2 БАЗОВЫЕ РАСТРОВЫЕ АЛГОРИТМЫ

2.1 РАСТРОВЫЕ ИЗОБРАЖЕНИЯ

Итак, растр – это матрица ячеек (пикселов). Каждый пиксел может иметь свой цвет. Совокупность пикселов различного цвета образует изображение. Для описания расположения пикселов используют разнообразные системы координат. Общим для всех таких систем является то, что координаты пикселов образуют дискретный ряд значений (необязательно целые числа). Часто для вывода на экран используется система целых координат – номеров пикселов с (0, 0) в левом верхнем углу.

Простые элементы, из которых складываются сложные объекты, будем называть *графическими примитивами*. Простейшим и, вместе с тем, наиболее универсальным растровым графическим примитивом является пиксел. Любое растровое изображение можно нарисовать по пикселам, но это и сложно, и долго. Необходимы более сложные элементы, для которых рисуются сразу несколько пикселов. Рассмотрим графические примитивы, которые используются наиболее часто в современных графических системах – это линии и окружности.

2.2 ГЕНЕРАЦИЯ ВЕКТОРОВ

Назначение генератора векторов (отрезков) - соединение двух точек изображения отрезком прямой.

Далее будут рассмотрены три алгоритма:

- два алгоритма ЦДА - цифрового дифференциального анализатора (DDA - Digital Differential Analyzer) для генерации векторов - обычный и несимметричный;
- алгоритм Брезенхема для генерации векторов.

Перед рассмотрением конкретных алгоритмов сформулируем общие требования к изображению отрезка:

- концы отрезка должны находиться в заданных точках;
- отрезки должны выглядеть прямыми,
- яркость вдоль отрезка должна быть постоянной и не зависеть от длины и наклона.

Ни одно из этих условий не может быть точно выполнено на растровом дисплее в силу того, что изображение строится из пикселов конечных размеров, а именно:

- концы отрезка в общем случае располагаются на пикселах, лишь наиболее близких к требуемым позициям и только в частных случаях координаты концов отрезка точно совпадают с координатами пикселов;
- отрезок формируется набором пикселов и лишь в частных случаях вертикальных, горизонтальных и отрезков под 45º они будут выглядеть прямыми, причем гладкими прямыми, без ступенек только для вертикальных и горизонтальных отрезков (рис. 2.1);
- яркость для различных отрезков и даже вдоль отрезка в общем случае различна, так как, например, расстояние между центрами пикселов для вертикального отрезка и отрезка под 45º различно (см. рис.2.1).

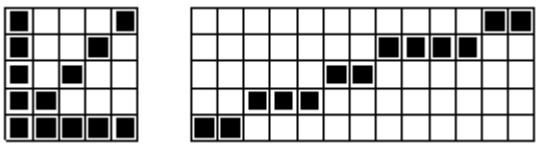

Рис. 2.1: Растровое представление различных векторов

Объективное улучшение вида отрезка достигается увеличением разрешения дисплея, но в силу существенных технологических проблем разрешение для растровых систем приемлемой скорости разрешение составляет порядка 1280×1024.

Субъективное улучшение основано на психофизиологических особенностях зрения и, в частности, может достигаться просто уменьшением размеров экрана. Другие способы субъективного улучшения качества основаны на различных программных ухищрениях по "размыванию" резких границ изображения.

Перед нами стоит следующая задача.

Даны 2 точки с координатами: (x_n, y_n) и (x_k, y_k). Необходимо их соединить отрезком прямой. Для вывода линии необходимо закрасить в определенный цвет все пикселы вдоль линии. Для того чтобы закрасить каждый пиксел, необходимо знать его координаты. Для определения координат закрашиваемых пикселов и используются алгоритмы генерации отрезка.

Замечание: дальнейшие алгоритмы описываются для стандартной декартовой системы координат

2.2.1 Цифровой дифференциальный анализатор (обычный)

С помощью ЦДА решается дифференциальное уравнение отрезка, имеющее вид:

$$\frac{dY}{dX} = \frac{Py}{Px}$$

где $Py = Y_k - Y_n$ приращение координат отрезка по оси Y,

а $Px = X_k - X_n$ - приращение координат отрезка по оси X.

При этом ЦДА формирует дискретный набор (аппроксимацию) непрерывного решения этого дифференциального уравнения.

В ***обычном ЦДА*** тем или иным образом определяется количество узлов N, используемых для построения (аппроксимации) отрезка. Затем за N циклов вычисляются координаты очередных точек отрезка:

$$X_0 = X_n;\ \ X_{i+1} = X_i + \frac{Px}{N}$$

$$Y_0 = Y_n;\ \ Y_{i+1} = Y_i + \frac{Py}{N}$$

Получаемые значения X_i, Y_i преобразуются в целочисленные значения координат очередного закрашиваемого пиксела либо округлением, либо отбрасыванием дробной части.

Генератор векторов, использующий этот алгоритм, имеет тот недостаток, что точки могут прорисовываться дважды, что увеличивает время построения.

Кроме того, из-за независимого вычисления обеих координат нет предпочтительных направлений, и построенные отрезки кажутся не очень красивыми.

2.2.2 Цифровой дифференциальный анализатор (несимметричный)

Субъективно лучше смотрятся отрезки с единичным шагом по большей относительной координате (несимметричный ЦДА). Для Px > Py (при Px, Py > 0) это означает, что координата по X-направлению должна увеличиться на 1 Px раз, а координата по Y-направлению должна также Px раз увеличиться, но на Py/Px.

Т.е. количество узлов построения отрезка (аппроксимации) берется равным числу пикселов вдоль наибольшего приращения.

Для генерации отрезка из точки (x_n, y_n) в точку (x_k, y_k) в первой четверти $(Px \geq Py \geq 0)$ алгоритм несимметричного ЦДА имеет вид:

1. Вычислить приращения координат:

$$Px = x_k - x_n;$$

$$Py = y_k - y_n;$$

2. Занести начальную точку отрезка и нарисовать

$$X1 = x_n;$$

$$Y1 = y_n;$$

PutPixel (X1, Y1);

3. Сгенерировать отрезок

while ($X1 < x_k$) {

X1= X1 + 1.0;

Y1= Y1 + Py/Px;

PutPixel (X1, Y1);

}

Пример генерации отрезка по алгоритму несимметричного ЦДА приведен на рис.2.2.

Рисунок 2.2 – Генерация отрезка несимметричным ЦДА

Основные *достоинства* данных алгоритмов:

1. Простота и ясность построения алгоритма.
2. Возможность работы с нецелыми значениями координат отрезка.

Недостатки:

Использование деления: в компьютере дробные числа представляются в формате с плавающей точкой не точно. Кроме погрешности представления дробных чисел существует ошибка выполнения арифметических операций с плавающей точкой. С каждым шагом ошибка накапливается, и может так произойти, что последнее значение Y1 не будет равняться заданному значению Yk.

2.2.3 Алгоритм Брезенхема генерации отрезка

Для устранения использования чисел с плавающей точкой, Брезенхем предложил алгоритм, не требующий деления, но обеспечивающий минимизацию отклонения сгенерированного образа от истинного отрезка.

Основная идея алгоритма состоит в том, что если угловой коэффициент прямой < 1/2, то естественно точку, следующую за точкой (0,0), поставить в позицию (1,0) (рис. 2.3, а), а если угловой коэффициент > 1/2, то - в позицию (1,1) (рис. 2.3, б). Для принятия решения, куда заносить очередной пиксел, вводится величина отклонения E точной позиции от середины между двумя возможными растровыми точками в направлении наименьшей относительной координаты. Знак E используется как критерий для выбора ближайшей растровой точки.

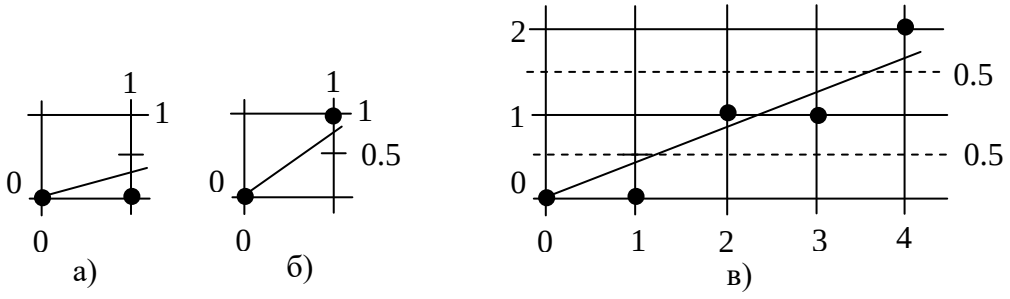

Рисунок 2.3 – Алгоритм Брезенхема генерации отрезков

Если E < 0, то точное Y-значение округляется до последнего меньшего целочисленного значения Y, т.е. Y-координата не меняется по сравнению с предыдущей точкой. В противном случае Y увеличивается на 1.

Например, рассмотрим генерацию отрезка, начинающегося в точке (0,0) и проходящего через точку (4, 1.7) (см. рис. 2.3, в), т.е. отрезок имеет положительный наклон меньший 1.

Из рис. 2.3, в видно, отклонение для первого шага:

$$E_1 = Py/Px - 1/2 < 0,$$

поэтому для закрашивания пиксела выбирается точка (1,0).

Отклонение для второго шага вычисляется добавлением приращения Y-координаты для следующей X-позиции (см. рис. 2.3, в):

$$E_2 = E_1 + Py/Px > 0,$$

поэтому для занесения пиксела выбирается точка (2,1). Так как отклонение считается от Y-координаты, которая теперь увеличилась на 1, то из накопленного отклонения для вычисления последующих отклонений надо вычесть 1:

$$E_2 = E_2 - 1.$$

Отклонение для третьего шага:

$$E_3 = E_2 + Py/Px < 0,$$

поэтому для занесения пиксела выбирается точка (3,1).

Суммируя и обозначая большими буквами растровые точки, а маленькими - точки вектора, получаем:

$$E_1 = y_n - 1/2 = Py/Px - 1/2.$$

Возможны случаи:

$E_1 > 0$	$E_1 \le 0$
ближайшая точка есть:	
$X_1 = X_0 + 1;\quad Y_1 = Y_0 + 1;$	$X_1 = X_0 + 1;\quad Y_1 = Y_0;$
$E_2 = E_1 + Py/Px - 1;$	$E_2 = E_1 + Py/Px.$

Так как интересует только знак E, то можно избавиться от неудобных частных умножением E на 2Px:

	$E_1 = 2Py - Px$
$E_1 > 0$:	$E_2 = E_1 + 2(Py - Px)$
$E_1 \le 0$:	$E_2 = E_1 + 2Py$

Таким образом, получается алгоритм, в котором используются только целые числа, сложение, вычитание и умножение:

$X= x_n;$

$Y= y_n;$

$Px= x_k - x_n;$

$Py= y_k - y_n;$

$E= 2Py - Px;$

$i= Px;$

PutPixel(X, Y); /* Первая точка вектора */

while ($i= i- 1 \ge 0$) {

if ($E \ge 0$) {

$X= X + 1;$

$Y= Y + 1;$

$E= E + 2(Py - Px);$

} else {

```
X= X + 1;
E= E + 2Py;}
PutPixel(X, Y); /* Очередная точка вектора */
}
```

Этот алгоритм пригоден для случая $0 \leq Py \leq Px$. Для других случаев алгоритм строится аналогичным образом.

На рис.2.4 приведен пример генерации по алгоритму Брезенхема того же самого отрезка, что и показанного на рис. 2.2 для генерации по алгоритму несимметричного ЦДА. Из сравнения рисунков видно, что результаты различны.

Рисунок 2.4 – Генерация отрезка по алгоритму Брезенхема

Разработаны алгоритмы генератора окружностей и других конических сечений.

2.3 ГЕНЕРАЦИЯ ОКРУЖНОСТИ

Во многих областях приложений, таких как, например, системы автоматизированного проектирования, естественными графическими примитивами, кроме отрезков прямых являются и конические сечения, т.е. окружности, эллипсы, параболы и гиперболы. Наиболее употребительным примитивом является окружность. Один из наиболее распространенных алгоритмов генерации окружности разработан Брезенхемом.

Алгоритм Брезенхема генерации окружности

Рассмотрим генерацию 1/8 окружности, центр которой лежит в начале координат. Остальные части окружности могут быть получены последовательными отражениями (использованием симметрии точек на окружности относительно центра и осей координат).

Окружность с центром в начале координат описывается уравнением:

$$X^2 + Y^2 = R^2$$

Алгоритм Брезенхема пошагово генерирует очередные точки окружности, выбирая на каждом шаге для закрашивания пиксель растра $P_i(X_i, Y_i)$, ближайший к истинной окружности, так чтобы ошибка:

$$E_i(P_i) = (X_i^2 + Y_i^2) - R^2$$

была минимальной.

Причем, как и в алгоритме Брезенхема для генерации отрезков, выбор ближайшей точки выполняется с помощью анализа значений управляющих переменных, для вычисления которых не требуется вещественной арифметики. Для выбора очередной точки достаточно проанализировать знаки.

Рассмотрим генерацию 1/8 окружности по часовой стрелке, начиная от точки X=0, Y=R.

Проанализируем возможные варианты выбора (i+1)-й точки, после выбора i-й.

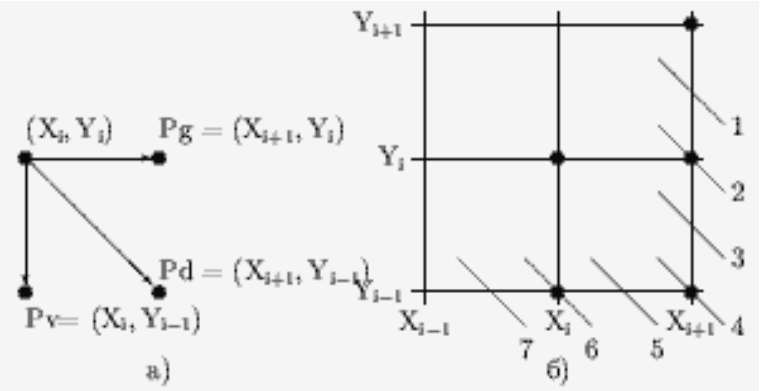

Рисунок 2.5 – Варианты расположения очередного пиксела окружности

При генерации окружности по часовой стрелке после выбора точки (X_i, Y_i) следующая точка может быть (см. рис. 2.5, а) либо Pg = (X_{i+1}, Y_i) - перемещение по горизонтали, либо Pd = (X_{i+1}, Y_{i-1}) - перемещение по диагонали, либо Pv = (X_i, Y_{i-1}) - перемещение по вертикали.

Из этих возможных точек выбирается наиболее близко расположенная. Для этого вычисляем и сравниваем абсолютные значения разностей квадратов расстояний от центра окружности до выбираемой точки и реальной точки окружности:

$$| Dg |=| (X+1)^2 + Y^2 - R^2 |$$
$$| Dd |=| (X+1)^2 + (Y-1)^2 - R^2 |$$
$$| Dv |=| X^2 + (Y-1)^2 - R^2$$

Выбирается и рисуется та точка, для которой это значение минимально.

Выбор точки определяется по значению Dd. Если Dd < 0, то диагональная точка внутри окружности. Это варианты 1-3 (см. рис. 2.5, б). Если Dd > 0, то диагональная точка вне окружности. Это варианты 5-7. И, наконец, если Dd = 0, то диагональная точка лежит точно на окружности. Это вариант 4. Рассмотрим случаи различных значений Dd в только что приведенной последовательности.

Случай Dd < 0

В качестве следующего пиксела могут быть выбраны или горизонтальный - Pg или диагональный - Pd.

Для определения того, какой пиксел выбрать Pg или Pd составим разность:

$$di = |Dg| - |Dd| = |(X+1)^2 + Y^2 - R^2| - |(X+1)^2 + (Y-1)^2 - R^2|$$

И будем выбирать точку Pg при di ≤ 0, в противном случае выберем Pd.

Рассмотрим вычисление di для разных вариантов.

Для вариантов 2 и 3 (рис 2.5, б):

Dg ≥ 0 и Dd < 0, т.к. горизонтальный пиксел либо вне, либо на окружности, а диагональный внутри.

$$di = (X+1)^2 + Y^2 - R^2 + (X+1)^2 + (Y-1)^2 - R^2;$$

Добавив и вычтя $(Y-1)^2$, получим:

$$di = 2 \cdot [(X+1)^2 + (Y-1)^2 - R^2] + 2 \cdot Y - 1$$

В квадратных скобках стоит Dd, так что

$$di = 2 \cdot (Dd + Y) - 1$$

Для варианта 1 (рис 2.5, б):

Ясно, что должен быть выбран горизонтальный пиксел Pg. Проверка компонента di показывает, что Dg < 0 и Dd < 0, причем di < 0, так как диагональная точка больше удалена от окружности, т.е. по критерию di < 0 как и в предыдущих случаях следует выбрать горизонтальный пиксел Pg,.

Случай Dd > 0

В качестве следующего пиксела могут быть выбраны или диагональный - Pd или вертикальный Pv.

Для определения того, какой пиксел выбрать Pd или Pv составим разность:

$$si = |Dd| - |Dv| = \left|(X+1)^2 + (Y-1)^2 - R^2\right| - \left|X^2 + (Y-1)^2 - R^2\right|$$

Если si ≤ 0, то расстояние до вертикальной точки больше и надо выбирать диагональный пиксел Pd, если же si > 0, то выбираем вертикальный пиксел Pv.

Рассмотрим вычисление si для разных вариантов.

Для вариантов 5 и 6 (рис 2.5, б):

Dd > 0 и Dv ≤ 0, так как диагональный пиксел вне, а вертикальный либо вне либо на окружности.

$$si = (X+1)^2 + (Y-1)^2 - R^2 + X^2 + (Y-1)^2 - R^2;$$

Добавив и вычтя $(X+1)^2$, получим:

$$si = 2 \cdot [(X+1)^2 + (Y-1)^2 - R^2] - 2 \cdot X - 1$$

В квадратных скобках стоит Dd, так что

$$si = 2 \cdot (Dd - X) - 1$$

Для варианта 7 (рис 2.5, б):

Ясно, что должен быть выбран вертикальный пиксел Pv. Проверка компонента si показывает, что Dd > 0 и Dv > 0, причем si > 0, так как диагональная точка больше удалена от окружности, т.е. по критерию si > 0 как и в предыдущих случаях следует выбрать вертикальный пиксел Pv, что соответствует выбору для вариантов 5 и 6.

Случай Dd = 0

Для компонента di имеем: Dg > 0 и Dd = 0, следовательно, по критерию di > 0 выбираем диагональный пиксел.

С другой стороны, для компонента si имеем: Dd = 0 и Dv < 0, так что по критерию si ≤ 0 также выбираем диагональный пиксел.

Итак:

Если Dd < 0, то проверяем di:

если di ≤ 0 - выбор горизонтального пиксела Pg

если di > 0 - выбор диагонального пиксела Pd

Если Dd > 0 , то проверяем si:

если si ≤ 0 - выбор диагонального пиксела Pd

если si > 0 - выбор вертикального пиксела Pv

Если Dd = 0 , то

выбор диагонального пиксела Pd.

Выведем рекуррентные соотношения для вычисления Dd для (i+1)-го шага, после выполнения i-го.

1. Для горизонтального шага к точке (X_{i+1}, Y_i):

$X_{i+1} = X_i + 1$

$Y_{i+1} = Y_i$

$Dd_{i+1} = (X_{i+1}+1)^2 + (Y_{i+1}-1)^2 - R^2 = X_{i+1}^2 + 2\cdot X_{i+1} + 1 + (Y_{i+1}-1)^2 - R^2 =$

$(X_i+1)^2 + (Y_i-1)^2 - R^2 + 2\cdot X_{i+1} + 1 = Dd_i + 2\cdot X_{i+1} + 1$

2. Для диагонального шага к точке (X_{i+1}, Y_{i-1})

$X_{i+1} = X_i + 1$

$Y_{i+1} = Y_i - 1$

$Dd_{i+1} = Dd_i + 2 \cdot X_{i+1} - 2 \cdot Y_{i+1} + 2$

3. Для вертикального шага к точке (X_i, Y_{i-1})

$X_{i+1} = X_i$

$Y_{i+1} = Y_i - 1$

$Dd_{i+1} = Dd_i - 2 \cdot Y_{i+1} + 1$

Для написания программы, реализующей описанный выше алгоритм и строящей дугу окружности в первой четверти необходимо задать следующие параметры начальной инициализации:

$X= 0$; $Y= R$; $Dd = (X+1)^2 + (Y-1)^2 - R^2 = 1 + (R-1)^2 - R^2 = 2*(1 - R)$

Пикселы в остальных четвертях можно получить отражением.

2.4 СТИЛЬ ЛИНИИ. ПЕРО

Для описания различных по виду изображений на основе линий используют термин – ***стиль линии*** или ***перо***.

2.4.1 АЛГОРИТМ ВЫВОДА ТОЛСТОЙ ЛИНИИ

Взяв за основу любой алгоритм вывода обычных тонких линий (например, алгоритм Брезенхема), запишем его в следующем виде:

.....

.....

Вывод пиксела (X, Y)

.....

.....

Данный алгоритм в общем виде можно представить как цикл, в котором определяются координаты (X, Y) каждого пиксела и данный пиксел выводится на экран.

Этот алгоритм можно модифицировать для вывода толстой линии следующим образом:

.....

.....

Вывод фигуры (или линии) пера с центром в точке (X, Y)

.....

.....

Т.е. вместо вывода отдельного пикселя стоит вывод фигуры или линии, соответствующей перу – прямоугольник, круг, отрезок прямой.

Такой подход к разработке алгоритмов вывода толстых линий имеет свои достоинства и недостатки.

Достоинство – можно прямо использовать эффективные алгоритмы для вычисления координат линии оси (например, алгоритмы Брезенхема).

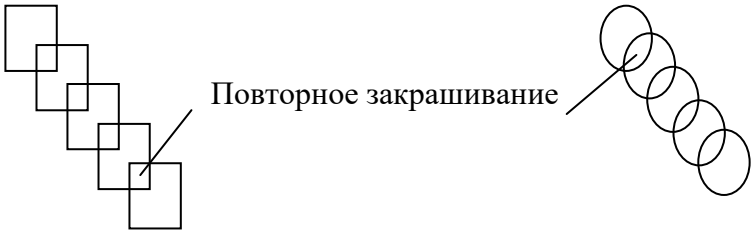

Рисунок 2.6 – Прямоугольное и круглое перья работают избыточно

Недостаток – неэффективность алгоритма для некоторых форм пера. Для перьев, которые соответствуют фигурам с заполнением, большинство пикселов многократно закрашивается в одних и тех же точках (рис.2.6).

Такие алгоритмы более эффективны для перьев в виде отрезков прямых. В этом случае каждый пиксель рисуется только один раз. Но в данном случае важным является наклон отрезка – ширина пера зависит от наклона отрезка (рис.2.7).

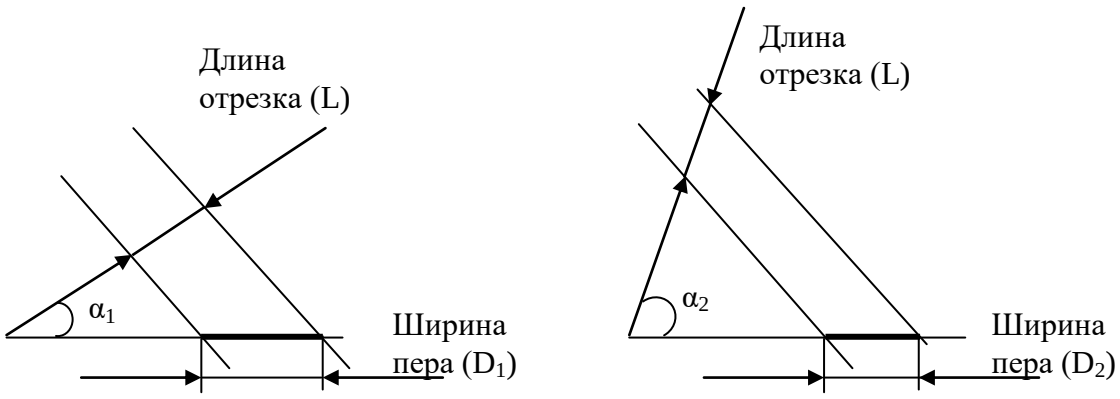

Рисунок 2.7 – Перья в виде отрезков

Очевидно, что вертикальное перо не может рисовать толстую вертикальную линию, а горизонтальное перо не может рисовать толстую горизонтальную линию.

Для вывода толстых линий с помощью пера в виде отрезка прямой чаще всего используются отрезки горизонтальной или вертикальной линии, реже – наклонные отрезки под углом 45°.

2.4.2 АЛГОРИТМ ВЫВОДА ПУНКТИРНОЙ ЛИНИИ

Алгоритм рисования тонкой пунктирной линии можно получить из алгоритма вывода тонкой непрерывной линии. Напомним, что алгоритм вывода непрерывной тонкой линии упрощенно можно представить следующим образом:

```
- - - -
Вывод пикселя (x, y)
- - - -
```

Если в данном алгоритме процедуру вывода пикселя заменить более сложной конструкцией, то в результате можно получить пунктирную линию:

```
- - - -
Проверка значения счетчика C:
   Если C удовлетворяет некоторым условиям, то
       вывод пикселя (x, y)
Значение C увеличивается на единицу
- - - -
```

В данном алгоритме используется новая переменная C – счетчик пикселов линии. Если C удовлетворяет некоторому логическому условию, то рисуется пиксель заданного цвета с текущими координатами (x, y). Логическое условие будет определять стиль линии. Например, если условием будет проверка четности C, то получим линию из отдельных точек. Для рисования пунктирной линии можно проанализировать остаток от деления C на S (S - некоторая переменная). Например, если

рисовать пиксели линии только тогда, когда C mod S < S/2, то получим пунктирную линию с длиной штриха S/2 и с шагом S. Задавая разные логические (условные) выражения, можно получить разные пунктирные линии.

2.4.3 АЛГОРИТМ ВЫВОДА ТОЛСТОЙ ПУНКТИРНОЙ ЛИНИИ

Объединив алгоритм вывода толстой непрерывной линии и алгоритм вывода тонкой пунктирной линии, можно получить следующий алгоритм:

```
- - - -
Проверка значения счетчика C:
   Если C удовлетворяет некоторым условиям, то
     вывод фигуры (линии) пера с центром в (x, y)
     C = C + 1
- - - -
```

Такой алгоритм достаточно прост. На практике используются и более сложные и изощренные алгоритмы.