное уравнение задает аффинное преобразование, которое преобразовывает конику частного положения $F(X,Y)=0$ в коническую дугу, вписанную в треугольник общего положения $A_0A_1A_2$.

При выполнении одной итерации алгоритма выходной сигнал снимается либо с выхода X, либо с выхода Y:

$$X_{i+1} = X_i + \xi_x{}^i, \quad Y_{i+1} = Y_i - \xi_y{}^i, \qquad \xi_x{}^i, \xi_y{}^i \in \{0,1\}.$$

В этом случае $\vec{A}(\tau)$ получит приращение

$$d\vec{A}_i = \vec{A}_{i+1} - \vec{A}_i = \vec{A}_{01}\frac{\xi_x^{\,i}}{R} + \vec{A}_{12}\frac{\xi_y^{\,i}}{R}.$$

Из последней формулы видно, что на каждой итерации функция $\vec{A}(\tau)$ получает фиксированное приращение, равное $\frac{1}{R}\vec{A}_{01}$, если выходной сигнал равен $\xi_x^{\,i}$, либо $\frac{1}{R}\vec{A}_{12}$, если выходной сигнал равен $\xi_y^{\,i}$.

Ранее было доказано, что линейный интерполятор, независимо от того, какой алгоритм (Брезенхема или ЦДА) он реализует, генерирует вектор с постоянной скоростью, то есть на каждой итерации вектор получает фиксированное приращение. Из этого следует, что алгоритм генерации кривой $\vec{A}(\tau)$ можно представить в виде рис. 29.

$$d\vec{A}(t) = \vec{A}_{01}x(t) + \vec{A}_{12}y(t)$$

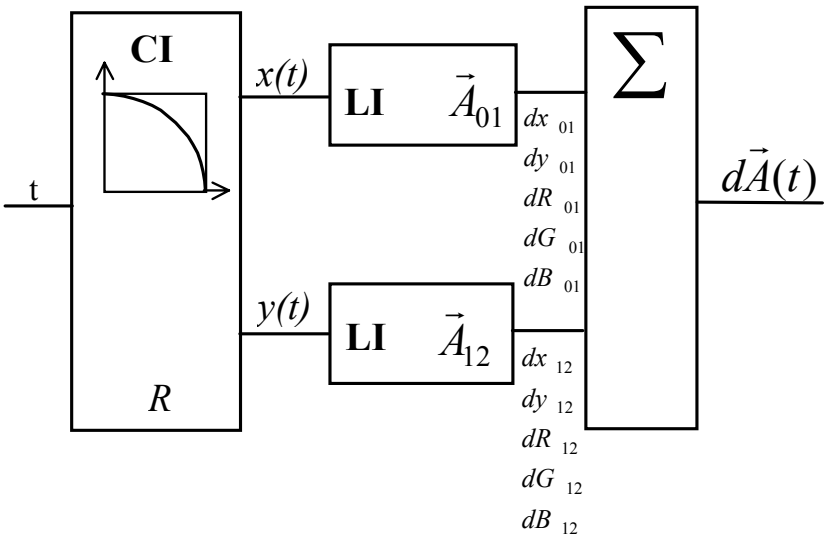


Рис. 29. Функциональная схема генератора конических дуг

Для дальнейшего анализа будет удобно описать работу конического интерполятора нормализированными функциями:

$$x(t) = \frac{X(t)}{R}, \qquad x(t) \in [0,1],$$

$$y(t) = 1 - \frac{Y(t)}{R}, \qquad y(t) \in [0,1],$$
$$t = \frac{i}{2R}, \qquad t \in [0,1],$$

где i - порядковый номер текущей итерации алгоритма $i \in [0,2R]$,

$$R = \max\{\|\vec{A}_{01}\|, \|\vec{A}_{12}\|\}, \qquad \|d\vec{A}\| = \max\{|dX|, |dY|, |dR|, |dG|, |dB|\}.$$

Уравнение управляющей коники принимает вид

$$x^2 + y^2 - 2Zxy - 2y(1+Z) = 0.$$

Ранее было доказано, что на каждой итерации генерируется выходной сигнал лишь из одного из выходов CI. Математически этот факт можно представить следующим образом:

$$t = \frac{x(t) + y(t)}{2}.$$

Из последних двух уравнений можно получить явные выражения для выходных сигналов $x(t), y(t)$:

$$x(t) = t + \sigma + sign(\sigma)\sqrt{\sigma(\sigma + 2t(1-t))},$$
$$y(t) = t - \sigma + sign(\sigma)\sqrt{\sigma(\sigma + 2t(1-t))},$$

где $\sigma = \frac{1}{2}\left(\frac{1+Z}{1-Z}\right)$ при $Z \neq 1$. Если $Z = 1$ (это - случай параболы), то выражения для $x(t), y(t)$ значительно упрощаются:

$$x(t) = 2t - t^2, \qquad y(t) = t^2.$$

Теперь уравнение конической дуги, вписанной в треугольник общего положения, можно задать следующей формулой:

$$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}x(t) + \vec{A}_{12}y(t).$$

На рис. 30 приведено семейство конических дуг, вписанных в треугольник. Нижние дуги ($\alpha = \{1/9, 2/9, 3/9, 4/9\}$) являются эллип-

тическими. Верхние дуги ($\alpha = \{5/9,6/9,7/9,8/9\}$) являются гиперболическими.

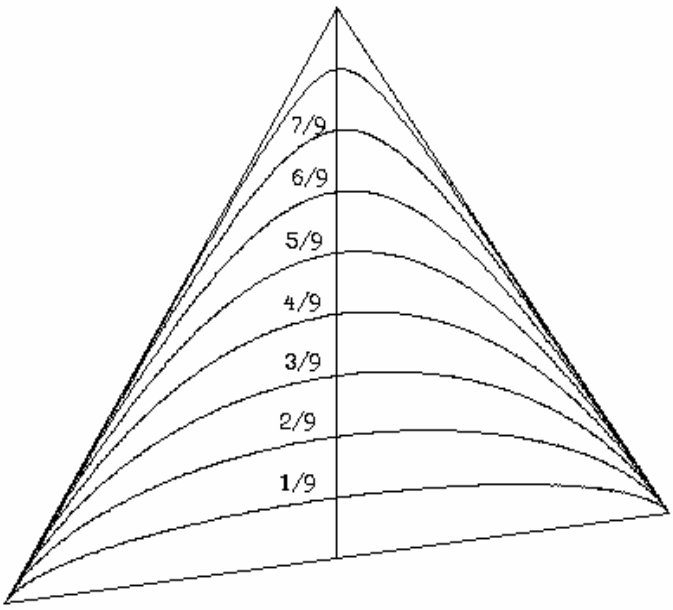


Рис. 30. Семейство конических дуг, вписанных в треугольник общего положения

3.3. Расщепление конической дуги пополам

При отображении кривых возникает ряд проблем, многие из которых решаются проще, если кривая представляет собой короткий незначительно изогнутый фрагмент, близкий по своим геометрическим свойствам к отрезку прямой. Для представления конической дуги последовательностью конических сегментов необходимо расщепить исходную кривую пополам и, если необходимо, повторить эту операцию над каждым сегментом требуемое число раз.

При каждом расщеплении дуги ее тип остается неизменным, однако эксцентриситет коники приближается к единице, то есть сегменты коники не только укорачиваются, но и приближаются к параболам (строго говоря, не к параболам а к отрезкам прямых – векторам, но вектор также может быть математически описан параметрическим уравнением параболы, что в данном случае и имеет место). Докажем эти утверждения.

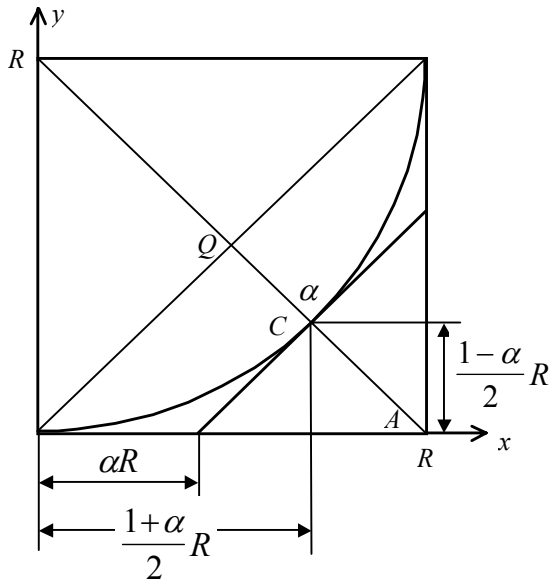


Рис. 31. Расщепление коники, вписанной в квадрат со стороной *R*

Рассмотрим рис. 31. Поскольку при аффинном преобразовании коники отношение трех точек не изменяется, все соотношения, полученные для данного частного случая, останутся справедливыми для конической дуги общего положения.

Исходная коника задается тремя точками $\{(0,0),(R,0),(R,R)\}$ и имеет эксцентриситет Z, определяемый значением коэффициента α, определяющего отношение трех точек Q, C, A. После расщепления образуются две одинаковые дуги, поэтому мы рассмотрим только первую, а все доказанные свойства автоматически будут справедливы и для второго сегмента. Вторичная коника будет задаваться тремя точками

$$\{(0,0),(\alpha R,0),(\frac{1+\alpha}{2}R,\frac{1-\alpha}{2}R)\}$$

Этот факт следует из очевидных геометрических соображений. Нашей целью является определение эксцентриситета вторичной кривой. Поскольку исходная и вторичная дуги являются кониками, аналитически их можно представить как конические дуги, вписанные в треугольники общего положения, координаты которых нам известны:

$$\vec{A}(t)=\begin{pmatrix}0\\0\end{pmatrix}+R\begin{pmatrix}1\\0\end{pmatrix}x(t)+R\begin{pmatrix}0\\1\end{pmatrix}y(t)$$ - исходная коника,

$$\hat{A}(t) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + R\begin{pmatrix} \alpha \\ 0 \end{pmatrix}\hat{x}(t) + R\frac{(1-\alpha)}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix}\hat{y}(t)$$ - вторичная коника.

Поскольку эти различные параметрические функции задают одну и ту же геометрическую линию $\vec{A}(t/2) \equiv \hat{A}(t)$, то мы можем найти соотношения между управляющими кониками исходной и вторичной дуг:

$$x = \alpha\hat{x} + \frac{1-\alpha}{2}\hat{y}, \qquad y = \frac{1-\alpha}{2}\hat{y},$$
$$\hat{x} = \frac{x-y}{\alpha}, \qquad \hat{y} = \frac{2y}{1-\alpha}.$$

Для обоих управляющих коник справедливо также их алгебраическое представление:

$$F(x,y) = x^2 + y^2 + 2Zxy - 2y(1+Z) = 0,$$

$$F(\hat{x},\hat{y}) = \hat{x}^2 + \hat{y}^2 + 2\hat{Z}\hat{x}\hat{y} - 2\hat{y}(1+\hat{Z}) = 0.$$

Подставив выражения $\hat{x}(x,y), \hat{y}(x,y)$ в квадратичную форму $F(\hat{x},\hat{y})$, мы находим следующие соотношения:

$$\hat{Z} = \sqrt{\frac{1+Z}{2}} = \frac{\alpha}{1-\alpha},$$

$$\hat{\alpha} = \frac{1}{1+\sqrt{2(1-\alpha)}}.$$

Получение этих соотношений и было целью нашего анализа. Отметим еще раз, что коэффициент $\hat{\alpha}$, рассчитанный по приведенной выше формуле, будет корректным для любой конической дуги, вписанной в треугольник общего положения. Итеративно вычисляя $\hat{\alpha}$, мы можем видеть, что он стремится к единице. Отсюда можно сделать вывод, что по мере расщепления сегменты любой конической дуги (эллиптической или гиперболической) приближаются к параболе.

Найдем явное представление расщепленной конической дуги:

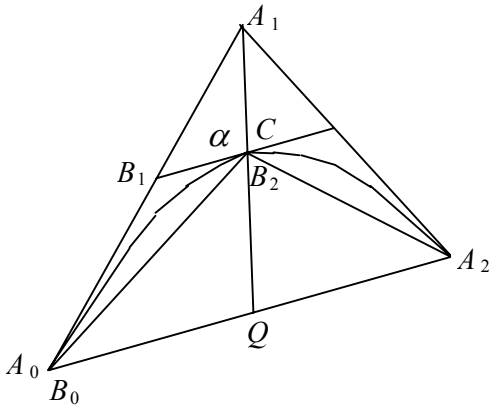


Рис. 32. Расщепление конической дуги, вписанной в треугольник общего положения

$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}x(t) + \vec{A}_{12}y(t)$ - исходная дуга,

$\vec{B}(t) = \vec{B}_0 + \vec{B}_{01}\hat{x}(t) + \vec{B}_{12}\hat{y}(t)$ - вторичная дуга (1/2 исходной дуги).

Пересчет параметров расщепленной дуги выполняется по формулам:

$$B_0 = A_0,$$
$$B_1 = A_0(1-\alpha) + A_1\alpha,$$
$$B_2 = Q(1-\alpha) + A_1\alpha.$$

$$\hat{\alpha} = \frac{1}{1+\sqrt{2(1-\alpha)}} \quad \rightarrow \quad \hat{Z} = \sqrt{\frac{1+Z}{2}} = \frac{\alpha}{1-\alpha};$$

Отметим, что при каждом расщеплении дуги пополам угол между касательными в начале и конце дуги уменьшается вдвое, что приводит к уменьшению нерегулярности ступенчатого эффекта и улучшает качество представления конической дуги на экране растрового дисплея.

3.4. Расщепление конической дуги в произвольной точке

Для расщепления лекальной кривой пополам необходимо уметь расщеплять коническую дугу в произвольной точке, задаваемой параметром t. Эта задача поясняется рис. 33.

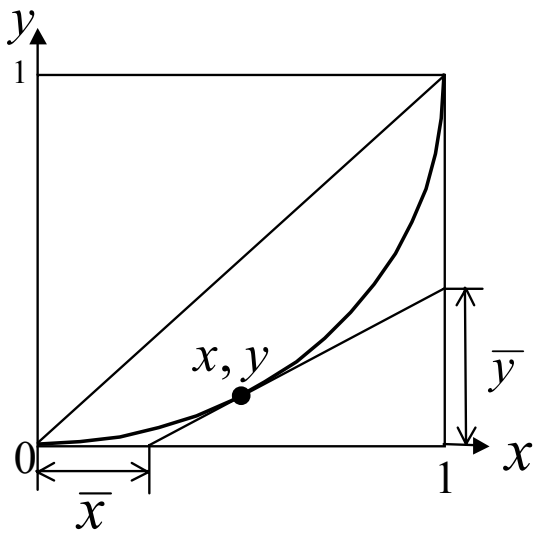


Рис. 33. Коническая дуга, вписанная в квадрат со стороной *R*=1

Коническая дуга, вписанная в единичный квадрат, задается уравнением

$$x^2 + y^2 + 2Zxy - 2y(1+Z) = 0\,.$$

Дифференцируя это уравнение по переменной x, находим производную $\dfrac{dy}{dx}$, равную тангенсу угла наклона коники в точке (x,y):

$$\frac{dy}{dx} = \frac{x+Zy}{(1+Z)-(y+Zx)} = \frac{\bar{y}}{1-\bar{x}}.$$

Зная значения $\bar{x}$ и $\bar{y}$, составим уравнение прямой линии

$$\frac{y}{\bar{y}} = \frac{x-\bar{x}}{1-\bar{x}}.$$

Решая систему из этих двух уравнений, находим выражения для $\bar{x}$ и $\bar{y}$:

$$\bar{x} = \frac{y(1+Z)}{x+Zy};$$

$$\bar{y} = \frac{x - y}{(1 + Z) - (y + Zx)} = \frac{2y}{x + y} = \frac{y}{t}.$$

Пара $(\bar{x}, \bar{y})$ однозначно задает положение точки (x, y) коники и может рассматриваться как координаты точки коники. Параметр t можно выразить через координаты точки:

$$t = \frac{1}{2}(x + y) = \frac{\bar{x}}{1 + \bar{x} - \bar{y}}.$$

Эксцентриситет Z является инвариантом коники, и для любой точки дуги справедливо соотношение

$$Z = \frac{1}{2}\frac{x^2 + y^2 - 2y}{y(1 - x)} = \frac{\bar{x}\bar{y} + \bar{y} - 2\bar{x}}{\bar{x}\bar{y} - \bar{y}}.$$

Теперь запишем уравнение коники в диапазоне $[0, t]$, считая, что координаты конечной точки (x_t, y_t) и $(\bar{x}_t, \bar{y}_t)$ нам известны:

$$x = \bar{x}\hat{x} + (x_t - \bar{x})\hat{y}, \qquad y = y_t\hat{y}.$$

Отсюда получаем

$$\hat{x} = \frac{x}{x_t} - \frac{x_t - \bar{x}_t}{\bar{x}_t} \cdot \frac{y}{y_t}, \qquad \hat{y} = \frac{y}{y_t},$$

где (x, y, Z) - исходная коника, $(\hat{x}, \hat{y}, \hat{Z})$ - сегмент исходной коники. (x_t, y_t) и $(\bar{x}_t, \bar{y}_t)$ - координаты конечной точки сегмента исходной коники.

Принимая во внимание, что алгебраическая форма уравнения коники не зависит от того, какой сегмент дуги мы рассматриваем, запишем уравнение коники $(\hat{x}, \hat{y}, \hat{Z})$

$$\hat{x}^2\bar{x}_t^{\,2} + 2\bar{x}(x_t - \bar{x}_t)\hat{x}\hat{y} + (x_t - \bar{x}_t)^2\hat{y}^2 + 2Z\bar{x}_t y_t\hat{x}\hat{y} + 2Zy_t(x_t - \bar{x}_t)\hat{y}^2 - 2(1 + Z)y_t\hat{y} = 0,$$

и найдем из него эксцентриситет $\hat{Z}$ сегмента дуги:

$$\hat{Z} = (1 + Z)\frac{y_t}{\bar{x}_t^{\,2}} - 1.$$

Вот и все. Нахождение $\hat{Z}$ и являлось целью проведенного анализа. Теперь мы можем любую коническую дугу (x, y, Z) представить в виде пары сопряженных конических дуг $(\hat{x}, \hat{y}, \hat{Z})$ и $(\bar{x}, \bar{y}, \hat{Z})$, где $\bar{Z}$ - эксцентриситет второго сегмента.

$$\bar{Z} = (1 + Z)\frac{1 - x_t}{(1 - \bar{y}_t)^2} - 1.$$

3.5. Использование алгоритма расщепления для расчета положения текущей точки конической дуги

Покажем, что алгоритм, близкий к известному алгоритму Кастелью, может быть использован для вычисления положения текущей точки дуги. Отметим сразу, что рассматриваемый алгоритм имеет регулярную структуру и может быть составлен из однотипных элементов, называемых PE (processing element).

Применение алгоритма эффективно при условии, что PE реализован на аппаратном уровне в виде СБИС. Программная реализация PE всегда возможна, но нецелесообразна, поскольку уже рассмотренный нами конический интерполятор, даже при программной реализации, будет иметь большее быстродействие, чем PE. С другой стороны, далее будет показано, что на основе PE можно построить генератор кривых более высокого порядка, чем коники, например кривые Безье.

Рассмотрим алгоритм, реализующий вычисление значения точки дуги, заданной параметром t:

```
for k:=1 to 2 do
     for j:=k to 2 do
          A^(k) = (1 - t_(j-1)^(k-1)) A_(j-1)^(k-1) + t_(j-1)^(k-1) A_j^(k-1);
     end j;
end k;
```

Этот алгоритм имеет простую геометрическую интерпретацию, которая приведена на рис. 34.

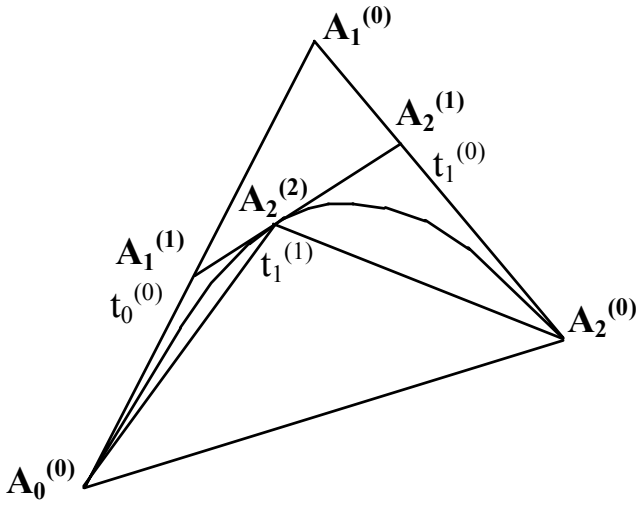


Рис. 34. Расщепление конической дуги методом Кастелью

Развернем приведенный алгоритм в последовательность операций.

$$\vec{A}_1^{(1)} = \vec{A}_0^{(0)}(1-t_0^{(0)}) + \vec{A}_1^{(0)}t_0^{(0)},$$
$$\vec{A}_2^{(1)} = \vec{A}_1^{(0)}(1-t_1^{(0)}) + \vec{A}_2^{(0)}t_1^{(0)},$$
$$\vec{A}_2^{(2)} = \vec{A}_1^{(1)}(1-t_1^{(1)}) + \vec{A}_2^{(1)}t_1^{(1)}.$$

Представим уравнение конической дуги $\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}x(t) + \vec{A}_{12}y(t)$ в следующем виде:

$$\vec{A}(t) = \vec{A}_0(1-x(t)) + \vec{A}_1(x(t)-y(t)) + \vec{A}_2y(t).$$

Тогда, принимая во внимание, что согласно приведенному выше алгоритму положение точки дуги, заданной значением параметра t, можно найти по формуле $\vec{A}_2^{(2)}$, получаем тождество

$$\vec{A}(t) \equiv \vec{A}_0^{(0)}(1-t_0^{(0)})(1-t_1^{(1)}) + \vec{A}_1^{(0)}\left[t_0^{(0)}(1-t_1^{(1)}) + (1-t_1^{(0)})t_1^{(1)}\right] + \vec{A}_2^{(0)}t_1^{(0)}t_1^{(1)};$$

Приравнивая коэффициенты при одинаковых вершинах, находим значения коэффициентов $t_0^{(0)}$, $t_1^{(0)}$, $t_1^{(1)}$:

$$t_0^{(0)} = \bar{x}(t), \qquad t_1^{(0)} = \bar{y}(t), \qquad t_1^{(1)} = t.$$

Чтобы доказать справедливость последних соотношений, следует подставить их в выражение $\vec{A}(t)$ и приравнять коэффициенты при одинаковых вершинах. При этом мы получаем систему тождеств

$$\begin{cases} 1-x=(1-\bar{x})(1-t), \\ x-y=\bar{x}(1-t)+(1-\bar{y})t, \\ y=\bar{y}t, \end{cases}$$

доказательство которых элементарно и является доказательством справедливости полученных коэффициентов $t_0^{(0)}$, $t_1^{(0)}$, $t_1^{(1)}$.

Приведем на рис. 35 вычислительную структуру, которая реализует предложенный выше алгоритм.

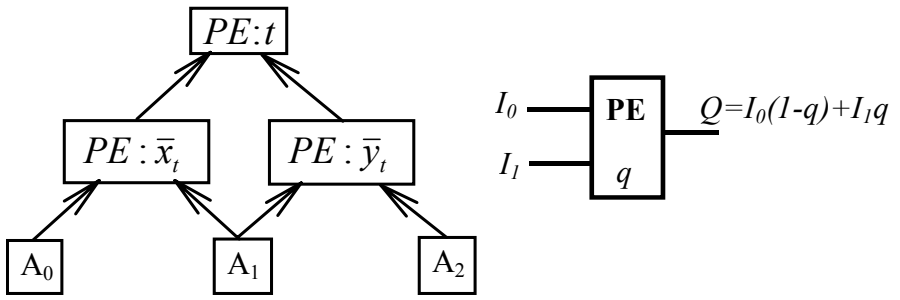


Рис. 35. Функциональная схема генератора конических дуг, построенного на основе алгоритма Кастелью (предполагается, что вычислительный элемент *PE* подлежит аппаратной реализации)

Каждый <PE : q> - элемент имеет два входа (I_0, I_1) и один выход (Q) и реализует операцию $Q=I_0(1-q)+I_1q$. Параметры $t, \bar{x}_t, \bar{y}_t$ можно рассчитать, используя конический интерполятор.

3.6. Расщепление конической дуги

Теперь, принимая во внимание инвариантность отношения трех точек аффинного преобразования, можно привести полный алгоритм расщепления конической дуги общего положения.

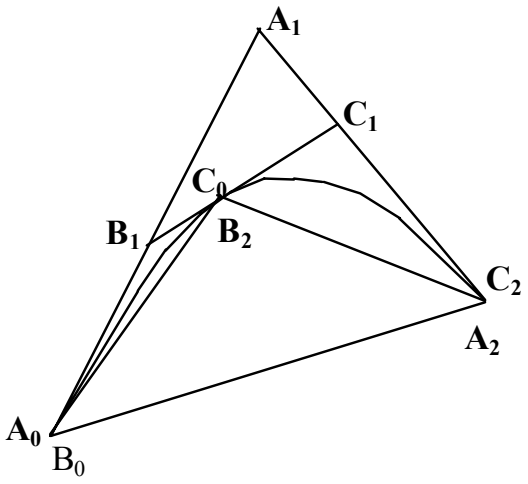


Рис. 36. Расщепление конической дуги

На рисунке обозначено:

$<A_0,A_1,A_2, Z>$ - исходная коническая дуга, $<B_0,B_1,B_2,Z_B>$ - вторичная коническая дуга, $<C_0,C_1,C_2,Z_C>$ - вторичная коническая дуга, параметр t - задает положение точки расщепления дуги $<A_0,A_1,A_2, Z>$.

$$\vec{B}_0 = \vec{A}_0, \qquad \vec{C}_2 = \vec{A}_2,$$

$$\vec{B}_1 = \vec{A}_0(1-\bar{x}_t) + \vec{A}_1\bar{x}_t, \qquad \vec{C}_1 = \vec{A}_1(1-\bar{y}_t) + \vec{A}_2\bar{y}_t,$$

$$Z_B = (1+Z)\frac{y_t}{\bar{x}_t^{\,2}} - 1, \qquad Z_C = (1+Z)\frac{1-x_t}{(1-\bar{y}_t)^2} - 1,$$

$$\vec{B}_2 = \vec{C}_0 = \vec{B}_1(1-t) + \vec{C}_1 t.$$

Докажем справедливость этих соотношений. Из общего уравнения конической дуги получаем два уравнения для исходной и расщепленной дуг, соответственно:

$$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}x + \vec{A}_{12}y,$$

$$\vec{B}(\tau) = \vec{B}_0 + \vec{B}_{01}\hat{x} + \vec{B}_{12}\hat{y}.$$

Подставляя выражение x, y через $\hat{x}, \hat{y}$ в уравнение исходной дуги, получаем

$$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}\left[\bar{x}_t\hat{x} + (x_t - \bar{x}_t)\hat{y}\right] + \vec{A}_{12}y_t\hat{y}.$$

Поскольку исходная и вторичная дуга на начальном сегменте расщепления описывают одну и ту же линию, то из условия $\vec{A}(t) \equiv \vec{B}(t)$ получаем связь между параметрами дуг:

$$\vec{B}(\tau) = \vec{B}_0 + \vec{A}_{01}\bar{x}_t\hat{x} + \left[\vec{A}_{01}(x_t - \bar{x}_t) + \vec{A}_{12}y_t\right]\hat{y},$$
$$\vec{B}_{01} = \vec{A}_{01}\bar{x}_t,$$
$$\vec{B}_{12} = \vec{A}_{01}(x_t - \bar{x}_t) + \vec{A}_{12}y_t.$$

Из последних трех выражений вытекают соотношения

$$\vec{B}_0 = \vec{A}_0,$$
$$\vec{B}_1 = \vec{A}_0(1 - \bar{x}_t) + \vec{A}_1\bar{x}_t,$$
$$\vec{C}_1 = \vec{A}_1(1 - \bar{y}_t) + \vec{A}_2\bar{y}_t,$$
$$\vec{B}_2 = \vec{C}_0 = \vec{B}_1(1-t) + \vec{C}_1 t.$$
$$Z_B = (1+Z)\frac{y_t}{\bar{x}_t^{\,2}} - 1.$$

3.7. Коническая дуга как рациональная кривая Безье

Из математики известно, что рациональная кривая Безье второго порядка задает коническую дугу. Найдем связь между управляющими параметрами формы Безье $\alpha_0, \alpha_1, \alpha_2$ и параметрами рассмотренной выше формы задания коники как конической дуги, вписанной в треугольник общего положения и подлежащей растрированию рассмотренным выше методом.

Рациональная кривая Безье второго порядка задается формулой

$$\vec{A}(t) = \frac{\vec{P}_0\alpha_0(1-t)^2 + 2\vec{P}_1\alpha_1(1-t)t + \vec{P}_2\alpha_2 t^2}{\alpha_0(1-t)^2 + 2\alpha_1(1-t)t + \alpha_2 t^2},$$

где $\vec{P}_0, \vec{P}_1, \vec{P}_2$ - вершины треугольника, $\alpha_0, \alpha_1, \alpha_2$ - скалярные параметры кривой Безье (они должны быть неотрицательны и не равны нулю все одновременно). Представим $\vec{A}(t)$ в форме

$$\vec{A}(t) = \vec{P}_0 + \vec{P}_{01}x(t) + \vec{P}_{12}y(t),$$

где

$$x(t) = \frac{2\alpha_1(1-t)t + \alpha_2 t^2}{\alpha_0(1-t)^2 + 2\alpha_1(1-t)t + \alpha_2 t^2},$$

$$y(t) = \frac{\alpha_2 t^2}{\alpha_0(1-t)^2 + 2\alpha_1(1-t)t + \alpha_2 t^2}.$$

Из проведенного выше анализа следует, что для любой точки коники должен иметь место инвариант

$$2(1+Z) = \frac{(x-y)^2}{y(1-x)}.$$

Подставляя $x(t)$, $y(t)$ в это выражение, получаем выражение эксцентриситета Z через параметры $\alpha_0, \alpha_1, \alpha_2$ рациональной кривой Безье второго порядка:

$$Z = \frac{2\alpha_1^2}{\alpha_0\alpha_2} - 1,$$

что и требовалось найти.

Представляет интерес и обратная задача: имея эксцентриситет Z, найти набор коэффициентов $\alpha_0, \alpha_1, \alpha_2$ рациональной кривой Безье второго порядка. Отметим, что набор коэффициентов $\alpha_0, \alpha_1, \alpha_2$ можно умножить на любое ненулевое число. Следовательно, коэффициенты можно нормировать таким образом, чтобы выполнялось условие $\alpha_0 + 2\alpha_1 + \alpha_2 = 1$. Из этого ограничения следует оценка $\alpha_1 \le 1/2$. Задаваясь $\alpha_1 \in [0, 1/2]$ и решая систему уравнений

$$\begin{cases} Z = \dfrac{2\alpha_1^2}{\alpha_0\alpha_2} - 1, \\ \alpha_0 + 2\alpha_1 + \alpha_2 = 1, \end{cases}$$

находим выражения α_0, α_2 через Z и α_1. Это решение является общим, но можно найти важное частное решение. Пусть $\alpha_1 = \alpha/2$, где α задает положение точки пересечения медианы треугольника с ко-

нической дугой. Как было показано выше, Z и α связаны следующим соотношением:

$$Z = \frac{\alpha^2 + 2\alpha - 1}{(1-\alpha)^2}.$$

В этом случае α_0, α_2 имеют следующие простые выражения: $\alpha_0 = \alpha_2 = \frac{1}{2}(1-\alpha)$. Коническая дуга (рис. 37) будет задаваться уравнением

$$\vec{A}(t) = \frac{\vec{P}_0(1-\alpha)(1-t)^2 + 2\vec{P}_1\alpha(1-t)t + \vec{P}_2(1-\alpha)t^2}{(1-\alpha)(1-t)^2 + 2\alpha(1-t)t + (1-\alpha)t^2}.$$

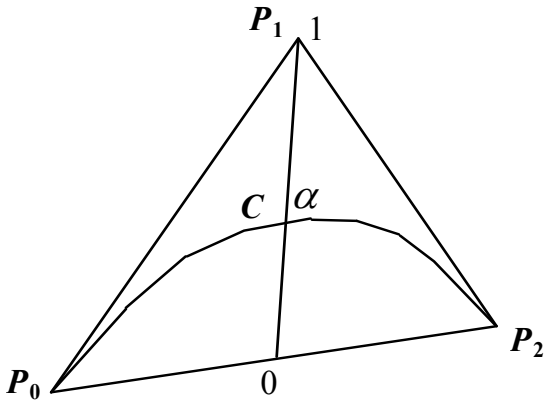


Рис. 37. Задание конической дуги как рациональной кривой Безье 2-го порядка

3.8. Представление конического сплайна набором конических дуг

Конический сплайн является частным случаем рационального B-сплайна степени 2, и его сегмент может быть представлен следующим образом:

$$\vec{C}_i(t) = \frac{\sum_{k=-1}^{1} h_{i+k} B_k(t) \vec{V}_{i+k}}{\sum_{k=-1}^{1} h_{i+k} B_k(t)}, \qquad t \subseteq [0,1],$$

где $\vec{C}_i(t)$ - i-тый сегмент сплайна, $< \vec{V}_i >$-управляющие вершины сплайна, $< h_i >$ - узловые коэффициенты сплайна, $B_k(t)$ - базовый полином B-сплайна,

$$B_{-1}(t) = \frac{1}{2}(1+t)^2, \qquad B_0(t) = \frac{1}{2}\left(-2t^2 + 2t + 1\right), \qquad B_1(t) = \frac{1}{2}t^2.$$

Представим сегмент сплайна в форме рациональной кривой Безье второго порядка:

$$\vec{C}(t) = \frac{w_0 \vec{K}_0 (1-t)^2 + 2w_1 \vec{K}_1 (1-t)t + w_2 \vec{K}_2}{w_0 (1-t)^2 + 2w_1 (1-t)t + w_2 t^2}, \qquad t \subseteq [0,1].$$

Приравнивая коэффициенты при одинаковых степенях *t* обоих форм описания сегмента, находим:

$$\vec{K}_0 = \frac{\vec{V}_{i-1} h_{i-1} + \vec{V}_i h_i}{h_{i-1} + h_i},$$

$$\vec{K}_1 = \vec{V}_i,$$

$$\vec{K}_2 = \frac{\vec{V}_{i+1} h_{i+1} + \vec{V}_i h_i}{h_i + h_{i+1}},$$

$$w_0 = \frac{h_i + h_{i-1}}{h_{i-1} + 6h_i + h_{i+1}},$$

$$w_1 = \frac{2h_i}{h_{i-1} + 4h_i + h_{i+1}},$$

$$w_2 = \frac{h_i + h_{i+1}}{h_{i-1} + 4h_i + h_{i+1}}.$$

На рис. 38 приведен пример сегмента конического сплайна. Этот сегмент является конической дугой, вписанной в треугольник K_0, K_1, K_2, положение вершин которого мы только что рассчитали.

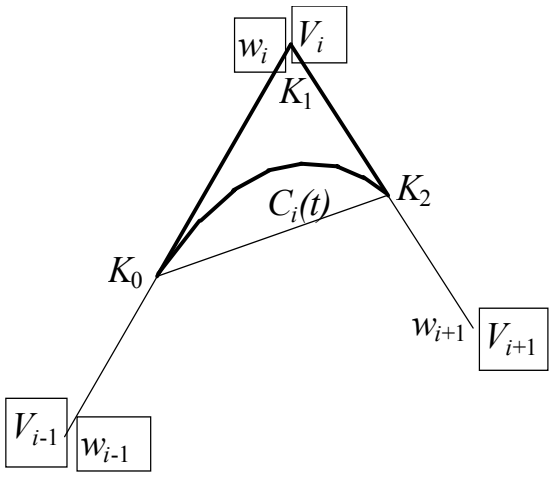


Рис. 38. Коническая дуга – сегмент конического сплайна

Ранее было доказано, что лекальная дуга может быть задана уравнением

$$\vec{C}(t) = \vec{K}_0 + \vec{K}_{01}x(t) + \vec{K}_{12}y(t).$$

Отметим, что параметр t в последнем уравнении не эквивалентен этому параметру в предшествующих формулах, однако алгебраическая форма обоих уравнений остается одной и той же:

для сегмента сплайна

$$x^2 + y^2 + 2xy\frac{2w_1^2 - w_0w_2}{w_0w_2} + \frac{4w_1^2}{w_0w_2}y = 0,$$

а для конической дуги

$$x^2 + y^2 + 2Zxy - 2y(1+Z) = 0.$$

Приравнивая коэффициенты при одинаковых степенях x и y, находим

$$Z = \frac{2w_1^2 - w_0w_2}{w_0w_2}.$$

Теперь мы можем применить генератор конических дуг для растрирования сегмента сплайна, а следовательно, и самого сплайна.

4. Интерполяционные структуры и лекальные кривые

4.1. Структура лекального интерполятора

Мы показали, что работа конического интерполятора описывается параметрическими функциями $x(t), y(t)$, причем параметр t линейно зависит от текущего количества входных инициирующих импульсов (итераций алгоритма). Из этого следует, что если любой выход CI соединить с входом другого CI, то с выходов этого второго интерполятора также будут сниматься единичные управляющие сигналы, и эти сигналы будут являться суперпозицией входного сигнала как функции параметра t. Приведем вычислительную структуру, составленную из конических интерполяторов.

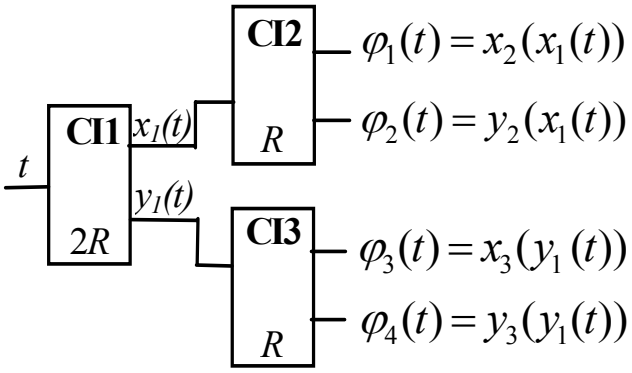


Рис. 39. Вычислительная структура, составленная из конических интерполяторов

Кривая, построенная на базе такой или подобной ей структуры, называется лекальной кривой (L-curve) и математически задается формулой

$$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}\varphi_1(t) + \vec{A}_{12}\varphi_2(t) + \vec{A}_{23}\varphi_3(t) + \vec{A}_{34}\varphi_4(t).$$

Функциональная схема генератора лекальных вершин в этом случае, будет иметь вид, показанный на рис. 40.

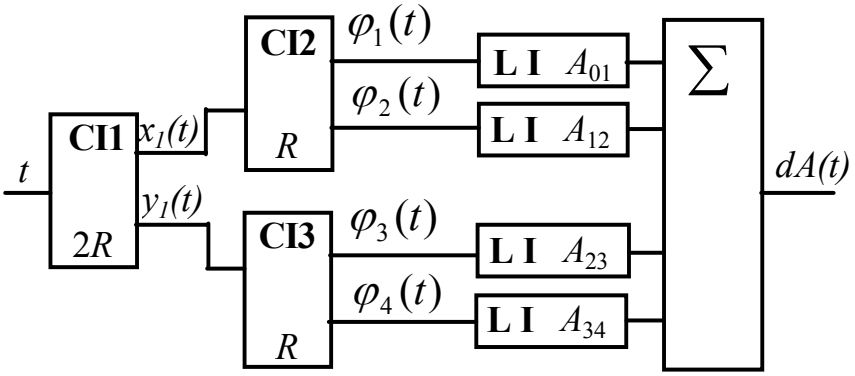


Рис. 40. Функциональная схема генератора лекальных кривых

Лекальная кривая соединяет начальную и конечную точки разомкнутого пятиугольника. В начальной и конечной точках этого многоугольника лекальная кривая имеет касательную параллельную смежному ребру (A_{01} и A_{34}**)** многоугольника. Если многоугольник выпуклый, то кривая будет лежать внутри многоугольника. Пример построения лекальных кривых приведен на рис. 41. Данное семейство кривых получено при условии $\alpha_1 = \alpha_2 = \alpha_3 = \{0.1,...,0.8\}$. Каждый коэффициент α_i определяет режим работы одного конического интерполятора.

Свойства лекальной кривой близки к свойствам кривой Безье, однако в отличие от кривой Безье, поведение которой полностью определяется положением ее вершин, лекальная кривая, кроме вершин, имеет еще три управляющих параметра (эксцентриситеты Z_1,Z_2,Z_3 или коэффициенты $\alpha_1,\alpha_2,\alpha_3$), которые управляют степенью близости линии к задающему многоугольнику и формой кривой.

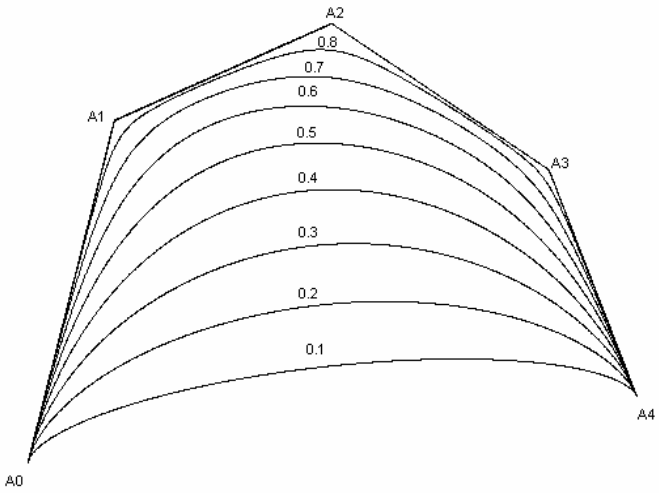


Рис. 41. Семейство лекальных кривых, вписанных в разомкнутый многоугольник

Из рис. 41 видно, что, если все три эксцентриситета близки к единице (гиперболический режим), то лекальная кривая прижимается к ребрам многоугольника. При выборе эксцентриситетов близкими к -1 (эллиптический режим) кривая прижимается к ребру, соединяющему начальную и конечную точки многоугольника. В остальных случаях лекальная кривая, управляемая параметрами Z_1,Z_2,Z_3, гибко изменяет свою форму.

Предложенная методика построения генераторов лекальных кривых из набора конических интерполяторов позволяет организовать множество вычислительных структур для генерации различных кривых, вписанных в многоугольник. Несколько таких структур приведено на рис. 42.

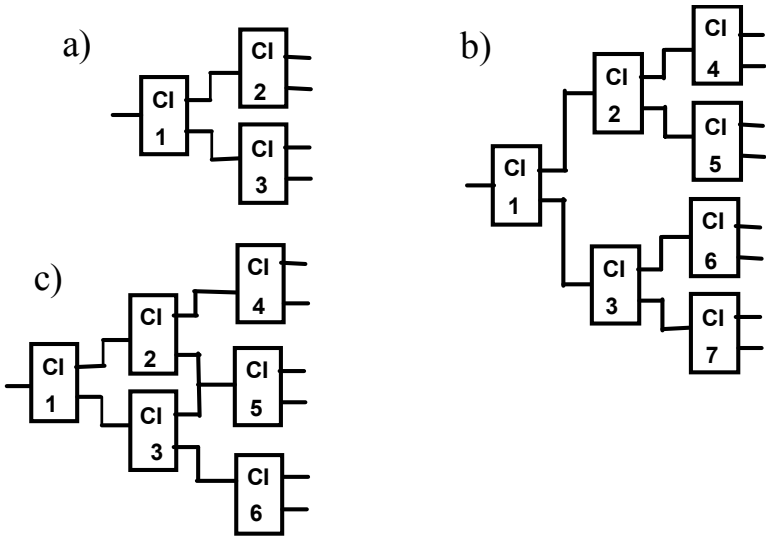


Рис. 42. Примеры управляющих схем генератора лекальных кривых

К выходам интерполяционной структуры должны подключаться входы генераторов векторов, а генерируемая лекальная кривая будет вписана в разомкнутый многоугольник, число ребер которого равно числу выходов управляющей интерполяционной структуры. Условимся в дальнейшем называть подобные вычислительные структуры лекальными интерполяторами LI.

Генерация лекальной кривой, таким образом, выполняется посредством квази-одновременной параллельной генерации набора векторов - ребер многоугольника. Мы говорим о квази-одновременности, поскольку в каждый момент времени, соответствующий одному единичному импульсу на входе лекального интерполятора, снимается сигнал лишь с одного выхода интерполяционной структуры и выполняется приращение вдоль лишь одного вектора. Рас-

пределенные вычисления, реализуемые лекальным интерполятором, даже при его аппаратной реализации по сути дела являются вычислениями с разделением времени. Нетрудно заметить, что CI второго уровня (CI2, CI3) простаивают 50% времени работы LI**,** CI третьего уровня (CI5, ..., CI7) простаивают 75% времени каждый, и так далее по мере усложнения структуры LI. Из этого наблюдения следует, что слишком усложнять структуру генератора лекальных кривых нецелесообразно.

Оценим время генерации конической кривой предложенным методом. Из иерархической структуры LI видно, что интерполяторы нижнего уровня должны получать на вход вдвое меньшее число входных импульсов, чем управляющий ими CI. Однако, очевидно, что конические интерполяторы низшего уровня, управляющие генерацией векторов, должны получить на вход, как минимум, столько сигналов, какова длина генерируемого ими вектора. Число входных и выходных сигналов интерполятора жестко связано с размером R квадрата, в который вписана управляющая коника. Найдем эту величину из условия

$$R_0 = \max\{\|\vec{A}_{01}\|, \|\vec{A}_{12}\|, \|\vec{A}_{23}\|, \|\vec{A}_{34}\|, \ldots\},$$

где $\|\vec{A}\| = \max(|A_x|, |A_y|, |A_z|, \ldots)$.

Таким образом, CI нижнего уровня имеет $R_1=R_0$, CI следующего уровня $R_2=R_3=2R_0$, затем $R_4=4R_0$, и так далее. Другими словами, переход на каждый следующий уровень требует удвоения величины R и, следовательно, удвоения времени работы LI. Для схемы рис. 42, a $R_1=2R_0$. Для схемы рис.42, b $R_1=4R_0$.

4.2. Нерегулярный характер ступенчатого эффекта

Попеременный характер генерации векторов имеет еще один недостаток. Мы знаем, что вектор, окружность, эллипс при дискретизации выглядят как набор вертикальных и горизонтальных ступенек. Этот эффект называется ступенчатым (лестничным) эффектом, и он, строго говоря, неустраним. Однако лестничный эффект для этих кривых носит регулярный, симметричный характер и «не режет глаз». В случае генератора лекальных кривых ситуация иная. Каждый вектор вносит свой вклад в создание лестничного эффекта. Так как векторы генерируются попеременно, то и лестничный эффект,

являясь суммарным эффектом от нескольких векторов, усиливается в число раз, равное количеству векторов, и имеет характерный нерегулярный характер. Если не принимать специальных мер, которые мы обсудим ниже, то лекальная кривая будет выглядеть, как лохматая шерстяная нить. Пример такой ситуации приведен на рис. 43.

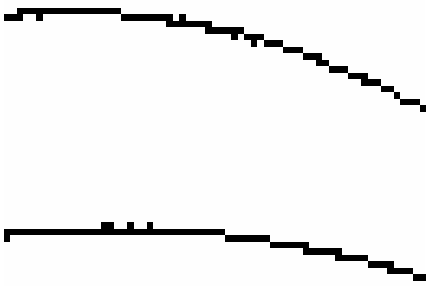


Рис. 43. Пример нерегулярного ступенчатого эффекта

Этот эффект имеет место и в случае конических дуг, однако для лекальных кривых высокого порядка он становится просто нетерпимым. Есть несколько подходов к устранению нерегулярного характера ступенчатого эффекта.

Очевидный способ - измельчение растровой сетки. Вычисления следует проводить с субпиксельной точностью. Собственно говоря, введение субпиксельности неизбежно в любом случае. Каждый вектор при дискретизации дает погрешность, равную 1/2 единицы растра. При суммировании приращений векторов мы суммируем и их погрешности. Отсюда следует, что единица растра должна быть уменьшена в число раз, равное количеству векторов. Таким образом, для генератора конических дуг эта величина равна 2, для лекальных генераторов, построенных по схемам рис. 42, a и рис. 42, b коэффициенты измельчения, соответственно, должны быть равны 4 и 8.

Отметим, что эта оценка обеспечивает лишь требование, чтобы уклонение пикселей от теоретической кривой не превышало $1/2$ единицы растра. Если мы ходим снизить нерегулярность ступенчатого эффекта, коэффициент субпиксельности следует увеличить кратно степени 2.

Практически генерация кривой с субпиксельной точностью означает увеличение длины вектора в целое число раз, а подсветка пикселя должна выполняться не при каждой итерации, а лишь при каждой второй/четвертой/восьмой и т.д., в зависимости от выбранного коэффициента расщепления единицы растра. Например, на рис. 44 коэффициент расщепления равен 4.

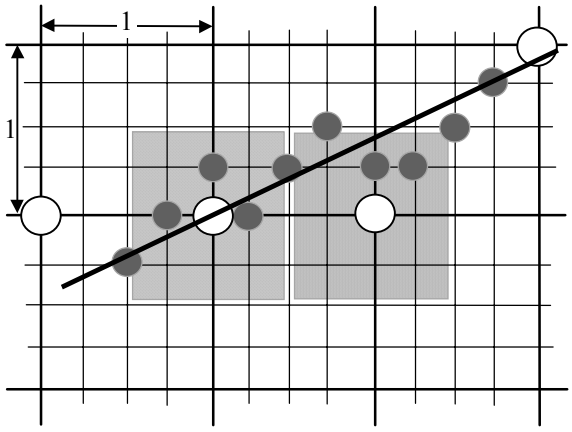


Рис. 44. Расщепление единичной сетки (субпиксельность) позволяет уменьшить нерегулярность ступенчатого эффекта

В пределах единицы растра можно проинтегрировать все субпиксельные приращения координат и найти усредненное значение координаты пикселя. Такое усреднение также ослабляет нерегулярный характер ступенчатого эффекта. Данный подход в целом работоспособен, но приводит к резкому замедлению генерации лекальных кривых. Его следует применять в сочетании с другими методами.

Отметим, что нерегулярность сильнее проявляется, если линейные интерполяторы реализуются на основе алгоритма Брезенхема для векторов, и менее заметна, если линейные интерполяторы реализуют алгоритм ЦДА. Причина такого различия заключается в том, что в первом случае координаты векторов сначала округляются до целого значения, а затем суммируются, при этом погрешности округления также суммируются. В случае алгоритма ЦДА сначала суммируются текущие координаты векторов (в формате с фиксированной точкой), и лишь затем выполняется округление до целого значения, при этом погрешность округления однократна.

Другой способ основан на наблюдении, что нерегулярность имеет место в тех случаях, когда ребра управляющего многоугольника направлены в противоположные стороны. Один вектор увеличивает координату, а на следующей итерации другой вектор ее уменьшает. В результате цепочка смежных пикселей колеблется вдоль теоретического положения линии и создает эффект «шерстяной нити». Выше было показано, что при расщеплении кривых пополам они укорачиваются и спрямляются, а, следовательно, направления их ребер сближаются, что приводит к ослаблению, причем существенному, нерегулярности ступенчатого эффекта.

Этот подход, строго говоря, не требует никаких изменений в вычислительной структуре генератора кривых. Он даже ослабляет требования к субпиксельности, однако при этом перекладывает часть вычислительных затрат на инициализацию алгоритма. Расщепление кривой на последовательность сопряженных сегментов всегда возможно, однако эта задача должна выполняться в прикладной программе, которая инициализирует аппаратно реализованный генератор кривых.

5. Кривые Безье и их свойства

Теперь, когда мы имеем генератор лекальных кривых, рассмотрим специфику его применения для растрирования линий, отличных от лекальных кривых. Лекальные кривые появились как способ математического описания работы рассмотренной выше вычислительной структуры, и хотя их также можно использовать для задания кривых и поверхностей, следует признать, что в компьютерной графике и системах автоматизированного проектирования САПР эти кривые пока не применяются.

За прошедшие десятилетия был предложен ряд функций для задания и управления формой линий и поверхностей. С некоторыми из этих функций мы познакомимся в дальнейшем. Сейчас рассмотрим наиболее популярный в компьютерной графике тип кривых – кривые Безье. Эти кривые даже реализованы в системе Windows, наряду с эллипсами, векторами и прочими графическими примитивами.

5.1. Определение и основные свойства кривых Безье.

Математически кривая Безье степени n задается формулой

$$\vec{P}_n(t) = \sum_{i=0}^{n} B_n^i(t)\vec{P}_i\,, \qquad t \subseteq [0,1],$$

где $\vec{P}_i = \{X_i, Y_i, Z_i, ...\}$ - координаты i-той вершины разомкнутого многоугольника $P_0,...,P_n$, $B_n^i(t) = C_n^i(1-t)^{n-i}t^i$ - базис Бернштейна, $C_n^i = \dfrac{n!}{(n-i)!i!}$ - биномиальный коэффициент.

Кривая Безье обладает многими интересными и полезными свойствами, из которых мы отметим следующие:

1. Кривая Безье начинается в начальной вершине P_0 многоугольника и заканчивается в его конечной точке P_n.
2. В начальной и конечной точках кривая Безье имеет касательные, совпадающие с направлением векторов (ребер многоугольника) P_0P_1 и $P_{n-1}P_n$, соответственно.
3. Кривая всегда лежит внутри выпуклой оболочки, натянутой на вершины многоугольника.
4. Базис Бернштейна не является линейно независимым: из этого следует, что кривые разного порядка могут геометрически описывать одну и ту же линию.
5. Кривая Безье всегда может быть представлена некоторым степенным полиномом, причем степень полинома определяет наименьший порядок кривой Безье.
6. Кривая Безье проходит вблизи вершин многоугольника, причем, чем больше управляющих вершин на данном сегменте линии, тем ближе к ним проходит кривая Безье.

Базисы Бернштейна интуитивно можно рассматривать как некоторые центры притяжения, аналогичные магнитам, а аналогом кривой в этом случае будет закрепленная в конечных точках металлическая (стальная) лента. В качестве примера, иллюстрирующего последнее утверждение, приведем на рис. 45 график семейства базисов пятого порядка.

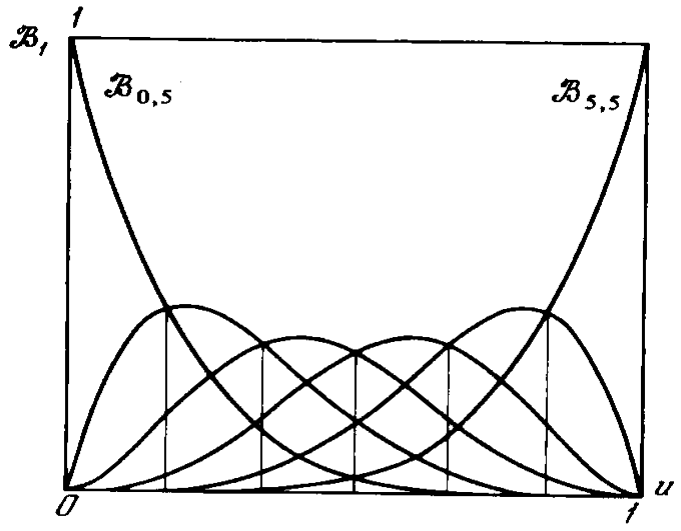


Рис. 45. Базисы Бернштейна пятого порядка

Рассмотрим пример кривой Безье четвертого порядка, Кривая Безье $\vec{P}_{0,n}(t)$ обладает важным математическим свойством:

$$\vec{P}_{0,n}(t) = \vec{P}_{0,n-1}(t)(1-t) + \vec{P}_{1,n}(t)t\,.$$

Благодаря этому свойству можно задать эффективный алгоритм вычисления значения кривой по заданному параметру *t*. Этот алгоритм называется алгоритмом расщепления Кастелью. Ему соответствует следующая рекурсивная процедура:

```
Point3D Bezier(Point3D P[], double t, int n, int i)
{
      if(n == 1)
            return Point3D(P[i] * (1.0-t) + P[i+1] * t);
      else
            return Bezier(P, t, n-1,   i) * (1.0-t) +
                   Bezier(P, t, n-1, i+1) * t;
}
```

Чтобы сгенерировать N точек кривой Безье 4-го порядка следует сделать вызов процедуры `Bezier()` N раз.

```
Point3D P[5] = { ... };    // Координаты вершин
                           //{{X0,Y0,Z0}, ..., {X4,Y4,Z4}}
int N = 100;
double dt = 1.0 / N;

for(double t=0.0; t<=1.0; t += dt)
{
      Point3D Pt = Bezier(P, t, 4, 0);
      Draw(Pt);
}
```

Этот алгоритм

$$\vec{P}_{0,n}(t) = \vec{P}_{0,n-1}(t)(1-t) + \vec{P}_{1,n}(t)t$$

реализует также вычислительную схему, изображенную на рис. 46.

Здесь PE (Processing Element) имеет 2 входа и один выход. На вход подаются две точки, а с выхода снимается одна точка, положение которой рассчитывается по формуле

$$Q_{out} = Q_{in}^{i} t_1 + Q_{in}^{i+1} t\,.$$

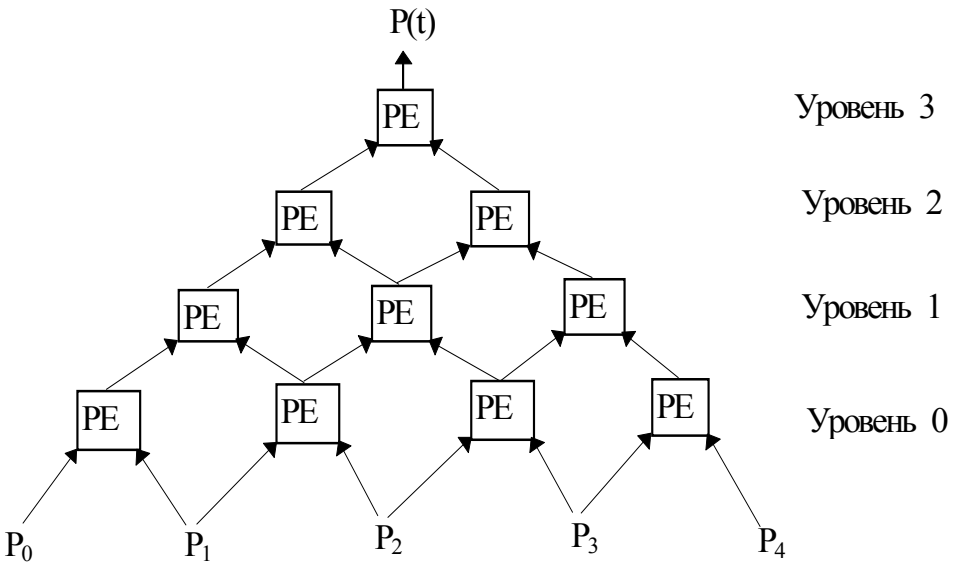


Рис. 46. Вычислительная структура, реализующая алгоритм Кастелью

При реализации этой операции неизбежно хотя бы одно целочисленное умножение. Отметим, что параметры t, $t_1 = 1 - t$ являются постоянными и одинаковыми для всех PE при заданном значении t. Приведенная вычислительная схема имеет очевидную рекурсивную структуру.

Данная вычислительная структура допускает аппаратную реализацию. При этом имеет место распараллеливание вычислительного процесса, что существенно увеличивает скорость вычислений. При программной реализации для вычисления положения одной точки кривой требуется последовательно выполнить 10 раз операцию PE (для кривой 4-го порядка). При аппаратной реализации алгоритма таких последовательных операций потребуется всего 4, а если принять во внимание, что для растрирования кривой необходимо последовательно рассчитывать положение множества точек, то, применяя конвейерную обработку, можно выполнить расчет положения одной точки за время 1 такт PE.

Продолжим исследование кривой Безье. Очевидно, что при расщеплении кривая укорачивается и спрямляется, при этом степень описывающего ее полинома также должна уменьшаться. Оценим степень уменьшения коэффициента при старшей степени полинома t^n при его однократном расщеплении. Рассмотрим алгоритм расщепления на примере рис. 47.

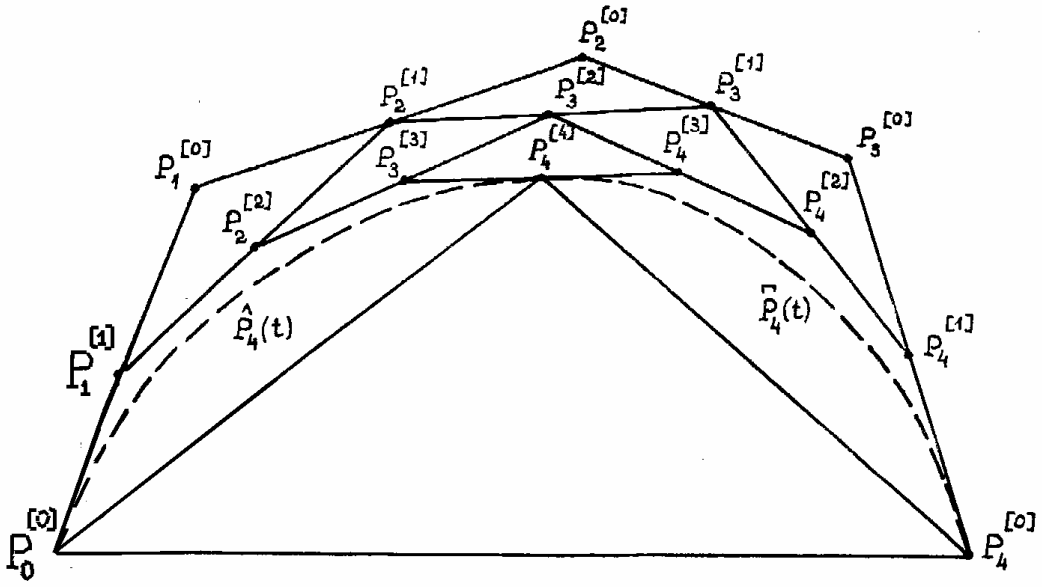


Рис. 47. Иллюстрация алгоритма расщепления кривой Безье

Рис. 47 иллюстрирует работу алгоритма расщепления, в результате которого мы из исходной кривой получили два сопряженных сегмента, каждый из которых также является кривой Безье того же порядка:

$$\vec{P}_n(<\vec{P}_0,...,\vec{P}_n>,t) \begin{cases} \nearrow \hat{P}_n(<\vec{P}_0^{[0]},...,\vec{P}_n^{[n]}>,t) \\ \searrow \tilde{P}_n(<\vec{P}_n^{[n]},...,\vec{P}_n^{[0]}>,t) \end{cases}$$

Если кривая расщепляется пополам $(t = 0.5)$, то положение точек $\vec{P}_k^{[m]}$ можно найти по формуле

$$\vec{P}_k^{[m]} = 2^{-m} \sum_{i=0}^{m} C_m^i \vec{P}_{k-i}^{[0]} .$$

Последнее утверждение доказывается по индукции.

5.2. Повышение и понижение порядка кривой Безье

Кривая Безье может быть представлена в виде степенного полинома

$$\vec{P}_n(t) = \sum_{k=0}^{n} (-1)^k C_n^k \vec{\Delta}_k t^k ,$$

где $\vec{\Delta}_k = \sum_{i=0}^{k} (-1)^i C_k^i \vec{P}_i$. Из приведенной выше формулы видно, что возможна ситуация, когда $\vec{\Delta}_n = 0$, в этом случае степень полинома будет меньше порядка кривой Безье, из которой этот полином был получен. Теперь можно из степенного полинома получить кривую Безье меньшего порядка.

Покажем, что между вершинами исходной кривой Безье и вершинами кривой меньшего порядка имеется простая связь. Пусть $\vec{Z}_{n-1}(<\vec{Z}_0,...,\vec{Z}_{n-1}>,t)$ - кривая Безье $n-1$ порядка. Принимая во внимание, что $1 \equiv (1-t)+t$, умножим $\vec{Z}_{n-1}(t)$ на 1:

$$\vec{P}_n(<\vec{P}_0,...,\vec{P}_n>,t) = \left[(1-t)+t\right]\vec{Z}_{n-1}(<\vec{Z}_0,...,\vec{Z}_{n-1}>,t)$$

и, приравняв коэффициенты (вершины) при одинаковых базисах двух кривых одинакового порядка, находим алгоритм понижения порядка кривой Безье на 1 (при условии $\vec{\Delta}_n = 0$):

$$\vec{Z}_k = \frac{n}{n-k}\vec{P}_k - \frac{k}{n-k}\vec{P}_{k-1}, \qquad k = 0,...,n-1.$$

Повышение порядка кривой на 1 возможно всегда. Для этого надо лишь пересчитать положение вершин по формуле

$$\vec{P}_k = \frac{k}{n}\vec{Z}_{k-1} + (1-\frac{k}{n})\vec{Z}_k, \qquad k = 0,...,n.$$

Приведем иллюстрацию изменения порядка кривой (рис. 48)

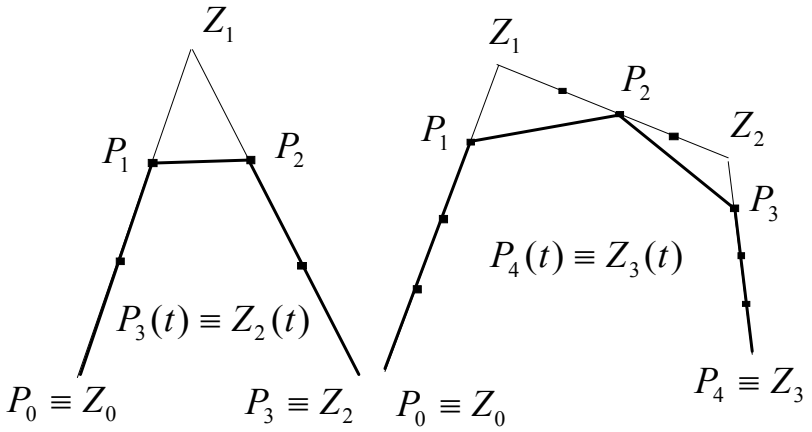


Рис. 48. Повышение порядка кривых Безье 2-го и 3-го порядка на единицу

Изменение порядка кривой Безье - свойство весьма любопытное, но как его можно использовать? Найдем коэффициенты при старшей степени t^n сегментов расщепленной пополам кривой Безье

$$\hat{\Delta}_n = \tilde{\Delta}_n = 2^{-n}\vec{\Delta}_n,$$

где $\vec{\Delta}_n$ - коэффициент при старшей степени t^n исходной кривой Безье, $\hat{\Delta}_n$ - коэффициент при старшей степени t^n левого сегмента кривой после расщепления, $\tilde{\Delta}_n$ - коэффициент при старшей степени t^n правого сегмента кривой после расщепления.

Следовательно, при расщеплении кривой она укорачивается и спрямляется. Очевидно, что, повторяя процедуру расщепления над каждым из сегментов многократно, мы достигнем условия $\left|\hat{\Delta}_n\right| = \left|\tilde{\Delta}_n\right| < 0.5$, то есть момента, когда различие между исходной кривой и ее аналогом меньшего порядка станет меньше половины размера пикселя. Таким образом, мы показали, что расщепление кривой Безье может сопровождаться понижением ее порядка и нашли объективный критерий для применения понижения порядка кривой.

Полученная выше оценка погрешности понижения порядка кривой основывалась на представлении кривой Безье в степенной форме. Отбрасывая последний член полинома, мы получали приближенный полином, причем наибольшая погрешность приходилась на конечную точку кривой. Возможен другой подход, при котором погрешность будет распределена по всей длине кривой. Для этого следует представить кривую Безье в виде степенного полинома, ко-

торый, как известно из математики, всегда может быть разложен по косинусам.

Отбрасывая последнюю гармонику, мы получим приближенный тригонометрический полином, который можно преобразовать в форму Безье, но уже меньшего порядка. Нашей целью будет оценить амплитуду старшей гармоники и предложить алгоритм пересчета полинома из формы Безье в тригонометрическую форму и обратно.

Покажем, что кривая Безье $\vec{P}_n(t)$ эквивалентна тригонометрическому полиному вида

$$\vec{Z}_{0,n}(\tau) = \sum_{k=0}^{n} \vec{Z}_k \cos k\tau, \qquad \tau \in [0, \pi].$$

Доказательство проведем по индукции. При $n = 1$ $\vec{P}_{0,1}(t)$ задает уравнение прямой, проходящей через точки $\vec{P}_0$ и $\vec{P}_1$. Приравнивая $\vec{Z}_{0,1}(\tau) \equiv \vec{P}_{0,1}(t)$, находим параметры тригонометрического полинома:

$$\vec{Z}_0 = \frac{1}{2}\left(\vec{P}_0 + \vec{P}_1\right), \quad \vec{Z}_1 = \frac{1}{2}\left(\vec{P}_0 - \vec{P}_1\right), \quad \tau = \arccos(1 - 2t).$$

Таким образом, основание индукции доказано. Далее, принимая во внимание, что для кривой Безье справедливо соотношение

$$\vec{P}_{0.n}(t) = (1-t)\vec{P}_{0,n-1}(t) + t\vec{P}_{1,n}(t),$$

получаем аналогичное соотношение для тригонометрического полинома:

$$\vec{Z}_{0,n}(\tau) = \frac{1}{2}(1+\cos\tau)\vec{Z}_{0,n-1}(\tau) + \frac{1}{2}(1-\cos\tau)\vec{Z}_{1,n}(\tau),$$

Отсюда, считая полиномы $\vec{Z}_{0,n-1}(<\vec{Z}_0^-,\ldots,\vec{Z}_{n-1}^->,\tau)$ и $\vec{Z}_{1,n}(<\vec{Z}_0^+,\ldots,\vec{Z}_{n-1}^+>,\tau)$ известными, находим:

$$\vec{Z}_{0n}(\tau) = \frac{1}{2}\left(\vec{Z}_0^- - \vec{Z}_0^+\right)\cos\tau + \frac{1}{2}\sum_{k=0}^{n-1}\left(\vec{Z}_k^- + \vec{Z}_k^+\right)\cos k\tau +$$
$$+ \frac{1}{4}\sum_{k=1}^{n-1}(\vec{Z}_k^- - \vec{Z}_k^+)\cos(k-1)\tau + \frac{1}{4}\sum_{k=1}^{n-1}\left(\vec{Z}_k^- - \vec{Z}_k^+\right)\cos(k+1)\tau.$$

Из этого выражения можно выразить набор параметров $\left\langle \vec{Z}_0,...,\vec{Z}_n \right\rangle$ полинома $\vec{Z}_{0n}(\tau)$ через параметры полиномов $\vec{Z}_{0,n-1}(\left\langle \vec{Z}_0^-,...,\vec{Z}_{n-1}^- \right\rangle,\tau)$ и $\vec{Z}_{1,n}(\left\langle \vec{Z}_0^+,...,\vec{Z}_n^+ \right\rangle,\tau)$. Решение этой задачи удобно представить в матричной форме:

$$\vec{\Delta}_n = \hat{\Delta}_n + G_n\vec{\delta}_n,$$

где $\vec{\Delta}_n = \left(\vec{Z}_0,...,\vec{Z}_n\right)$ - набор коэффициентов полинома $\vec{Z}_{0,n}(\tau)$,

$$\hat{\Delta}_n = \left(\frac{1}{2}\left(\vec{Z}_0^- + \vec{Z}_0^+\right),...,\frac{1}{2}\left(\vec{Z}_{n-1}^- + \vec{Z}_{n-1}^+\right),0\right),$$

$$\vec{\delta}_n = \left(\frac{1}{2}\left(\vec{Z}_0^- - \vec{Z}_0^+\right),\frac{1}{4}\left(\vec{Z}_1^- - \vec{Z}_1^+\right)..,\frac{1}{4}\left(\vec{Z}_{n-1}^- - \vec{Z}_{n-1}^+\right)\right),$$

$$G_n = \left(\begin{array}{cccccc|cccc}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
- & - & - & - & - & - & - & - & - & - \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{array}\right).$$

Полученные выше соотношения доказывают эквивалентность полиномов $\vec{P}_{0n}(t)$ и $\vec{Z}_{0n}(\tau)$, но не позволяет выразить параметры одного полинома через известные параметры другого. Найдем алгоритм, по которому решается эта задача, и покажем, что он допускает целочисленную реализацию.

Целью алгоритма является построение матрицы пересчета набора коэффициентов <Z> в набор <P> и обратно

$$Z^{[m]} = S_m P^{[m]},$$

где $P^{[m]} = \left(\vec{P}_0,...,\vec{P}_m\right)$ - массив коэффициентов полинома $\vec{P}_{0n}(t)$,
$Z^{[m]} = \left(\vec{Z}_0,...,\vec{Z}_m\right)$- массив коэффициентов полинома $\vec{Z}_{0n}(\tau)$,
S_m - матрица размерности $(m+1)\times(m+1)$.

Для упрощения математических выкладок при нахождении матрицы S_m введем следующие обозначения:

$$\Delta_m^- = \begin{bmatrix} & & & 0 \\ & S_{m-1} & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad \Delta_m^+ = \begin{bmatrix} 0 & & & & \\ 0 & & & & \\ 0 & & S_{m-1} & & \\ 0 & & & & \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\hat{\Delta}_m = \frac{1}{2}\left(\Delta_m^- + \Delta_m^+\right),$$

$$\vec{\delta}_m = \frac{1}{4}\left(\Delta_m^- - \Delta_m^+\right).$$

Отсюда находим

$$S_m^{[i]} = \delta_m^{[i-1]} + \Delta_m^{[i]} + \delta_m^{[i+1]},$$

где $S_m^{[i]}$ - i-я строка матрицы S_m, $\delta_m^{[i-1]}$-(i-1) -я строка матрицы δ_m, $\delta_m^{[i+1]}$ - (i+1)-я строка матрицы δ_m, $\Delta_m^{[i]}$ - i-я строка матрицы $\hat{\Delta}_m$, $S_m, \delta_m, \hat{\Delta}_m$ - матрицы размерности $(m+1)\times(m+1)$.

Для того, чтобы предлагаемый алгоритм можно было использовать, необходимо найти основание рекурсии. Так как

$$\begin{bmatrix} \vec{Z}_0 \\ \vec{Z}_1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} \vec{P}_0 \\ \vec{P}_1 \end{bmatrix},$$

то

$$S_1 = \frac{1}{2}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Реализуя рассмотренный алгоритм, последовательно находим матрицы $S_2,...,S_m$, что и являлось целью нашего исследования. Отметим, что матрицы S_m удобно представить в виде $S_m = 2^{1-2m} L_m$. Тогда матрица L_m будет содержать лишь целочисленные коэффициенты. На этом задачу представления кривой Безье в виде тригонометрического полинома можно считать решенной.

Приведем С-код, реализующий данный алгоритм расчета матриц.

```
void BezierToCos(int k)
{
      static long P[10][10],M[10][10],R[10][10];
      static long S[10][10], D[10][10];
      int i,j;

      for(i=0; i<k; i++)
      for(j=0; j<k; j++)
          P[i][j] = M[i][j] = R[i][j] = S[i][j] = D[i][j] = 0L;

      D[0][0] = D[0][1] = D[1][0] = 1L;      D[1][1] = -1L;

      int n=1;
      for(int L=2; L<=k; L++)
      {
            n <<= 2;
            for(i=0; i<=L-1; i++)
            for(j=0; j<=L-1; j++)
                  M[i][j ] = P[i][j+1] = D[i][j];

            for(i=0; i<=L; i++)
            for(j=0; j<=L; j++)
            {
                  S[i][j] = (M[i][j] + P[i][j]) << 1;
                  R[i][j] =  M[i][j] - P[i][j];
            }

            for(i=0; i<=L; i++) R[0][i] <<= 1;

            for(i=0; i<=L-1; i++)
            for(j=0; j<=L;   j++)
                  D[i][j] = S[i][j];

            for(j=0; j<=L; j++)
            {
                  D[0  ][j] += R[1  ][j];
                  D[L-1][j] += R[L-2][j];
                  D[L  ][j]  = R[L-1][j];
            }

            for(i=1; i<=L-2; i++)
            for(j=0; j<=L;   j++)
                  D[i][j] += R[i-1][j] + R[i+1][j];
```

```
            if(L<k) continue;

            printf("\r\n"); printf("N=%d",n<<1); printf("\r\n");
            for(i=0; i<=L; i++)
            {
                  printf("\r\n");
                  for(j=0; j<=L; j++) printf("%7d", D[i][j]);
            }
      }
}
```

С помощью данной программы можно рассчитать матрицы любого порядка, но поскольку алгоритм реализован в целых числах, то имеется ограничение на разрядность представления данных.

Приведем несколько матриц для кривых Безье до пятого порядка включительно:

$$S_1 = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \qquad S_2 = \frac{1}{8}\begin{pmatrix} 3 & 2 & 3 \\ 4 & 0 & -4 \\ 1 & -2 & 1 \end{pmatrix},$$

$$S_3 = \frac{1}{32}\begin{pmatrix} 10 & 6 & 6 & 10 \\ 15 & 3 & -3 & -15 \\ 6 & -6 & -6 & 6 \\ 1 & -3 & 3 & -1 \end{pmatrix}, \quad S_4 = \frac{1}{128}\begin{pmatrix} 35 & 20 & 18 & 20 & 35 \\ 56 & 16 & 0 & -16 & -56 \\ 28 & -16 & -24 & -16 & 28 \\ 8 & -16 & 0 & 10 & -8 \\ 1 & -4 & 6 & -4 & 1 \end{pmatrix},$$

$$S_5 = \frac{1}{512}\begin{pmatrix} 126 & 70 & 60 & 60 & 70 & 128 \\ 210 & 70 & 20 & -20 & -70 & -210 \\ 120 & -40 & -80 & -80 & -40 & 120 \\ 45 & -65 & -30 & 30 & 65 & -45 \\ 10 & -30 & 20 & 20 & -30 & 10 \\ 1 & -5 & 10 & -10 & 5 & 1 \end{pmatrix}.$$

S_1 соответствует вектору, а S_2 – параболе.

Из приведенных выше матриц видно, что амплитуда самой высокочастотной гармоники вычисляется по формуле $\vec{Z}_n = 2^{1-2n}\vec{\Delta}_n$, где $\vec{\Delta}_n = \sum_{k=0}^{n}(-1)^k C_n^k \vec{P}_k$ - n-я разделенная разность. Очевидно, что гармоника тригонометрического полинома изменяется в пределах

своей амплитуды вдоль линии графика полинома. Отбрасывая старшую гармонику, мы вносим погрешность не более, чем $\left\|\vec{Z}_n\right\|$. Если потребовать, чтобы при понижении порядка погрешность полинома меньшей степени лежала в пределах половины единицы пикселя, то мы получим объективный критерий для понижения порядка кривой Безье. Алгоритм представления кривой Безье набором сопряженных сегментов (кривых Безье меньшего на единицу порядка) можно представить в следующем виде:

```
Push (<P_n>);   // Загрузка в стек вершин исходной кривой Безье.
while (Stack is not empty)    // Повторять, пока стек не опустеет.
{
     do  // Расщеплять исх. кривую до тех пор, пока не появится
     {     // возможность заменить ее сегменты кривыми Безье
           //   меньшего  порядка.
       P[n] = Pop(<P_n>); // Вершины кр. Безье берутся из стека
       Z[n] =G_n P[n];          // Амплит.гармоник тригон. полинома
       if(|Z_n|<0.5)            // Критерий отброса старшей гармоники
       {
              Push<P[n]>; // Сегмент готов к понижению порядка
              break;
       }
       Split<P[n]>→Push(<P[n]_left>),Push(<P[n]_right>);// Расщепление
                                                    // кривой Безье
     }
     while(1);
     do // Понизить порядок сегм. Безье, если это допустимо, и
     {                          // извлечь их из стека
       P[n] = Pop(<P_n>); // Извлечь из стека вершины кр. Безье
       Z[n] =G_n P[n];          // Найти ампл. гармоник триг. полинома
       if(|Z_n|<0.5) // Проверить критерий отброса старшей гарм.
       {
              Z[n] → Z[n-1];          // Отбросить старшую гармонику
              P[n-1] = Z[n-1] G_{n-1}^{-1} // Понизить порядок  кр. Безье
              Out(P[n-1]);            // Сохр. вершины нового сегмента
              continue;               // Перейти к следующему сегменту
       }
       break;                         // В стеке больше нет пригодных
                                  // для понижения их  порядка сегментов
     }
     while(Stack is not empty)
}
```

В результате работы данного алгоритма мы представим исходную кривую Безье набором сегментов, каждый из которых будет также являться кривой Безье, но порядок такой кривой будет на единицу меньше, чем у исходной кривой. Применяя эту процедуру многократно, мы сможем представить исходную кривую набором смежных кривых Безье невысокого порядка.

Напомним, что быстрая поточечная генерация возможна лишь для кривых Безье невысокого порядка. Генератор конических дуг может растрировать лишь параболы (кривые Безье второго порядка), а генератор лекальных кривых (на основе трех CI) может растрировать кривую не выше четвертого порядка. Алгоритм Кастелью (de Casteljau algorithm) можно использовать для растрирования кривой любого порядка, но его аппаратная реализация (число PE ограничено) также накладывает ограничение на степень кривой Безье. Таким образом, мы показали, что понижение порядка кривой есть необходимое условие ее эффективного растрирования.

5.3. Генерация кривых Безье с помощью генератора лекальных кривых

Если настроить все конические интерполяторы на гиперболический режим ($\alpha = 1/2$)

$$x(t) = 2t - t^2, \qquad y(t) = t^2,$$

то выходной управляющий сигнал лекального интерполятора будет задаваться следующими формулами:

$$\varphi_1(t) = 4t - 6t^2 + 4t^3 - t^4,$$
$$\varphi_2(t) = 4t^2 - 4t^3 + t^4,$$
$$\varphi_3(t) = 2t^2 - t^4,$$
$$\varphi_4(t) = t^4,$$

а лекальная кривая

$$\vec{A}(t) = \vec{A}_0 + \vec{A}_{01}\varphi_1(t) + \vec{A}_{12}\varphi_2(t) + \vec{A}_{23}\varphi_3(t) + \vec{A}_{34}\varphi_4(t)$$

будет представлять собой некоторый степенной полином. Поскольку кривая Безье $\vec{P}_n(t)$ также может быть представлена в форме степенного полинома

$$\vec{P}_n(t) = \sum_{k=0}^{n} \vec{P}_k C_n^k (1-t)^{n-k} t^k = \sum_{k=0}^{n} (-1)^k t^k \sum_{i=0}^{k} (-1)^i C_k^i \vec{P}_i ,$$

то, приравнивая коэффициенты при одинаковых степенях полиномов $\vec{A}(t)$ и $\vec{P}_4(t)$, получаем линейную систему четырех уравнений. Решая эту систему методом исключения, находим связь между вершинами кривой Безье и эквивалентной ей лекальной кривой:

$$\vec{P}_0 = \vec{A}_0, \qquad \vec{P}_1 = \vec{A}_1, \qquad \vec{P}_2 = \frac{1}{3}(\vec{A}_1 + \vec{A}_2 + \vec{A}_3), \qquad \vec{P}_3 = \vec{A}_3 .$$

Аналогичные соотношения можно получить и для кривой Безье третьего порядка

$$\vec{P}_0 = \vec{A}_0, \qquad \vec{P}_1 = \frac{1}{4}(\vec{A}_0 + 3\vec{A}_1), \qquad \vec{P}_2 = \frac{1}{4}(\vec{A}_4 + 3\vec{A}_3), \qquad \vec{P}_3 = \vec{A}_4 .$$

Пересчет вершин кривой Безье в вершины эквивалентной ей лекальной кривой удобно представить матричным преобразованием **A**=G**P**:

$$\begin{pmatrix} \vec{A}_{01} \\ \vec{A}_{12} \\ \vec{A}_{23} \\ \vec{A}_{34} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & -1 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{P}_{01} \\ \vec{P}_{12} \\ \vec{P}_{23} \\ \vec{P}_{34} \end{pmatrix},$$

$$\begin{pmatrix} \vec{A}_{01} \\ \vec{A}_{12} \\ \vec{A}_{23} \\ \vec{A}_{34} \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 3 & 0 & 0 \\ 2 & 2 & -1 \\ -1 & 2 & 2 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} \vec{P}_{01} \\ \vec{P}_{12} \\ \vec{P}_{23} \end{pmatrix}.$$

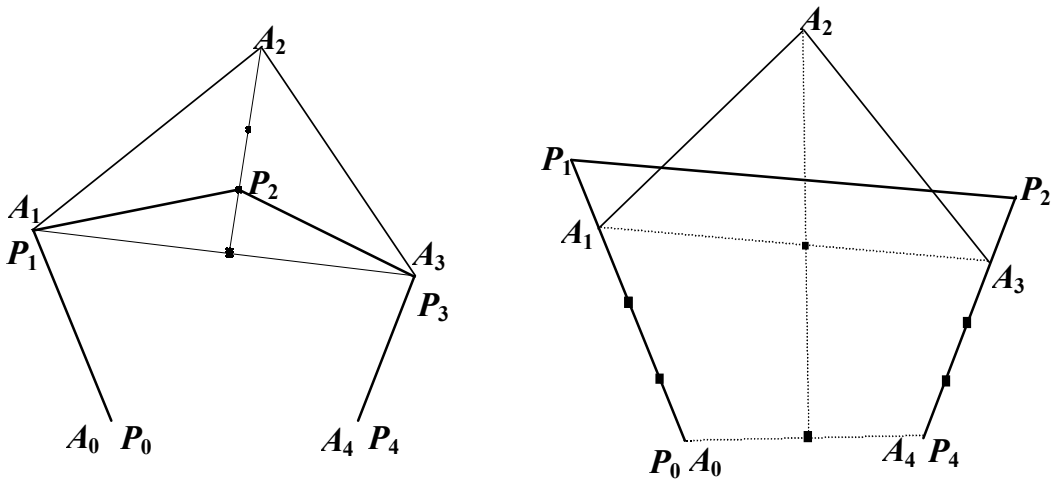


Рис. 49. Связь между управляющими вершинами кривых Безье 4-го и 5-го порядков и эквивалентных им лекальных кривых

Приведенные матрицы соответствуют лекальному генератору с тремя коническими интерполяторами (рис. 42, a). На рис. 49 полученные соотношения представлены в графическом виде (<*P*> - вершины кривой Безье, а <*Z*> - вершины лекальной кривой, эквивалентной данной кривой Безье). Следующая схема, состоящая из семи конических интерполяторов (рис. 42, b) и восьми линейных интерполяторов, способна растрировать кривые Безье до восьмого порядка включительно.

Методика построения матриц пересчета положения вершин кривой Безье в вершины эквивалентной ей лекальной кривой тождественна использованной для предшествующей вычислительной структуры. Приведем матрицы пересчета $G_{8,n}$ для кривых Безье порядков n=8,…, n=5.

$$G_{8,8}=\frac{1}{3}\begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 18 & -52 & 53 & -24 & 4 & 0 \\ 0 & 18 & -129 & 312 & -318 & 144 & -24 & 0 \\ 0 & 32 & -192 & 424 & -416 & 186 & -31 & 0 \\ 0 & -31 & 186 & -416 & 424 & -192 & 32 & 0 \\ 0 & -24 & 144 & -318 & 312 & -129 & 18 & 0 \\ 0 & 4 & -24 & 53 & -52 & 18 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

$$G_{8,7}=\frac{1}{24}\begin{pmatrix} 21 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 60 & -66 & 4 & 39 & -24 & 4 \\ 18 & -150 & 291 & -24 & -234 & 144 & -24 \\ 32 & -192 & 312 & 32 & -318 & 186 & -31 \\ -31 & 186 & -318 & 32 & 312 & -192 & 32 \\ -24 & 144 & -234 & -24 & 291 & -150 & 18 \\ 4 & -24 & 39 & 4 & -66 & 60 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 21 \end{pmatrix},$$

$$G_{8,6}=\frac{1}{4}\begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 2 & 4 & -6 & 4 & -1 & 0 \\ -1 & -4 & 26 & -24 & 6 & 0 \\ 0 & -8 & 32 & -28 & 7 & 0 \\ 0 & 7 & -28 & 32 & -8 & 0 \\ 0 & 6 & -24 & 26 & -4 & -1 \\ 0 & -1 & 4 & -6 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 & 3 \end{pmatrix},$$

$$G_{8,5}=\frac{1}{24}\begin{pmatrix} 15 & 0 & 0 & 0 & 0 \\ 14 & 4 & -6 & 4 & -1 \\ -9 & 36 & 6 & -24 & 6 \\ -8 & 32 & 12 & -28 & 7 \\ 7 & -28 & 12 & 32 & -8 \\ 6 & -24 & 6 & 36 & -9 \\ -1 & 4 & -6 & 4 & 14 \\ 0 & 0 & 0 & 0 & 15 \end{pmatrix}.$$

Отметим, что многоугольники, задающие кривые Безье и эквивалентные им лекальные кривые, различны. Быстродействие генератора лекальных кривых, как было показано выше, определяется вычислительной структурой (уровнем ее организации) и величиной R, которая вычисляется над набором сторон разомкнутого многоугольника $<A_k>$. Рассчитаем R_A над набором ребер $<A_k>$, и R_P над набором $<P_k>$:

$$R_A = \max\{\|\vec{A}_k\|\}, \quad R_P = \max\{\|\vec{P}_k\|\},$$

где $\|\vec{A}_k\| = \max\{|X_k - X_{k-1}|, |Y_k - Y_{k-1}|, |Z_k - Z_{k-1}|, ...\}$, $\|\vec{P}_k\|$ определяется аналогично $\|\vec{A}_k\|$.

Нетрудно заметить, что $R_A > R_P$, то есть генерация кривой Безье будет выполняться медленнее, чем генерация лекальной кривой, вписанной в тот же многоугольник $<P_k>$. Найдя отношение R_A / R_P, можно оценить коэффициент замедления кривой Безье по сравнению с генерацией лекальной кривой, вписанной в тот же многоугольник.

Пусть $<P_k>$ - ребра кривой Безье, тогда $\vec{A}_k = \sum_i g_{ki} \vec{P}_i$ - ребра эквивалентной ей лекальной кривой. Получаем оценку

$$\frac{R_A}{R_P} \le \max_k \{\sum_i |g_{ki}|\}.$$

Матрицы g_{ki} приведены выше. На основе этих расчетов составлена следующая таблица:

Порядок кривой Безье	2	3	4	5	6	7	8
R_A / R_P	1	1.25	3	3.6	18.7	48.9	447

Из таблицы видно, что генерация кривых Безье выше 5-го порядка с помощью генератора лекальных кривых нецелесообразна. Таким образом, хотя генератор лекальных кривых является достаточно быстродействующим устройством, его возможности в плане генерации кривых Безье высокого порядка ограничены. Отметим, что уменьшение быстродействия никак не влияет на устойчивость работы генератора: быстрее или медленнее, но кривая все равно будет сгенерирована.

5.4. Представление равномерного B-сплайна набором кривых Безье

Описание кривых сплайнами имеет много качеств, привлекательных для использования в системах интерактивного геометриче-

ского проектирования. Разработано множество алгоритмов и программ для представления кривых B-сплайнами. Нашей целью является показать, что растрирование сплайна может быть успешно выполнено с помощью лекального интерполятора.

Из теории сплайнов известно, что B-сплайн можно представить набором сопряженных сегментов, задаваемых следующим уравнением:

$$\vec{Z}_m^{[J]}(t) = \sum_{k=0}^{m} \vec{Z}_{m-k+J} B_{k,m}(t),$$

где $\vec{Z}_m^{[J]}(t)$ - J-тый сегмент сплайн-функции, являющийся степенным полиномом степени m, J - индекс сегмента сплайн-функции, $t \in [0,1]$ - скалярный параметр. В данном разделе мы рассмотрим только равномерные сплайны, для которых

$$B_{k,m}(t) = \frac{1}{m!} \sum_{i=0}^{k} (-1)^i C_{m+1}^i (t+k-i)^m ,$$

где $C_n^k = \dfrac{n!}{k!(n-k)!}$ - биномиальный коэффициент.

Представим сегмент $\vec{Z}_m^{[J]}(t)$ в виде степенного полинома

$$\vec{Z}_m^{[J]}(t) = \frac{1}{m!} \sum_{k=0}^{m} C_m^k t^k \sum_{i=0}^{m} \vec{Z}_{i+J} \sum_{j=0}^{m-i} (-1)^j C_{m+1}^j (m-i-j)^{m-k} .$$

Кривая Безье также может быть представлена в форме степенного полинома

$$\vec{P}_m(t) = \sum_{k=0}^{m} \vec{P}_k (1-t)^{m-k} t^k = \sum_{k=0}^{m} (-1)^k C_m^k t^k \sum_{i=0}^{k} (-1)^i C_k^i \vec{P}_i .$$

Выражения $\vec{Z}_m^{[J]}(t)$ и $\vec{P}_m^{[J]}(t)$ описывают одну и ту же функцию, откуда следует их эквивалентность. Приравнивая коэффициенты при одинаковых степенях t, находим связь между управляющими точками сплайна и управляющими точками кривой Безье, представляющей J-й сегмент сплайна:

$$\sum_{i=0}^{k}(-1)^i C_k^i \vec{P}_i^{[J]} = \frac{(-1)^k}{m!}\sum_{i=0}^{m}\vec{Z}_{i+J}\sum_{j=0}^{m-i}(-1)^j C_{m+1}^j (m-i-j)^{m-k} .$$

Из комбинаторной математики известна справедливость соотношений

$$b_k = \sum_{i=0}^{k}(-1)^i C_k^i a_i , \qquad a_k = \sum_{i=0}^{k}(-1)^i C_k^i b_i .$$

Из этих соотношений легко видеть, что

$$\vec{P}_k^{[J]} = \frac{1}{m!}\sum_{l=0}^{k} C_k^l \sum_{i=0}^{m}\vec{Z}_{i+J}\sum_{j=0}^{m-i}(-1)^j C_{m+1}^j (n-j-j)^{m-i} , \quad k \in \{0,...,m\} .$$

Таким образом, мы нашли формулу для пересчета управляющих точек сплайна в управляющие точки его сегментов, заданных в форме Безье. Данная формула является математическим решением, но она не подходит для быстрых вычислений. Представим решение в матричной форме

$$(\vec{P}_0^{[J]},...,\vec{P}_m^{[J]})^T = \frac{1}{m!} G_m \cdot (\vec{Z}_J,...,\vec{Z}_{m+J})^T ,$$

где: «^{T}» - операция транспонирования вектора, G_m - матрица размерности $m \times m$,

$$g_{k,i}^{[m]} = \sum_{i=0}^{k} C_k^i \sum_{j=0}^{m-i}(-1)^j C_{m+1}^j (m-i-j)^{m-i} , \qquad k,i \in [0,m].$$

Принимая во внимание соотношение для биномиальных коэффициентов $C_m^k = C_{m-1}^k + C_{m-1}^{k-1}$, представим $g_{k,i}^{[m]}$ в рекуррентной форме

$$g_{k,i}^{[m]} = g_{k-1,i}^{[m]} + g_{k-1,i-1}^{[m-1]} - g_{k-1,i}^{[m-1]},$$

где

$$k \in [1,m], \qquad g_{0,0}^{[m]} = 1, \qquad g_{0.i}^{[m]} = \sum_{j=0}^{m-1} g_{j,i}^{[m-1]} , \qquad m! = \sum_{i=0}^{m} g_{0,i}^{[m]} .$$

Теперь мы можем предложить простой алгоритм для вычисления матрицы G_m любого порядка.

```
void Gm(int M)
{
      static long g[13][13], g1[13][13];// 13 – максим. порядок
      int i=0, j=0, F=0;                // для 32 разрядн.целых

      g[0][0]  = g1[0][0] = g1[1][1] = 1;
      g1[0][1] = g1[1][0] = 0;

      for(int m=2; m <= M;  m++)
      {
            for(F=i=0; i < m; i++)
            {
                  for(F=j=0; j < m; j++)
                        F += g1[j][i];
                  g[0][i] = F;
            }

            for(g[0][m]=0, F=i=1; i<m; i++)
            {
                  for(j=i, F +=g[0][i]; j< m; j++)
                     g[i][j]=g[i-1][j]+g1[i-1][j-1]-g1[i-1][j];
                  g[i][m] = 0;
            }


            for(i=1, g[m][m]=1; i <= m; i++)
                  for(j=0; j < i; j++)
                        g[i][j] = g[m-i][m-j];

            for(i=0; i <= m; i++)
                  for(j=0; j<=m; j++)
                        g1[i][j]=g[i][j];
      }

      for(i=0,  printf("m! = %ld", F); i<m; i++)

      for(j=0,  printf("\r\n"); j<m; j++)
            printf("%8d", g[i][j]);
}
```

Пользуясь данной программой, рассчитаем матрицы пересчета для равномерных сплайнов 3, … ,7 порядков.

$$G_3 = \frac{1}{3!}\begin{pmatrix} 1 & 4 & 1 & 0 \\ 0 & 4 & 2 & 0 \\ 0 & 2 & 4 & 0 \\ 0 & 1 & 4 & 1 \end{pmatrix}, \qquad G_4 = \frac{1}{4!}\begin{pmatrix} 1 & 11 & 11 & 1 & 0 \\ 0 & 8 & 14 & 2 & 0 \\ 0 & 4 & 16 & 4 & 0 \\ 0 & 2 & 14 & 8 & 0 \\ 0 & 1 & 11 & 11 & 1 \end{pmatrix},$$

$$G_5 = \frac{1}{5!}\begin{pmatrix} 1 & 26 & 66 & 26 & 1 & 0 \\ 0 & 16 & 66 & 36 & 2 & 0 \\ 0 & 8 & 60 & 48 & 4 & 0 \\ 0 & 4 & 48 & 60 & 8 & 0 \\ 0 & 2 & 36 & 66 & 16 & 0 \\ 0 & 1 & 26 & 66 & 26 & 1 \end{pmatrix},$$

$$G_6 = \frac{1}{6!}\begin{pmatrix} 1 & 57 & 302 & 302 & 57 & 1 & 0 \\ 0 & 32 & 262 & 342 & 82 & 2 & 0 \\ 0 & 16 & 212 & 372 & 116 & 4 & 0 \\ 0 & 8 & 160 & 384 & 160 & 8 & 0 \\ 0 & 4 & 116 & 372 & 212 & 16 & 0 \\ 0 & 2 & 82 & 342 & 262 & 32 & 0 \\ 0 & 1 & 57 & 302 & 302 & 57 & 1 \end{pmatrix},$$

$$G_7 = \frac{1}{7!}\begin{pmatrix} 1 & 120 & 1191 & 2416 & 1191 & 120 & 1 & 0 \\ 0 & 64 & 946 & 2416 & 1436 & 176 & 2 & 0 \\ 0 & 32 & 716 & 2336 & 1696 & 256 & 4 & 0 \\ 0 & 16 & 520 & 2176 & 1952 & 368 & 8 & 0 \\ 0 & 8 & 368 & 1952 & 2176 & 520 & 16 & 0 \\ 0 & 4 & 256 & 1696 & 2336 & 716 & 32 & 0 \\ 0 & 2 & 176 & 1436 & 2416 & 946 & 64 & 0 \\ 0 & 1 & 120 & 1191 & 2416 & 1191 & 120 & 1 \end{pmatrix}.$$

Таким образом, мы показали, что сплайн можно представить набором сопряженных кривых Безье, которые в свою очередь могут быть растрируемы с помощью генератора лекальных кривых. Получение матриц пересчета в целочисленном виде также гарантирует точность вычислений.

Наглядной иллюстрацией полученных результатов для сплайнов третьей и четвертой степеней может служить рис. 50. Здесь <*Z*> - опорные точки сплайна, <*P*> - опорные точки кривой Безье (сегмента сплайна). Левый рисунок соответствует сплайнам третьей степени, правый - сплайнам четвертой степени.

На рис. 51 приведен пример представления равномерного сплайна третьего порядка набором сопряженных кривых Безье третьего порядка. Отметим, что первая и последняя опорные точки сплайна являются «висящими», их задание неочевидно, однако, если кривая замкнута, эта проблема снимается.

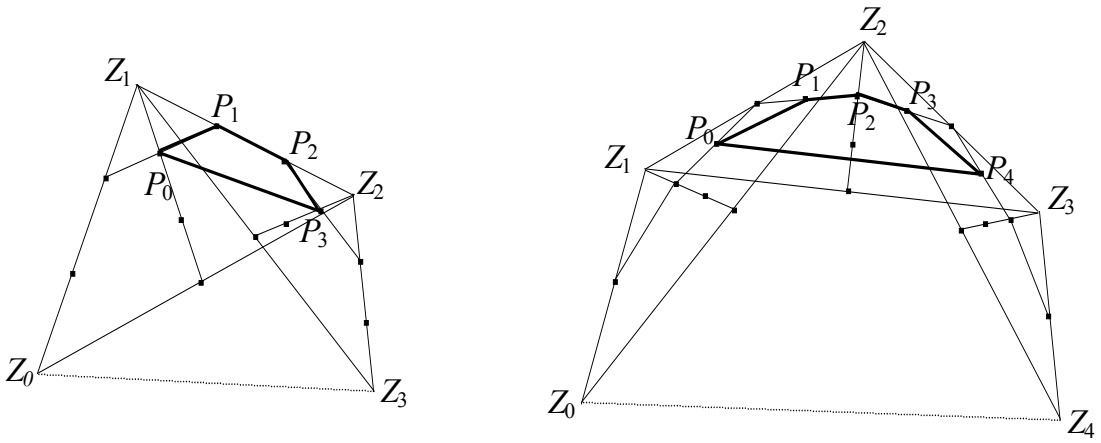


Рис. 50. Пересчет управляющих точек равномерных B-сплайнов 3-го и 4-го порядков в управляющие точки кривых Безье, соответствующих сегментам сплайнов

Построение сплайнов (интерполяционных или сглаживающих), строго говоря, не является нашей задачей, наша цель – визуализировать заданный сплайн доступными нам средствами, каковыми являются, например, кривые Безье.

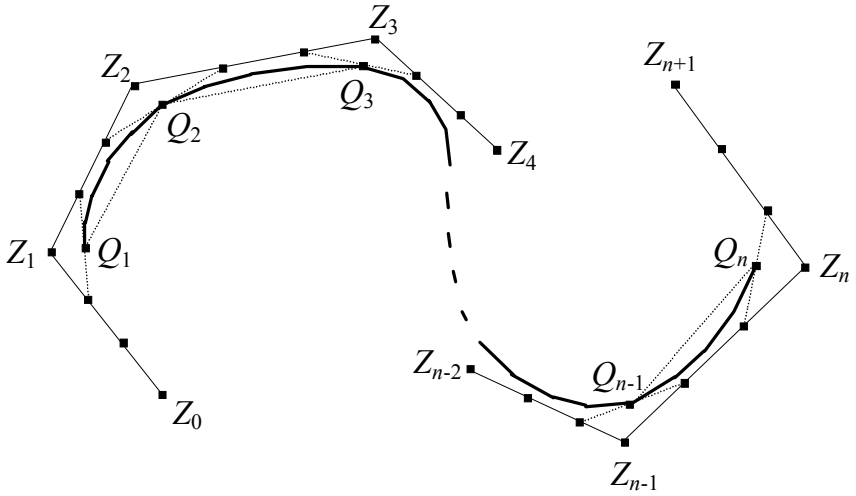


Рис. 51. Представление равномерного B-сплайна 3-го порядка набором сопряженных кривых Безье

5.5. Представление неравномерного B-сплайна набором кривых Безье

Для задания кривых имеется множество методов – методы Лагранжа, Эрмита, Безье, B-сплайнов, NURB, напряженных сплайнов и многие другие. До последнего времени наибольшей популярностью пользовался метод B-сплайнов. Для управления формой кривой метод B-сплайнов использует управляющие опорные точки, в общем случае не лежащие на самой кривой. При этом каждая опорная точка влияет на поведение сплайна лишь на ограниченном участке кривой. Это свойство называется локальностью сплайна и является существенным достоинством такого способа задания кривых.

Сплайн является слишком сложной функцией для непосредственной реализации его в качестве графического примитива, однако вследствие локальности сплайн может быть представлен последовательностью кусочных полиномов, заданных в форме Безье, для которых, как было показано ранее, имеются эффективные методы растрирования, допускающие реализацию как программными, так и аппаратными средствами.

Представление кривой на экране состоит из трех этапов:

1. Аппроксимация исходных данных сплайном.
2. Представление сплайна в виде последовательности кривых Безье.
3. Последовательная визуализация полученного набора кривых Безье.

Первый этап решается методами теории сплайнов и не рассматривается в компьютерной графике. Мы пользуется для получения сплайна уже готовыми алгоритмами. Третий этап был рассмотрен нами ранее. Покажем теперь, что представление сплайна набором кривых Безье также выполняется элементарными вычислительными средствами.

B-сплайн степени m на сегменте $[x_i, x_{i+1}]$ определяется выражением

$$N_{i,m} = \frac{x - x_i}{x_{i+m} - x_i} N_{i,m-1}(x) + \frac{x_{i+m+1} - x}{x_{i+m+1} - x_{i+1}} N_{i+1,m-1}(x),$$

где

$$N_{i,0}(x) = \begin{cases} 1, if\,(x \in [x_i, x_{i+1}]), \\ 0, in\_other\_cases. \end{cases}$$

Набор <x_i> называется узловым вектором, который определяет положение точек склеивания кусочных сегментов сплайна. Если принять $x_i = iL$, где L – константа, близкая к длине кривой, то мы получим равномерный сплайн, который мы уже рассмотрели выше.

В общем виде сплайн задается функцией

$$\vec{Z}_m^{[J]}(t) = \sum_{k=0}^{m} \vec{Z}_{m-k+J} N_{k,m}(t).$$

Для примера рассмотрим простейший вид сплайна - линейный.

$$\vec{Z}(t) = \vec{Z}_0 N_{0,1}(t) + \vec{Z}_1 N_{1,1}(t) + \vec{Z}_2 N_{2,1}(t) + ... + \vec{Z}_n N_{n,1} =$$

$$\vec{Z}_0\left(\frac{t-t_0}{t_1-t_0} + \frac{t_2-t}{t_2-t_1}\right) +$$

$$\vec{Z}_1\left(\frac{t-t_1}{t_2-t_1} + \frac{t_3-t}{t_3-t_2}\right) +$$

$$\vec{Z}_2\left(\frac{t-t_2}{t_3-t_2} + \frac{t_3-t}{t_4-t_3}\right) + ...$$

$[t_0, t_1]$ $[t_1, t_2]$ $[t_2, t_3]$ $[t_3, t_4]$...

Суммируя по вертикали для каждого диапазона $[t_i, t_{i+1}]$, находим:

$$\vec{Z}_{01}(u) = \vec{Z}_0(1-u) + \vec{Z}_1 u, \qquad u = \frac{t-t_0}{t_1-t_0},$$

$$\vec{Z}_{12}(u) = \vec{Z}_1(1-u) + \vec{Z}_2 u, \qquad u = \frac{t-t_1}{t_2-t_1},$$

$$\vec{Z}_{23}(u) = \vec{Z}_2(1-u) + \vec{Z}_3 u, \qquad u = \frac{t-t_2}{t_3-t_2},$$

* * * * * * * * * * * * * *

$$\vec{Z}_{n-1n}(u) = \vec{Z}_{n-1}(1-u) + \vec{Z}_n u, \quad u = \frac{t-t_{n-1}}{t_n-t_{n-1}}.$$

Аналогичные выражения можно получить для сплайна любого порядка. Опуская утомительные алгебраические выкладки, приведем матрицы для пересчета параметров сплайна в опорные точки соответствующих ему кривых Безье (i - й сегмент).

1. Линейный сплайн

$$\begin{pmatrix} \vec{P}_i \\ \vec{P}_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \vec{Z}_i \\ \vec{Z}_{i+1} \end{pmatrix}.$$

2. Квадратичный сплайн

$$\begin{pmatrix} \vec{P}_0^{[i]} \\ \vec{P}_1^{[i]} \\ \vec{P}_2^{[i]} \end{pmatrix} = \begin{pmatrix} 1-t_{12} & t_{12} & 0 \\ 0 & 1 & 0 \\ 0 & 1-t_{23} & t_{23} \end{pmatrix} \begin{pmatrix} \vec{Z}_i \\ \vec{Z}_{i+1} \\ \vec{Z}_{i+2} \end{pmatrix}.$$

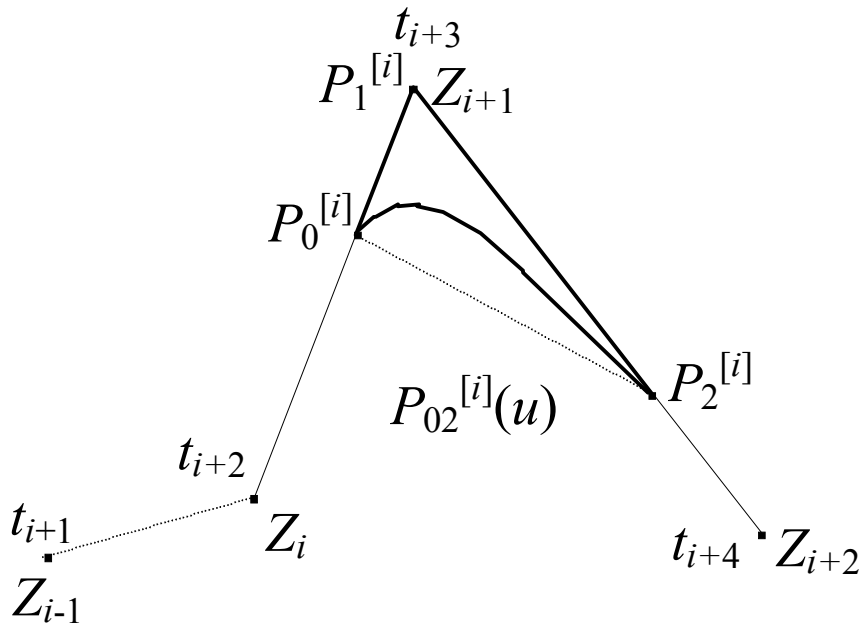


Рис. 52. Пример параболического сегмента B-сплайна 2-го порядка

3. Кубический сплайн

$$\begin{pmatrix} \vec{P}_0^{[i]} \\ \vec{P}_1^{[i]} \\ \vec{P}_2^{[i]} \\ \vec{P}_3^{[i]} \end{pmatrix} = \begin{pmatrix} \dfrac{t_{34}^2}{t_{14}t_{24}} & \dfrac{t_{13}t_{34}}{t_{14}t_{24}} + \dfrac{t_{35}t_{23}}{t_{24}t_{25}} & \dfrac{t_{23}^2}{t_{24}t_{25}} & 0 \\ 0 & \dfrac{t_{35}}{t_{25}} & \dfrac{t_{23}}{t_{25}} & 0 \\ 0 & \dfrac{t_{45}}{t_{25}} & \dfrac{t_{34}}{t_{25}} & 0 \\ 0 & \dfrac{t_{45}^2}{t_{25}t_{35}} & \dfrac{t_{24}t_{45}}{t_{25}t_{35}} + \dfrac{t_{34}t_{46}}{t_{35}t_{36}} & \dfrac{t_{34}^2}{t_{35}t_{36}} \end{pmatrix} \begin{pmatrix} \vec{Z}_i \\ \vec{Z}_{i+1} \\ \vec{Z}_{i+2} \\ \vec{Z}_{i+3} \end{pmatrix}.$$

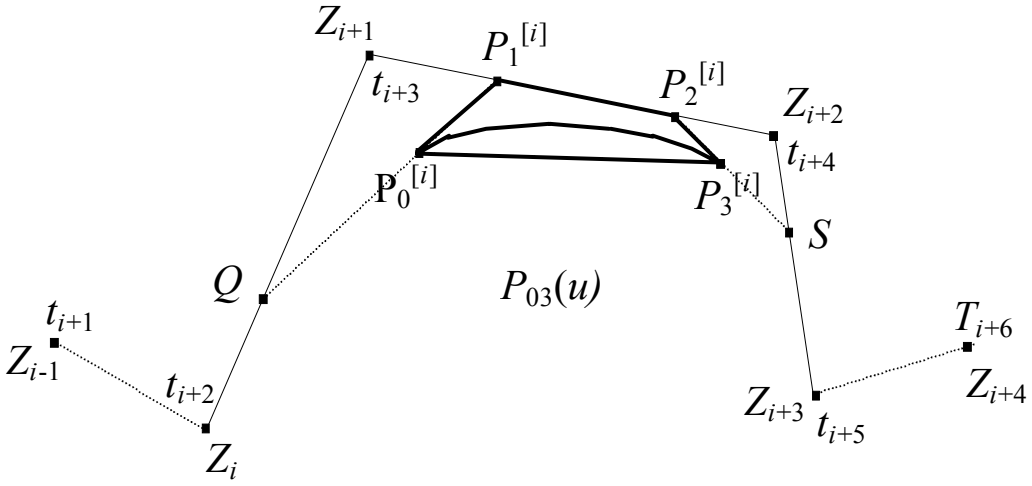


Рис. 53. Пример сегмента B-сплайна третьего порядка.

Вычислительная схема (рис. 53) может быть реализована также следующими формулами:

$$\vec{Q} = \vec{Z}_i \frac{t_{34}}{t_{14}} + \vec{Z}_{i+1} \frac{t_{13}}{t_{14}}, \qquad \vec{P}_1^{[i]} = \vec{Z}_{i+1} \frac{t_{35}}{t_{25}} + \vec{Z}_{i+2} \frac{t_{23}}{t_{25}},$$

$$\vec{P}_0^{[i]} = \vec{Q} \frac{t_{34}}{t_{15}} + \vec{P}_1^{[i]} \frac{t_{23}}{t_{25}}, \qquad \vec{P}_2^{[i]} = \vec{Z}_{i+1} \frac{t_{45}}{t_{25}} + \vec{Z}_{i+2} \frac{t_{24}}{t_{25}},$$

$$\vec{S} = \vec{Z}_{i+2} \frac{t_{46}}{t_{36}} + \vec{Z}_{i+3} \frac{t_{34}}{t_{36}}, \qquad \vec{P}_3^{[i]} = \vec{P}_2^{[i]} \frac{t_{45}}{t_{35}} + \vec{S} \frac{t_{34}}{t_{35}},$$

где $t_{km} \equiv t_{i+m} - t_{i+k}$, i - индекс сегмента сплайна, <Z_i> - опорные точки сплайна, <t_i> - узловой вектор сплайна, <$P^{[i]}$> - опорные точки i-того сегмента (кривой Безье). Если $t_{km} \neq 0$, то в точках склеивания сегментов будут равны все производные сплайна. В противном случае, некоторые производные будут различны, и сплайн-линия окажется менее гладкой, возможны даже изломы. Все приведенные алгоритмы и матрицы пересчета получены нами в предположении, что все $t_{km} \neq 0$.

Таким образом, мы показали, что неравномерный B-сплайн также может быть представлен набором сопряженных кривых Безье, и предложили алгоритм для пересчета параметров сплайна в наборы опорных точек его сегментов (кривых Безье). На рис. 52, 53 приведены расчетные схемы для нахождения вершин Безье-сегментов для сплайнов второй и третьей степеней.

5.6. Рациональная кривая Безье

Математически рациональная кривая Безье задается следующим уравнением:

$$\vec{R}_n(t) = \frac{\sum_{i=0}^{n} w_i B_i^n(t) \vec{P}_i}{\sum_{i=0}^{n} w_i B_i^n(t)}, \qquad t \subseteq [0,1],$$

где $\{\vec{P}_0,...,\vec{P}_n\}$ - управляющие вершины многоугольника, задающего кривую Безье, $\{w_0,...,w_n\}$ - весовые коэффициенты $(w_i \geq 0, \sum_n w_i \neq 0)$, задающие форму кривой, $B_i^n(t) = C_n^i (1-t)^{n-i} t^i$ - полином Бернштейна, $n+1$- количество управляющих вершин кривой.

Геометрические свойства рациональной кривой Безье близки к свойствам стандартной элементарной кривой Безье, а если все весовые коэффициенты <w_i> равны между собой, то уравнение $\vec{R}_n(t)$ будет эквивалентно уравнению $\vec{P}_n(t)$, и графики обоих кривых будут совпадать. Отметим свойства рациональной кривой Безье.

- Кривая начинается в вершине $\vec{P}_0$ и заканчивается в вершине $\vec{P}_n$.

- Касательные к кривой в ее начальной и конечной точках совпадают с направлением инцидентных этим точкам ребер $\vec{P}_{01}$ и $\vec{P}_{n-1,n}$:

$$\dot{\vec{R}}(0) = n\frac{w_1}{w_0}(\vec{P}_1 - \vec{P}_0), \qquad \dot{\vec{R}}(1) = n\frac{w_{n-1}}{w_n}(\vec{P}_n - \vec{P}_{n-1}).$$

- Кривая лежит в выпуклой оболочке, порожденной массивом управляющих вершин $< \vec{P}_i >$.
- В случае, если все управляющие вершины компланарны, кривая Безье лежит в плоскости, инцидентной вершинам.
- Форма кривой определяется не только управляющими вершинами $< \vec{P}_i >$, но и набором весовых коэффициентов $< w_i >$.
- Изменение положения хотя бы одной вершины или весового коэффициента приводит к полному изменению формы кривой.
- Рациональная кривая Безье второго порядка задает коническую дугу.

Аналогично тому, как B-сплайны могли быть представлены набором сопряженных стандартных кривых Безье, существует тип сплайнов, которые могут быть представлены набором сопряженных рациональных кривых Безье. Такие сплайны называются рациональными сплайнами (NURB-сплайнами).

Алгоритм пересчета NURB-сплайна второго порядка, называемого также коническим сплайном, в соответствующий ему набор конических дуг был рассмотрен ранее.

5.7. Кривые Эрмита и их применение

Математически кривая Эрмита является полиномом третьего порядка и задается уравнением

$$\vec{R}(t) = \vec{P}_0 H_0^3(t) + \vec{R}_1 H_1^3(t) + \vec{R}_2 H_2^3(t) + \vec{P}_3 H_3^3(t), \qquad t \subseteq [0,1],$$

где $\{\vec{P}_0, \vec{P}_3\}$ - начальная и конечная точки кривой, $\{\vec{R}_1, \vec{R}_2\}$ - управляющие векторы кривой Эрмита, $\{H_k^3(t)\}$ - базовые полиномы Эрмита третьего порядка, $B_k^n(t) = C_n^k(1-t)^{n-k}t^k$ - полином Бернштейна,

$$H_0^3(t) = B_0^3(t) + B_1^3(t) = 2t^3 - 3t^2 + 1,$$
$$H_1^3(t) = \frac{1}{3} B_1^3(t) = t^3 - 2t^2 + t,$$
$$H_2^3(t) = -\frac{1}{3} B_2^3(t) = t^3 - t^2,$$
$$H_3^3(t) = B_2^3(t) + B_3^3(t) = -2t^3 + 3t^2.$$

Кривую Эрмита можно представить в форме Безье:

$$\vec{R}_3(t) = \vec{P}_0(1-t)^3 + 3(\vec{P}_0 + \frac{1}{3}\vec{R}_1)(1-t)^2 t + 3(\vec{P}_3 - \frac{1}{3}\vec{R}_2)(1-t)t^2 + \vec{P}_3 t^3.$$

Проиллюстрируем графически пересчет параметров кривой Эрмита $\{\vec{P}_0, \vec{R}_1, \vec{R}_2, \vec{P}_3\}$ в параметры кривой Безье $\{\vec{P}_0, \vec{P}_1, \vec{P}_2, \vec{P}_3\}$ (рис. 54).

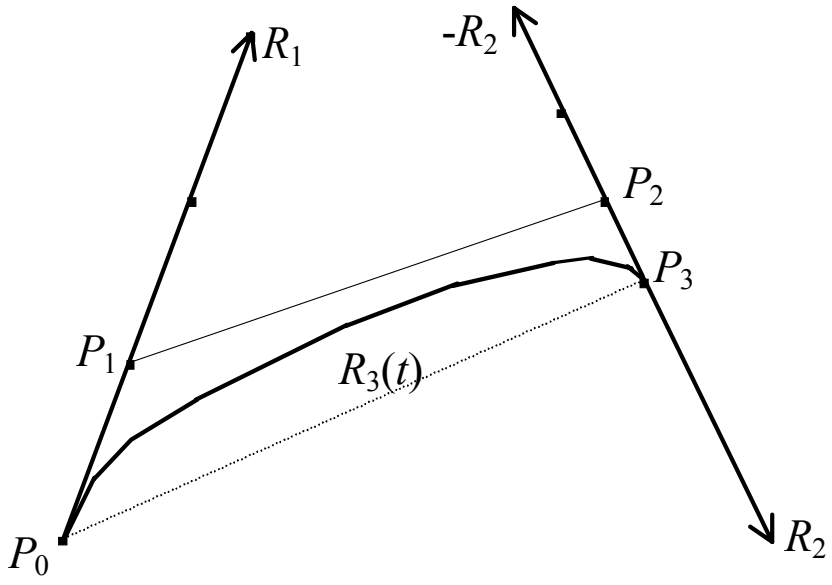


Рис. 54. Связь между управляющими параметрами кривой Эрмита и кривой Безье третьего порядка

Кривые Эрмита удобно использовать для интерактивного проектирования интерполирующих кривых. Предположим, что мы имеем набор точек в пространстве, через которые нужно провести гладкую линию. Составная кривая Эрмита для этого случая будет выглядеть так, как это показано на рис. 55.

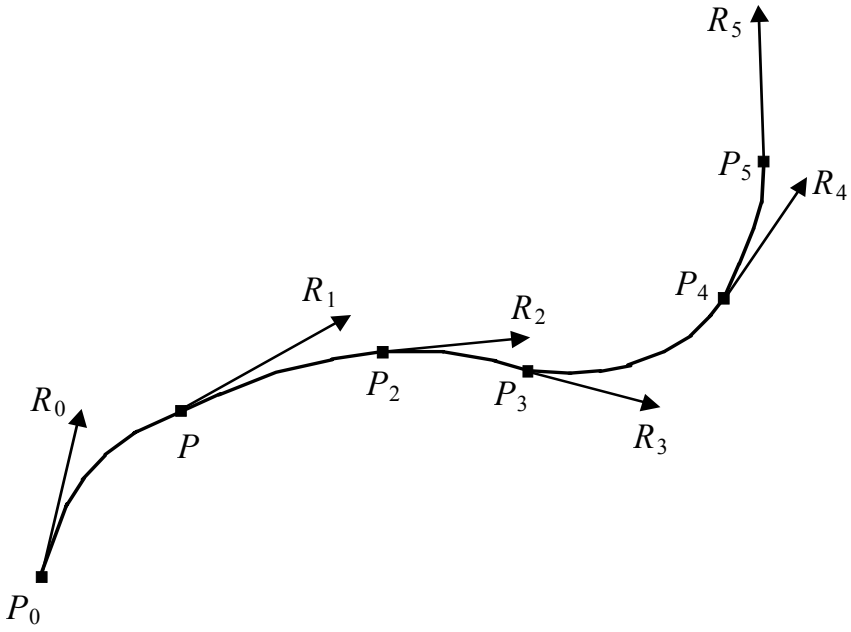


Рис. 55. Локальная интерполяция точек с помощью кривых Эрмита

6. Задание поверхностей в компьютерной графике

Основным 3D графическим примитивом в компьютерной графике на сегодняшний день является треугольник, задаваемый координатами его вершин. Как правило, для каждой вершины задаются ее положение в пространстве $\{X,Y,Z\}$, цвет $\{R,G,B\}$ или координаты в текстуре $\{S,T\}$. Если тонирование предполагается выполнять по методу Фонга [23], то с каждой вершиной связывается единичный вектор, называемый нормалью, хотя, строго говоря, он не обязательно должен являться геометрической нормалью к поверхности треугольника.

Для задания неплоских гладких поверхностей используются различные методы описания поверхностей параметрическими функциями от двух параметров. Сложные поверхности собираются из отдельных кусков (patch), каждый из которых является полиномом. Такой полином можно тонировать (растрировать) двумя способами. Во-первых, его можно представить интерполирующим многогранником, каждая элементарная грань которого будет являться треугольником или прямоугольником. Другой подход состоит в растрировании семейства близко лежащих линий, для генерации которых можно использовать аппаратно реализованный генератор кривых. Второй подход, как правило, не применяется, поскольку простые

видеокарты не способны аппаратными средствами генерировать кривые Безье.

Наиболее употребительными кусочными функциями являются пэтчи Безье и пэтчи Кунса. Рассмотрим методы их задания и использования.

6.1. Пэтчи Безье, их свойства

Метод Безье для проектирования кривых можно обобщить для проектирования поверхностей. Если к двумерному набору опорных точек поверхности $< \vec{P}_{m,n} >$ применить метод декартова произведения, то получим следующее выражение:

$$\vec{P}_{m,n}(s,t) = \sum_{i=0}^{m}\sum_{j=0}^{n} \vec{P}_{ij} C_m^i (1-s)^{m-i} s^i C_n^j (1-t)^{n-j} t^j , \qquad s,t \subseteq [0,1].$$

Именно это выражение мы в дальнейшем будем иметь в виду, когда будем упоминать пэтч Безье. Следует, однако, отметить, что имеется еще один метод получения двухпараметрической функции на основе кривых Безье. Этот метод основан на использовании булевой суммы вместо декартова произведения. Математически такой пэтч задается формулой

$$\vec{Q}_{m,n}(t,s) = \sum_{i=0}^{m} \vec{P}_{i,0} B_m^i(s) + \sum_{j=0}^{n} \vec{P}_{0,j} B_n^j(t) - \sum_{i=0}^{m}\sum_{j=0}^{n} \vec{P}_{ij} B_m^i(s) B_n^j(t).$$

В дальнейшем мы не будем использовать этот способ построения пэтча. Он ничем не хуже своего декартова аналога, но в практике как-то не прижился, возможно, из за его недостаточной наглядности и излишней сложности.

Проиллюстрируем геометрическое задание пэтча Безье на примере бикубического пэтча $(m = n = 3)$ (рис. 56). Для задания пэтча Безье достаточно задать набор $m \times n$ точек в пространстве. Такой набор задает полигональную сетку (рис. 56).

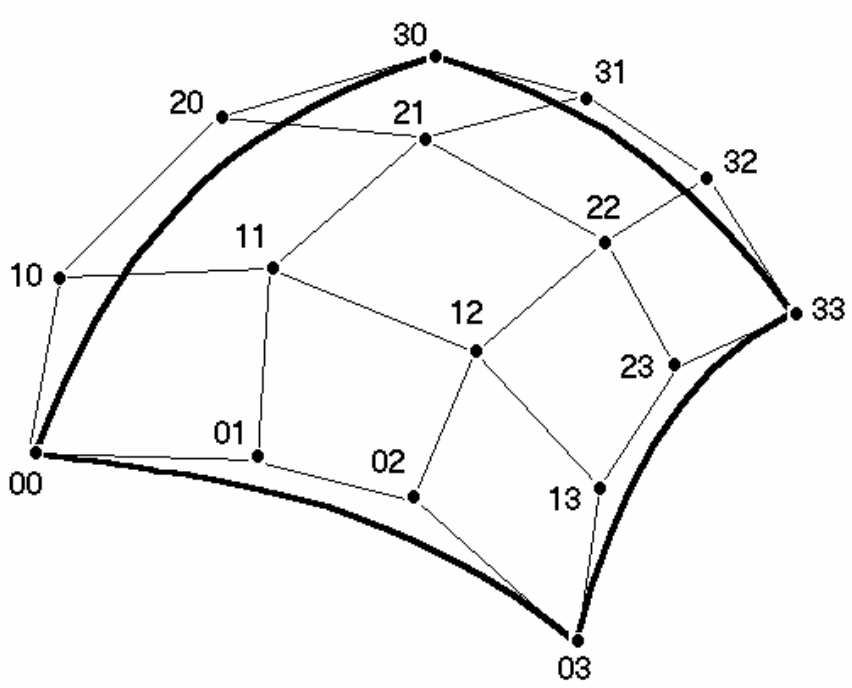


Рис. 56. Задание бикубического пэтча Безье полигональной сеткой

Чтобы обеспечить гладкое сопряжение двух смежных пэтчей, необходимо, чтобы смежные ребра были коллинеарны. На рис. 57. показано гладкое сопряжение двух бикубических пэтчей Безье.

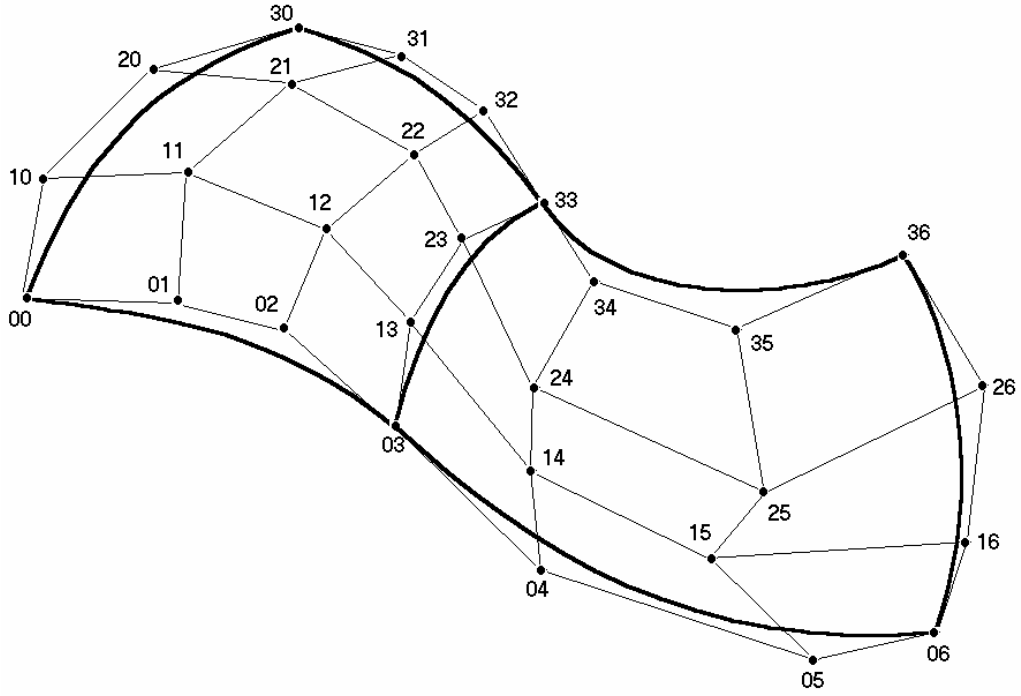


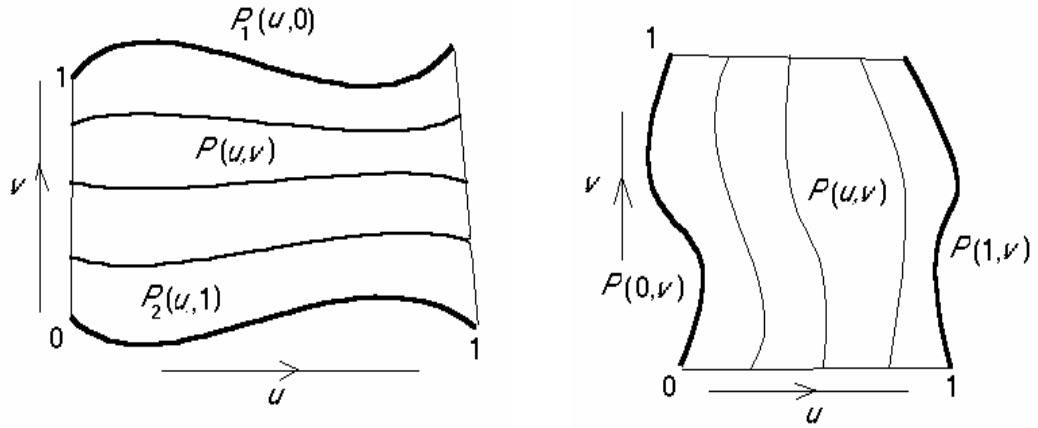
Рис. 57. Пример гладкого сопряжения двух пэтчей Безье

Для гладкого сопряжения двух пэтчей, как это показано на рис. 57, необходимо чтобы были коллинеарны следующие тройки точек: (02,03,04), (12,13,14),(22,23,24),(32,33,34). В этом случае при интерполяции возникнет визуальная иллюзия гладкой поверхности многогранника.

Если для аппроксимации поверхности используется двухпараметрический B-сплайн, то его кусочные сегменты будут являться пэтчами Безье. Для управления формой пэтча Безье достаточно изменять положение его опорных точек.

6.2. Плазовые поверхности

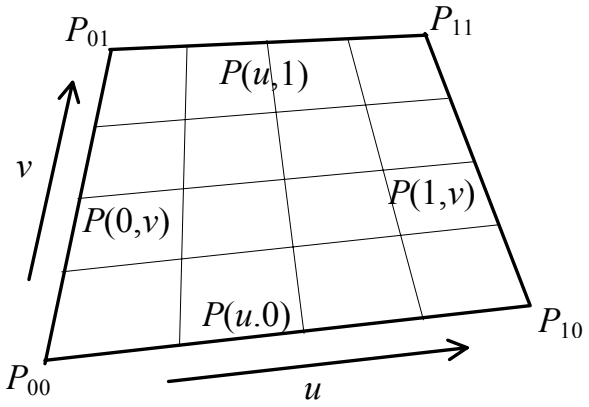
В инженерной практике кораблестроения до сих пор применяется способ задания поверхности, называемый плазовым. Плазовые поверхности получают методом линейной интерполяции граничных кривых. Обобщение этого метода известно как поверхности (куски, пэтчи) Кунса. Опуская описание механической реализации плазовой поверхности, приведем ее математическое задание (рис. 58). Параметры u,v, в свою очередь, могут быть монотонными параметрическими функциями, изменяющимися в диапазоне от 0 до 1.



$$\vec{P}(u,v) = \vec{P}_1(u,0)(1-v) + \vec{P}_2(u,1)v \qquad \vec{P}(u,v) = \vec{P}(0,v)(1-u) + \vec{P}(1,v)u$$

Рис. 58. Билинейная интерполяция пары граничных кривых

Важным частным случаем плазовой поверхности является билинейная поверхность. Пример такой поверхности приведен на рис. 59.



$$\vec{S}(u,v) = \left[\vec{P}_{00}(1-u) + \vec{P}_{10}u\right](1-v) + \left[\vec{P}_{01}(1-u) + \vec{P}_{11}u\right]v$$

Рис. 59. Билинейная поверхность

Если принять, что функции $\vec{P}(0,u), \vec{P}(1,u), \vec{P}(v,0), \vec{P}(v,1)$ нам известны, то можно сконструировать пэтч, известный как поверхность Кунса. Математически пэтч Кунса описывается следующей формулой:

$$\vec{S}(u,v) = [\vec{P}(u,0)(1-v) + \vec{P}(u,1)v] + [\vec{P}(0,v)(1-u) + \vec{P}(1,v)u] -$$

$$- \left\{[\vec{P}_{00}(1-u)] + \vec{P}_{10}u](1-v) + [\vec{P}_{01}(1-u) + \vec{P}_{11}u]v\right\}.$$

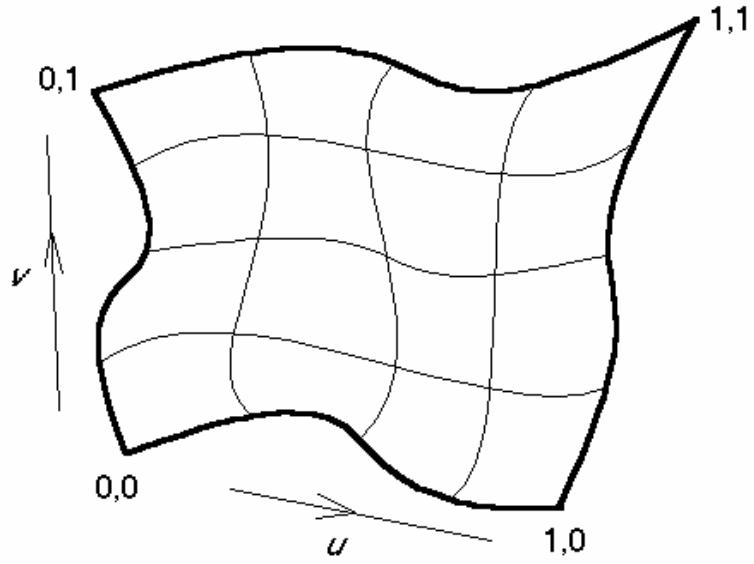


Рис. 60. Пэтч (кусок) Кунса

Геометрически пэтч Кунса выглядит как криволинейный четырехугольник, похожий на пэтч Безье, однако его опорными функциями могут быть не только кривые Безье, но и любые параметрические функции $\vec{P}(u,0), \vec{P}(u,1), \vec{P}(0,v), \vec{P}(1,v)$. Существуют и более сложные конструкции пэтча Кунса, учитывающие и направление нормалей в вершинах пэтча.

6.3. Треугольный пэтч Безье

Кроме криволинейных четырехугольников, существует метод для задания криволинейного треугольного пэтча. Математически треугольный пэтч Безье задается формулой

$$P(u,v,w) = \underbrace{\sum_{i=0}^{m}\sum_{j=0}^{m}\sum_{k}^{m} \vec{P}_{ijk} B_m^i(u) B_m^j(v) B_m^k(w)}_{i+j+k=m;\, u+v+w=1;\, u,v,w\subseteq[0,1]}.$$

Приведем в качестве примера характеристическую сетку треугольного пэтча четвертого порядка с треугольными элементами разбиения (рис. 61).

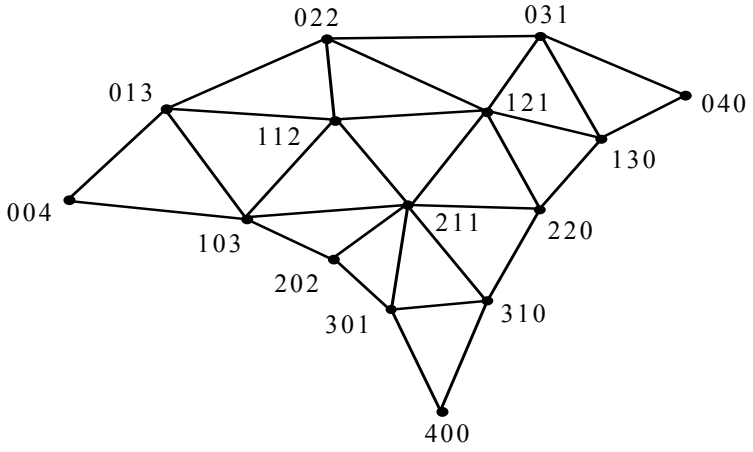


Рис. 61. Задание треугольного пэтча 4-го порядка

Рис. 60 наглядно показывает, что часть управляющих точек находится внутри пэтча так же, как в случае прямоугольного пэтча, однако размещение этих точек и их количество не так очевидно. Для того, чтобы обеспечить касание примыкающих треугольных элемен-

тов по линии примыкания, необходимо, чтобы грани характеристической сетки по обе стороны линии сшивки были компланарны, как это проиллюстрировано на рис. 62.

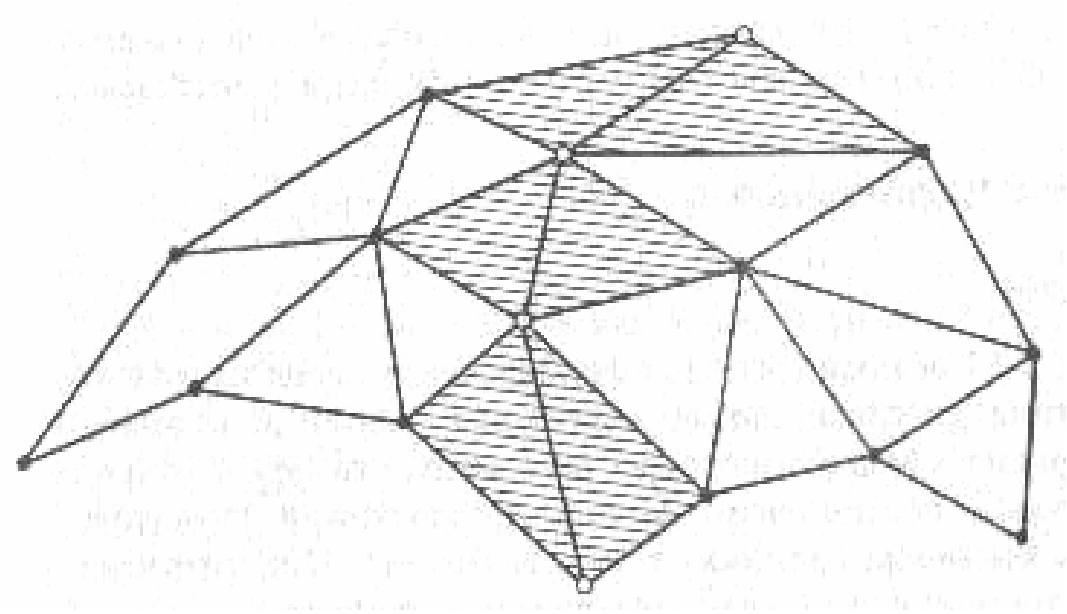


Рис. 62. Компланарность заштрихованных треугольников - условие гладкого сопряжения двух треугольных пэтчей третьего порядка

Для вычисления положения точки по ее известным координатам u,v,w удобно воспользоваться следующим алгоритмом:

```
for l:=1 to m do
{
      for j:=0 to m-l do
      {
            for i:=0 to m-l-j do
            {
                  k:=m-l-i-j;
                  P(l)ijk = uP(l-1)i+1,j,k + vP(l-1)i,j+1,k + wP(l-1)i,j,k+1;
            }
      }
}
```

Работа этого итеративного алгоритма для пэтча третьего порядка продемонстрирована на рис. 63.

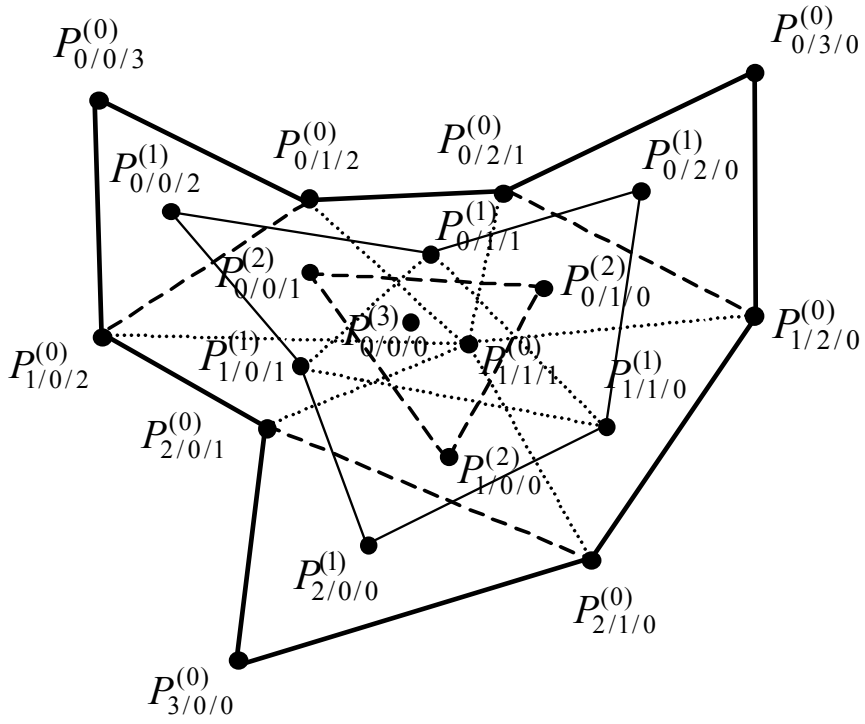


Рис. 63. Геометрическая иллюстрация расчета положения точки поверхности, заданной однородными координатами u, v, w, треугольного пэтча 3-го порядка

Расчет вспомогательных опорных точек выполняется по следующим формулам.

$$\vec{P}^{(1)}_{200} = \vec{P}^{(0)}_{300}u + \vec{P}^{(0)}_{210}v + \vec{P}^{(0)}_{201}w, \qquad \vec{P}^{(1)}_{110} = \vec{P}^{(0)}_{210}u + \vec{P}^{(0)}_{120}v + \vec{P}^{(0)}_{111}w,$$
$$\vec{P}^{(1)}_{020} = \vec{P}^{(0)}_{120}u + \vec{P}^{(0)}_{030}v + \vec{P}^{(0)}_{021}w, \qquad \vec{P}^{(1)}_{011} = \vec{P}^{(0)}_{111}u + \vec{P}^{(0)}_{021}v + \vec{P}^{(0)}_{012}w,$$
$$\vec{P}^{(1)}_{002} = \vec{P}^{(0)}_{102}u + \vec{P}^{(0)}_{012}v + \vec{P}^{(0)}_{003}w, \qquad \vec{P}^{(1)}_{101} = \vec{P}^{(0)}_{201}u + \vec{P}^{(0)}_{111}v + \vec{P}^{(0)}_{102}w,$$
$$\vec{P}^{(2)}_{100} = \vec{P}^{(1)}_{200}u + \vec{P}^{(1)}_{110}v + \vec{P}^{(1)}_{101}w,$$
$$\vec{P}^{(2)}_{010} = \vec{P}^{(1)}_{110}u + \vec{P}^{(1)}_{020}v + \vec{P}^{(1)}_{011}w,$$
$$\vec{P}^{(2)}_{001} = \vec{P}^{(1)}_{101}u + \vec{P}^{(1)}_{011}v + \vec{P}^{(1)}_{002}w,$$
$$\vec{P}^{(3)}_{000} = \vec{P}^{(2)}_{100}u + \vec{P}^{(2)}_{010}v + \vec{P}^{(2)}_{001}w \quad \text{- результат } \vec{P}(u,v,w).$$

Отметим, что границы треугольного пэтча являются кривыми Безье:

$$\vec{P}_3(u) = \vec{P}^{(0)}_{300}(1-u)^3 + 3\vec{P}^{(0)}_{210}(1-u)^2u + 3\vec{P}^{(0)}_{120}(1-u)u^2 + \vec{P}^{(0)}_{030}u^3,$$
$$\vec{P}_3(v) = \vec{P}^{(0)}_{030}(1-v)^3 + 3\vec{P}^{(0)}_{021}(1-v)^2v + 3\vec{P}^{(0)}_{012}(1-v)v^2 + \vec{P}^{(0)}_{003}v^3,$$
$$\vec{P}_3(w) = \vec{P}^{(0)}_{003}(1-w)^3 + 3\vec{P}^{(0)}_{102}(1-w)^2w + 3\vec{P}^{(0)}_{201}(1-w)w^2 + \vec{P}^{(0)}_{300}w^3.$$

7. Геометрические преобразования в 3D пространстве

7.1. Основные сведения

Мы рассмотрели ряд кривых и поверхностей, используемых в компьютерной графике, и во всех случаях основным параметром, определяющим форму кривой или поверхностей, был набор опорных точек - вершин управляющего многоугольника или многогранника. Для выполнения геометрических преобразований – масштабирования, смешения, поворота заданного объекта необходимо подвергнуть этим преобразованиям каждую опорную точку данного объекта.

В аналитической геометрии предлагался набор матричных преобразований для выполнения масштабирования точки относительно центра координат ее смещения и поворота относительно одной из осей координат.

Масштабирование

$$x' = x \cdot S_x, \qquad y = y \cdot S_y, \qquad z' = z \cdot S_z,$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \vec{P}' = S \cdot \vec{P}.$$

Смещение

$$x' = x + T_x, \qquad y' = y + T_y, \qquad z' = z + T_z,$$

$$(x', y', z')^T = (x, y, z)^T + (T_x, T_y, T_z)^T, \quad \vec{P}' = \vec{P} + \vec{T}.$$

Поворот вокруг оси X

$$x' = x, \qquad y' = y \cdot \cos\varphi + z \cdot \sin\varphi, \qquad z' = -y \cdot \sin\varphi + z \cdot \cos\varphi,$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & -\sin\varphi & \cos\varphi \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \vec{P}' = R_x(\varphi) \cdot \vec{P}.$$

Поворот вокруг оси **Y**

$$x' = x \cdot \cos\psi - z \cdot \sin\psi, \quad y' = y, \quad z' = x \cdot \sin\psi + z \cdot \cos\psi,$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\psi & 0 & -\sin\psi \\ 0 & 1 & 0 \\ \sin\psi & 0 & \cos\psi \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \vec{P}' = R_y(\psi) \cdot \vec{P}.$$

Поворот вокруг оси **Z**

$$x' = x \cdot \cos\chi + y \cdot \sin\chi, \quad y' = -x \cdot \sin\chi + \cos\chi, \quad z' = z,$$

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \qquad \vec{P}' = R_z(\chi) \cdot \vec{P}.$$

Матричная запись геометрических преобразований очень удобна, поскольку позволяет задать геометрическое преобразование в виде композитной матрицы, полученной перемножением матриц элементарных преобразований (масштабирования и поворота). Однако преобразование смещения не является матричной операцией, что не позволяет в самом общем случае получить матричное преобразование на основе матриц размерности 3x3. Этот недостаток можно обойти, если задавать координаты точки однородными координатами.

Координаты точки $\{x, y, z\}$ можно задать четверкой чисел $\{X, Y, Z, h\}$, где $h \neq 0$. Связь между однородными и неоднородными координатами точки определяется следующими соотношениями

$$x = \frac{X}{h}, \qquad y = \frac{Y}{h}, \qquad z = \frac{Z}{h}.$$

Очевидно, что четверку $\{X, Y, Z, h\}$, однозначно задающую положение точки, также можно считать ее координатами. Однородные координаты замечательны тем, что с их помощью можно задать все геометрические преобразование в виде матричных операций и получить, таким образом, композитное преобразование в виде одной матрицы. Забегая вперед, отметим, что преобразование проецирования точки на экранную плоскость также выполняется умножением на некоторую матрицу.

Приведем матрицы элементарных геометрических преобразований точки, заданной ее однородными координатами $\{x, y, z, 1\}$.

Масштабирование:
$$S = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Смещение:
$$T = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Поворот вокруг оси X:
$$R_x(\varphi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Поворот вокруг оси Y:
$$R_y(\psi) = \begin{pmatrix} \cos\psi & 0 & \sin\psi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\psi & 0 & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Поворот вокруг оси Z:
$$R_z(\chi) = \begin{pmatrix} \cos\chi & -\sin\chi & 0 & 0 \\ \sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Из приведенных соотношений видно, что композиция нескольких преобразованиях может быть задана одной матрицей ***G***

$$G = \begin{pmatrix} g_{11} & g_{12} & g_{13} & 0 \\ g_{21} & g_{22} & g_{23} & 0 \\ g_{31} & g_{32} & g_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Приведем пример композитного преобразования: промасштабировать относительно точки $\{dx, dy, dz\}$ и повернуть относительно осей X и Y, соответственно на углы φ и ψ точку с координатами $\{x,y,z\}$.

$$\mathbf{G} = \mathbf{T}(dx,dy,dz,1)\ \mathbf{R_y}(\psi)\ \mathbf{R_x}(\varphi)\ \mathbf{S}(s_x,s_y,s_z,1)\ \mathbf{T}(-dx,-dy,-dz,1);$$
$$P' = \mathbf{G}*P.$$

Все приведенные соотношения справедливы, только если мы работаем в правой системе координат, как показано на рис. 64.

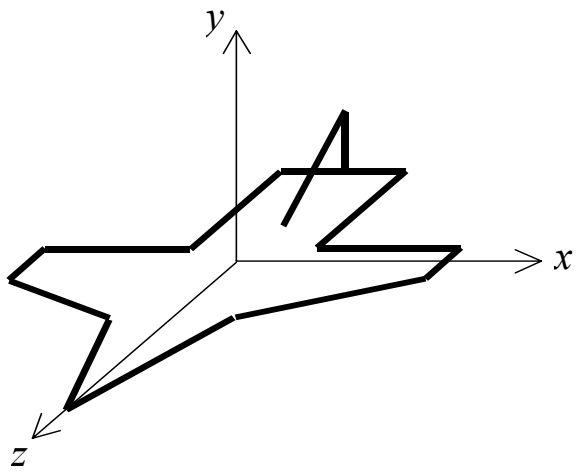


Рис. 64. Правая система координат – ориентация осей

Ряд графических библиотек и некоторые справочники по умолчанию предполагают, что рабочая система координат - левая. Для перехода от правой системы координат к левой следует транспонировать все приведенные выше матрицы элементарных геометрических (S,T,R_x,R_y,R_z) преобразований и вместо умножения вектора-столбца на матрицу $P'=G*P$ выполнять умножение матрицы на вектор-строку $P'^{T}=P^{T}*G'$, а при нахождении матрицы композитного преобразования перемножать матрицы не справа налево $(T(R_y(R_x(ST^{-1}))))$, а слева направо $((((T'^{-1}S')\ R'_x)\ R'_y)\ T')$.

7.2. Проецирование на экранную плоскость

В конечном счете, целью синтеза 3D объекта является представление его на плоской поверхности дисплея или на бумаге, если вы используете плоттер. Для этого следует выполнить проецирование объекта на экранную плоскость, которую иногда называют картинной плоскостью. Существует несколько методов построения раз-

личных видов проекций. Изучением проективных преобразований и методов их построения занимается начертательная геометрия. В компьютерной графике нашли широкое применение лишь два типа проективных преобразований - параллельное и центральное проецирования. Для нас наибольший интерес представляет изучение центрального проецирования, поскольку именно с его помощью получаются реалистические изображения трехмерных сцен.

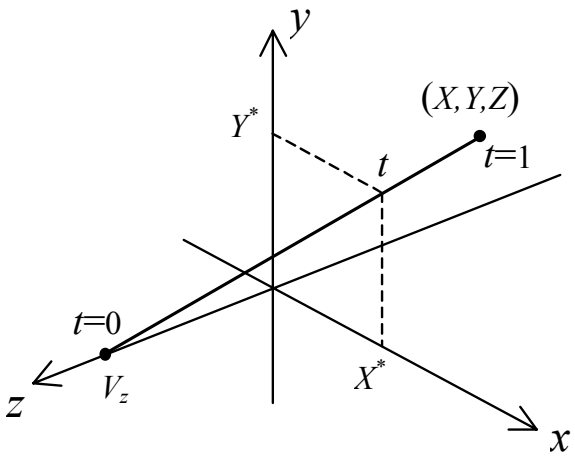


Рис. 65. Центральное проецирование точки (X,Y,Z) на экранную плоскость XY

Предположим (рис. 65), что ось Z проходит через глаз наблюдателя $(0,0,V_z)$. Тогда экранные координаты (X^*,Y^*) наблюдаемой точки (X,Y,Z) можно найти следующим образом. Составим параметрическое уравнение для отрезка луча $L(t)=V(1-t)+Pt$, выходящего из точки $(0,0,V_z)$ и проходящего через проецируемую на экран точку $P(X,Y,Z)$. Из рис. 64 видно, что в точке пересечения лучом экранной плоскости (X^*,Y^*) координата Z^* равна нулю $V_z(1-t')+Zt'=0$, откуда находим значение параметра t', соответствующего точке пересечения луча с экранной плоскостью:

$$t' = \frac{1}{1-\frac{Z}{V_z}}.$$

Тогда экранные координаты (X^*,Y^*) точки (X,Y,Z) равны

$$X^* = \frac{X}{1-\frac{Z}{V_z}}, \qquad Y^* = \frac{Y}{1-\frac{Z}{V_z}}.$$

Полученные выражения можно представить как результат матричной операции

$$\begin{pmatrix} X \\ Y \\ 0 \\ 1-\dfrac{Z}{V_z} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/V_z & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}.$$

После выполнения матричного преобразования следует разделить все компоненты полученного вектора на его последнюю координату $1-\dfrac{Z}{V_z}$, после чего мы получаем экранные координаты (X^*,Y^*).

Таким образом, мы показали, что проективное преобразование также является матричным преобразованием, и весь процесс геометрического преобразования объекта и его представление в виде точек на экранной плоскости выполняется одним умножением вектора на матрицу. Нередко объекты состоят из многих тысяч граней, и геометрические преобразования, применяемые к каждой вершине грани, потребляют заметную долю вычислительных ресурсов графического процессора.

7.3. Отсечение отрезка в экранной плоскости (алгоритм Сазерленда-Кохена)

Часто размеры рисунка в экранной плоскости значительно превосходят размер поля вывода. Для того, чтобы избежать ненужных вычислительных затрат на растрирование объектов, которые не будут отображаться на экране, следует обработать отображаемый объект, удалив из него все графические примитивы, которые полностью лежат вне поля вывода. Примитивы, частично лежащие в поле вывода, следует подвергнуть клипированию, то есть модифицировать данный примитив таким образом, чтобы он описывал только свою видимую часть.

Поскольку, как было показано выше, большинство линий задается многоугольниками, то отсечение ребер многоугольников относительно прямоугольного поля вывода является весьма актуальной задачей, эффективное решение которой способно заметно повысить скорость отрисовки сцены.

Рассмотрим популярный алгоритм Сазерленда-Кохена, выполняющий отсечение отрезка прямоугольным окном. Прямоугольное окно, стороны которого параллельны координатным осям, разбивает экранную плоскость на девять областей (рис. 66), которые кодируются следующим образом. Код области - четырех битный код, в котором

бит **0** означает, что точка лежит левее прямоугольника,
бит **1** означает, что точка лежит выше прямоугольника,
бит **2** означает, что точка лежит правее прямоугольника,
бит **3** означает, что точка лежит ниже прямоугольника.

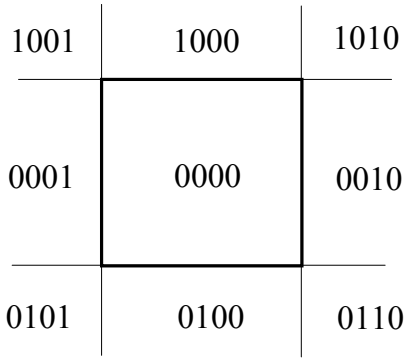


Рис. 66. Кодирование областей экранной плоскости

Приведем С-код для расчета кода области.

```
int OutCode(int x, int y,  // Координаты клиппируемой точки.
      int X1, int Y1,// Коорд. левой верхней вершины прямоуг-ка
      int X2, int Y2)// Коорд. Правой нижней вершины прямоуг-ка
{
      int code = 0;  // Код области в которой лежит точка(x,y).
      if(x < X1) code |= 0x01;
      if(y < Y1) code |= 0x02;
      if(x > X2) code |= 0x04;
      id(x > Y2) code |= 0x08;
      return code;
}
```

Приведем С-код [22] для клипирования и отрисовки вектора (отрезка прямой линии).

```
void Swap(int& a, int &b)// Вспомогательная программа для
                         //  перестановки данных.
{
      int c = a;
      a = b; b = c;
}

void ClipLine(int x1, int y1,    // Координаты начала вектора.
```

```
      int x2, int y2,  // Координаты конца вектора.
      int X1, int Y1,  // Коорд. левой верхней вершины прям-ка
      int X2, int Y2)  // Коорд. правой нижней вершины прям-ка
{
      int code1 = OutCode(x1,y1,X1,Y1,X2,Y2); //Код начала в-ра
      int code2 = OutCode(x2,y2,X1,Y1,X2,Y2); //Код конца в-ра
      BOOL inside = (code1 | code2) == 0; //Весь вектор внутри?
      BOOL outside = (code1 & code2)!=0; //Весь вектор снаружи?

      while(!outside && !inside)
      {
            if(code1==0)//Поменять местами начало и конец в-ра.
            {
                  Swap(x1,x2); Swap(y1,y2); Swap(code1,code2);
            }

            if(code1 & 0x01)      // Clip left
            {
                  y1 += (long)(y2-y1)*(X1-x1)/(x2-x1); x1 = X1;
            }
            else if(code1 & 0x02) // Clip above
            {
                  x1 += (long)(x2-x1)*(Y1-y1)/(y2-y2); y1 = Y1;
            }
            else if(code1 & 0x04) // Clip right

            {
                  r1 += (long)(y2-y1)*(X2-x1)/(x2-x1); x1 = X2;
            }
            else if(code1 & 0x08) // Clip below
            {
                  x1 += (long)(x2-x1)*(Y2-y1)/(y2-y1); y1 = Y2;
            }

            code1 = OutCode(x1, y1, X1, Y1, X2, Y2);
            code2 = OutCode(x2, y2, X1, Y1, X2, Y2);
            inside   = (code1  |  code2) == 0;
            outside = (code1 & code2) != 0;
      }
      line(x1, y1, x2, y2); // Отрисовка в-ра (x1,x2) (y1,y2)
}
```

Предложенный выше алгоритм Сазерленда-Кохена можно элементарно обобщить на случай отсечения по каноническому видовому кубу. Задание такого куба показано на рис. 67.

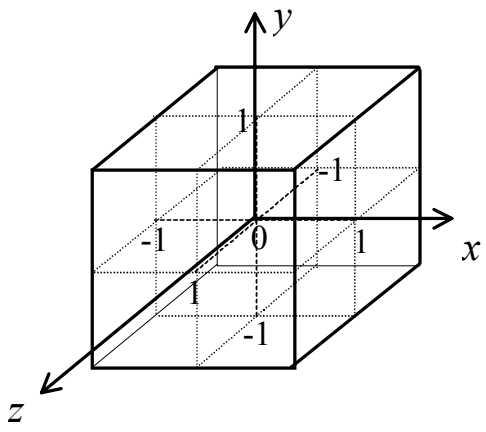


Рис. 67. Отсечение по каноническому видовому кубу

Этот куб разбивает 3D пространство на 6 областей, которые можно закодировать следующим образом:

бит **0** означает, что точка лежит выше куба, $y > 1$,
бит **1** означает, что точка лежит ниже куба, $y < -1$,
бит **2** означает, что точка лежит правее куба, $x > 1$,
бит **3** означает, что точка лежит левее куба, $x < -1$,
Бит **4** означает, что точка лежит за кубом, $z < -1$,
Бит **5** означает, что точка лежит перед кубом, $z > 1$.

Для нахождения точек пересечения вектора с гранями куба, представим вектор параметрическим уравнением

$$x = x_0(1-t)+x_1t, \qquad y = y_0(1-t)+y_1t, \qquad z = z_0(1-t)+z_1t,$$

где (x_0,y_0,z_0) - координаты начала вектора, (x_1,y_1,z_1) - координаты конца вектора, $t \subseteq [0,1]$- параметр. Зная грань, с которой пересекается вектор, находим значение t, соответствующее точке пересечения и затем рассчитываем по известным формулам остальные координаты точки пересечения. Например, предположим, что вектор пересекает верхнюю грань куба. Это значит, что $y=0$, откуда находим $t=(1-y_0)/(y_1-y_0)$, и, зная t, рассчитываем координаты X и Z:

$$x=x_0+(x_1-x_0)\,t, \qquad y=y_0+(y_1-y_0)\,t.$$

Остальные варианты рассчитываются аналогично. Вследствие некоторой громоздкости алгоритма мы не приводим его С-код алгоритма, но его можно найти в [24].

Отсечение по видовому кубу имеет практическое значение, поскольку существует последовательность геометрических преобразований, способная преобразовать пирамиду видимости общего положения в рассмотренный видовой куб (рис. 68). Применяя это преобразование к концам вектора, мы пересчитываем исходный вектор в некоторый вторичный вектор, который и подвергается клипированию предлагаемым алгоритмом. Полученные координаты концов отклипированного вектора подвергаются обратному геометрическому преобразованию, и таким образом находится видимая часть исходного вектора.

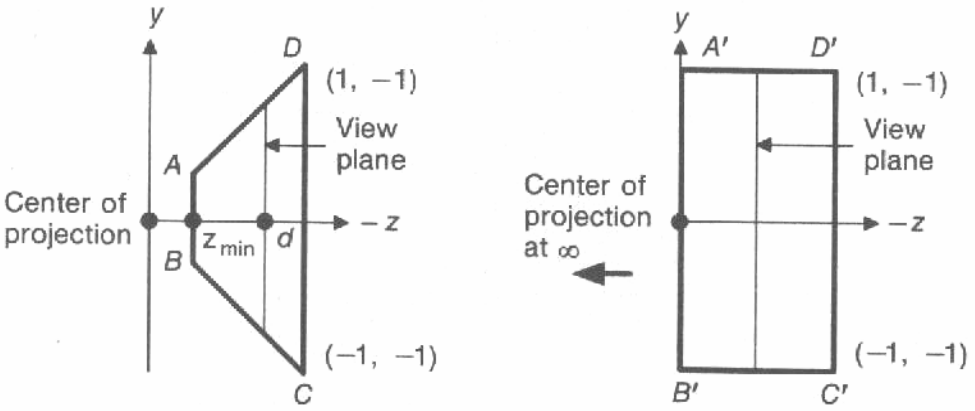


Рис. 68. Преобразование пирамиды видимости в канонический видовой куб [24]

Для других графических примитивов пока не предложено достаточно эффективных алгоритмов отсечения. Они либо отбрасываются целиком, для чего строится выпуклая оболочка по заданному набору опорных точек, либо, если примитив частично лежит в поле вывода, клипированию подвергается каждый его пиксель. Применение клипирования позволяет значительно повысить производительность графических систем, особенно для сцен, размеры которых значительно превышают размеры поля вывода.

8. Заключение

В данном пособии подробно рассмотрен круг проблем, связанных с растрированием линий единичной толщины. Основное внимание уделялось методикам синтеза алгоритмов растрирования. В основе каждой из методик лежит некоторая парадигма, реализация которой и приводит к созданию алгоритма растрирования (или семейства алгоритмов). В пособии описаны две такие парадигмы.

Первая - построение управляющей вычислительной структуры на основе конических интерполяторов. Такая структура реализует генератор лекальных кривых и может быть использована для растрирования кривых Безье невысокого порядка (не более 5). Поскольку в инженерной практике сплайны выше третьего порядка применяются редко, такой графический примитив имеет несомненный прикладной интерес. Для растрирования кривых более высокого порядка можно применить алгоритмы расщепления и понижения порядка. Предложен объективный критерий для применения понижения порядка кривой Безье.

Вторая – применение вычислительной структуры, реализующей, в простейшем ее варианте, алгоритм расщепления кривой Безье, известный также как алгоритм Кастелью. При некоторой модификации этого алгоритма его можно использовать для растрирования рациональных кривых Безье и конических дуг.

Большое внимание было уделено синтезу и исследованию свойств генератора конических дуг, поскольку этот графический примитив вывода включал в себя, как частные случаи, окружности, эллипсы и векторы, а также мог быть использован для визуализации конического сплайна.

Рассмотрены также алгоритмы пересчета одних форм в другие формы описания кривых. Например, сплайны представлялись набором кривых Безье, которые, в свою очередь, преобразовывались в лекальные кривые, которые и подлежали растрированию.

В данном пособии основное внимание было уделено математическому обоснованию рассматриваемых алгоритмов растрирования линий. За его рамками остались многие важные алгоритмы: генерация линий неединичной толщины, алгоритмы закраски и текстурирования поверхностей, удаление ступенчатого эффекта для текстурированных поверхностей и многие другие. В пособии рассматривались алгоритмы низкого уровня, подлежащие аппаратной реализации, а также методы их инициализации.

ЛИТЕРАТУРА

1. Bresenham J.E. Algorithm for Computer Control of Digital Plotter. // IBM System Journal. - 1965. - V. 4(1).- Ð.25-30.
2. Pitteway M. Algorithm for drawing ellipses or hyperbolae with a digital plotter. // Computer Journal. - 1966. - V. 10(3). - P. 282-289.
3. Botting R.J.,Plitteway M.L.V. Algorithm for drawing ellipses or hyperbolae with a digital plotter. // Computer Journal. - 1968. - V. 11 (1). - Ð. 120.
4. Bresenham J.E. A Linear Algorithm for Incremental Digital Display of Circular Arcs. // Communication of the ACM. - 1977. - V. 20(2). - P. 100-106.
5. P.E.Danlelson. Incremental curve generation. // IEEE Transaction on Computers. - 1970. - V.C-19. - P. 783-793.
6. A. W. Jordan, W.J.Lennon, W.J.Holm. An Improved Algorithm for Generation of Non-parametric Curves. // IEEE Transaction on Computers. - 1973. - V.C-22(12). - P. 1052-1060.
7. Yasuhlto Suenaga, Takahiko Kamae, Tomonori Kabayashi. A High Speed Algorithm for the Circular Arcs. // IEEE Trans. on Computers. - 1979. -V. C-28(10). - P. 728-736.
8. M.L.V.Plitteway. R.J.Bottlng. Integer circles etc. - Three move of Breseriham's algorithm. // Computer Graphics and Image Processing. -1974. - V.3. - P. 260-261.
9 B.K.P.Horn. Circle generators for display devices. //Computer Graphics, and Image Processing. -1976.-V.5.-P. 580-588.
10 N.I.Badler. Disk generators for a raster display device. // Computer, Graphics and Image Processing. - 1977. - V. 6. - P. 589-593.
11 W.L.Chung. On circle generation algorithms. // Computer Graphics and Image Processing. - 1977. - V.6. - P. 196-198.
12 M.Doros. Algorithms tor Generation to Discrete Circles, Rings and Disks. // Computer Graphics and Image Processing. - 1979. - V.10(4). - P. 366-371.
13 M.Plitteway, D.Watkinson. Bresenham's Algorithm with Gray Scale. // Communication of the ACM. - 1980. - V. 23(11).- P. 625-626.
14 G.Moller. Past digital vector and circle generator with binary rate multipliers. // Computer Graphics. - 1978.-V.12(4).- P. 81-91.
15 P.G.McCrea, P.W.Baker. On DDA circle generation for computer graphics. // IEEE Trans. on Computers. - 1975. - V. C-24. - P. 1109-1110.
16 P.C.Maxwell. P.W.Baker. The Generation of Polygons Representing Circles. Ellipses and Hyperbolas. // Computer Graphics end Image Processing. - 1979. - V.10. - P.84-93.

17 Chirikov S.V., Paltashev T.T. Integer algorithm for L-curve rasterization and application of L-curve for Bezier curve representation. // GRAFICON 93 (Conf.) Russia.
18 Chirikov S.V., Paltashev T.T., P.A.Balabko. New Method of Parametric Curve Rasterization and its Application for Fast Bezier Patch Shading. // GRAFICON 95 (Conf.) Russia.
19 Chirikov S.V., Paltashev T.T. , P.A.Balabko. New Method of Parametric Curve Rasterization and its Application for Fast Bezier Patch Shading. // WSCG'97 (Conf.) Czech Republic.
20 Д.Роджерс. Адгоритмические основы машинной графики. // М:Мир 1989.
21 Дж.Фоли, А.ванДам. Основы интерактивной машинной графики. т.1,2. // Москва:Мир 1985.
22 Е.В.Шикин, А.В.Боресков. Компьютерная графика. Динамика, реалистические изображения // Москва:"ДИАЛОГ-МИФИ",1995. – 288 с.
23 Е.В.Шикин, А.И.Плис. Кривые и поверхности на экране компьютера. Руководство по сплайнам для пользовалелей. // Москва:"ДИАЛОГ-МИФИ", 1996. – 240 с.
24 James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes. Computer graphics – principles and practice. // ADDISON-WESLEY PUBLISHING COMPANY. 1990.
25 Дж.Фоли, А.вэн Дэм. Основы интерактивной машинной графики. Т.1,2. Москва "Мир" 1985.

Сергей Васильевич Чириков
Алгоритмы компьютерной графики
(методы растрирования кривых)
Часть 1
Учебное пособие

Компьютерная верстка С.В.Чириков
Дизайн С.В.Чириков
Редакционно-издательский отдел СПб ГИТМО(ТУ)
Зав. РИО Н.Ф.Гусарова
Лизензия ИД N00408 от 05.11.99
Подписато к печати 06.12.00
Отпечатано на ризографе Заказ N

Тираж 150 экз.