ЛЕКЦИЯ 3 *Алгоритмы построения геометрических фигур. Отрезки прямой линии, окружности, эллипсы, фигуры Лиссажу, кривые Безье.*

ГЛАВА 2. АЛГОРИТМЫ ПОСТРОЕНИЯ И ПРЕОБРАЗОВАНИЯ ИЗОБРАЖЕНИЙ

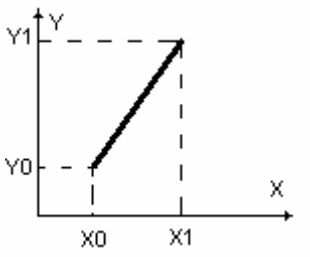
2.1. Растровые алгоритмы построения геометрических фигур

2.1.1. Отрезки прямой линии

Существует два способа аналитического задания прямой линии:

- в виде функции $y=a\cdot x+b$;
- в виде координат пары точек $((x_0, y_0), (x_1, y_1))$.

Связь этих двух способов: $a=(y_1-y_0)/(x_1-x_0)$; $b=y_0-a\cdot x_0$.



Отрезок прямой

«Естественный» алгоритм рисования прямой использует представление прямой в виде функции $y=a\cdot x+b$:

```
For x:=x0 to x1 do Точка(x, x*a+b);
```

Этот алгоритм нужно уточнить на ряд случаев:

1) если $|x_1-x_0|<|y_1-y_0|$, то изменять надо координату y, а не x.

2) если $x_1<x_0$ (или $y_1<y_0$), то надо уменьшать координату x (или y), т.е. следует организовать цикл:

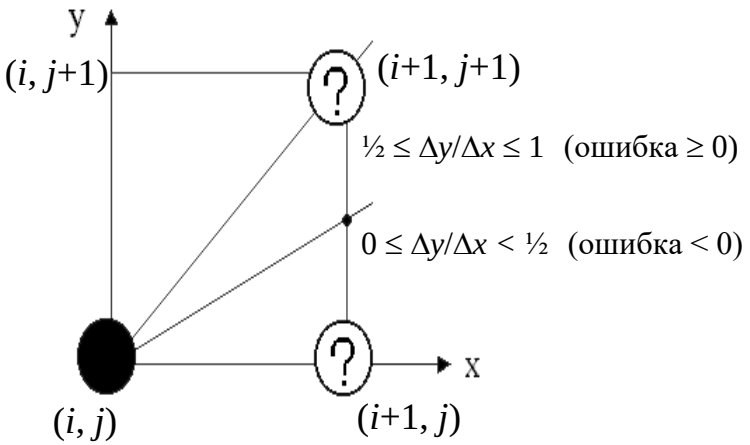
```
For x:=x1 to x0 Точка(x, x*a+b);
```

Инкрементный алгоритм Брезенхама (*Brezenham*) выбирает оптимальные растровые координаты для представления отрезка. Основная цель инкрементных алгоритмов – это построение циклов вычисления координат на основе только «быстрых» целочисленных операций (сложения, вычитания, сравнения) без использования «медленных» умножения и деления.

В процессе работы одна из координат - либо x, либо y (в зависимости от углового коэффициента) - изменяется на единицу. Изменение другой координаты (на 0 или 1) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние назовем ошибкой - e.

Алгоритм построен так, что требуется проверять лишь знак этой ошибки.

На рисунке приведена иллюстрация работы алгоритма Брезенхама для отрезка в первом октанте, т.е. когда угловой коэффициент принимает значения от 0 до 1.



Основная идея алгоритма Брезенхама рисования прямой

Анализируя рисунок, можно заметить, что если угловой коэффициент отрезка ($m=\Delta y/\Delta x$) находится в диапазоне [1/2; 1], то пересечение отрезка с прямой $x=i+1$ будет находиться ближе к узлу ($i+1$, $j+1$), чем к узлу ($i+1$, j). Следовательно, точка растра ($i+1$, $j+1$) лучше аппроксимирует ход отрезка, чем точка ($i+1$, j). Если угловой коэффициент меньше 1/2, то верно обратное, т.е. необходимо выбрать точку растра ($i+1$, j). Для углового коэффициента, равного 1/2, нет какого-либо предпочтительного выбора, это решение принимает программист. Примем решение выбирать "верхнюю" точку в данном "спорном" случае.

Так как желательно проверять только знак ошибки, то она первоначально устанавливается равной (m-1/2).

На каждой итерации величина ошибки поправляется с учетом того, увеличилась координата y на предыдущем шаге или нет.

Если ошибка e отрицательна, то отрезок пройдет ниже середины пиксела. Поэтому y не увеличивается. Величина ошибки в следующей точке растра в этом случае вычисляется как

$e = e + m$.

Если e положительна, т.е. отрезок пройдет выше середины пиксела. Координата y увеличивается на 1. Ошибка корректируется следующим образом:

$e=e + m - 1$.

На рисунке приведена схема алгоритма Брезенхема для первого октанта. Предполагается, что концы отрезка (x_0, y_0) и (x_1, y_1) не совпадают.

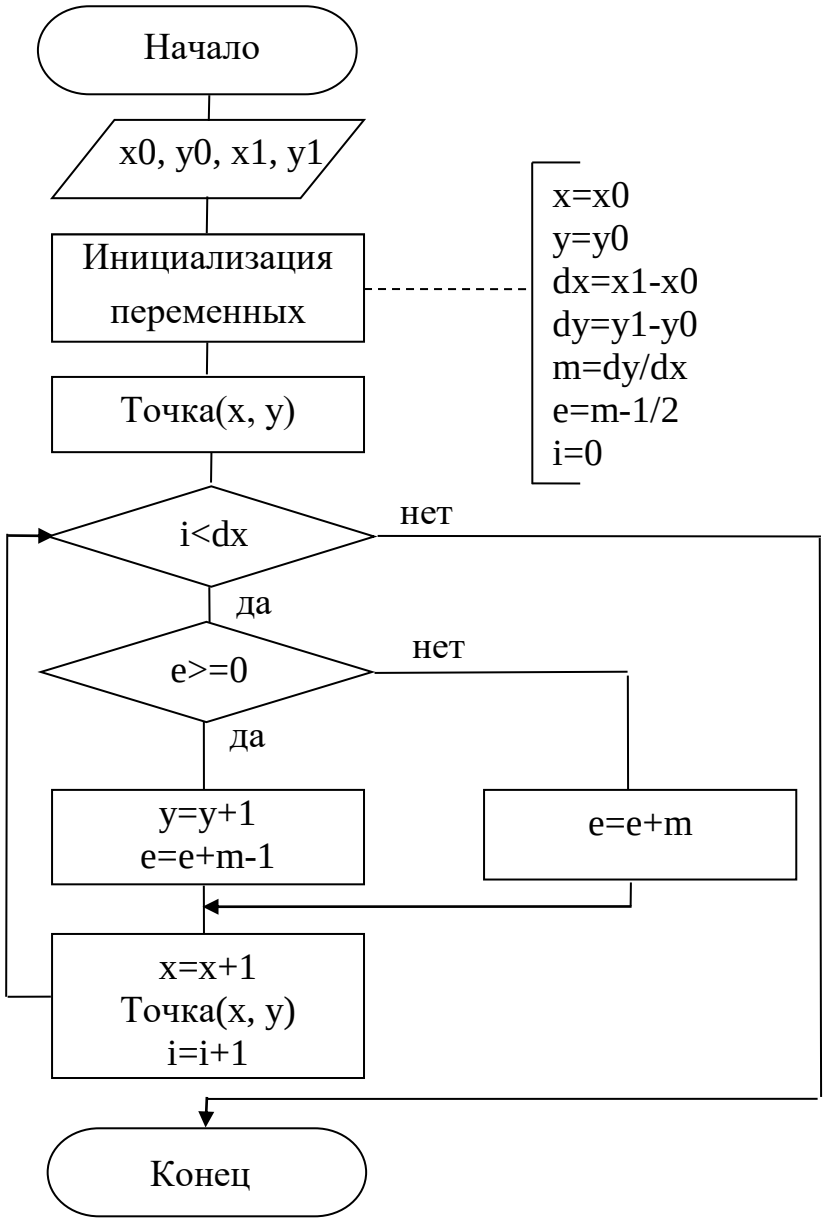
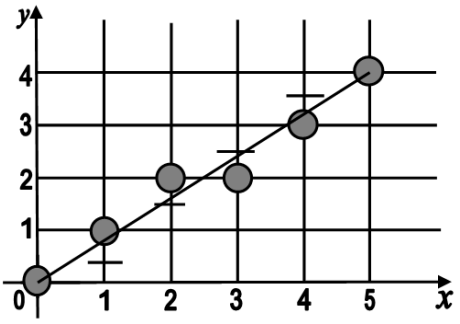


Схема алгоритма Брезенхама рисования прямой

В качестве иллюстрации работы данного алгоритма построим отрезок из точки (0, 0) в точку (5, 4). В данном случае *dx*=5, *dy*=4. Трассировка работы алгоритма и результат работы приведены на рисунках ниже.



Результат работы алгоритма Брезенхама

i	**Точка**	***e***	***x***	***y***
0	(0, 0)	3/10	0	0
1	(1, 1)	1/10	1	1
2	(2, 2)	-1/10	2	2
3	(3, 2)	7/10	3	2
4	(4, 3)	5/10	4	3
5	(5, 4)	3/10	5	4

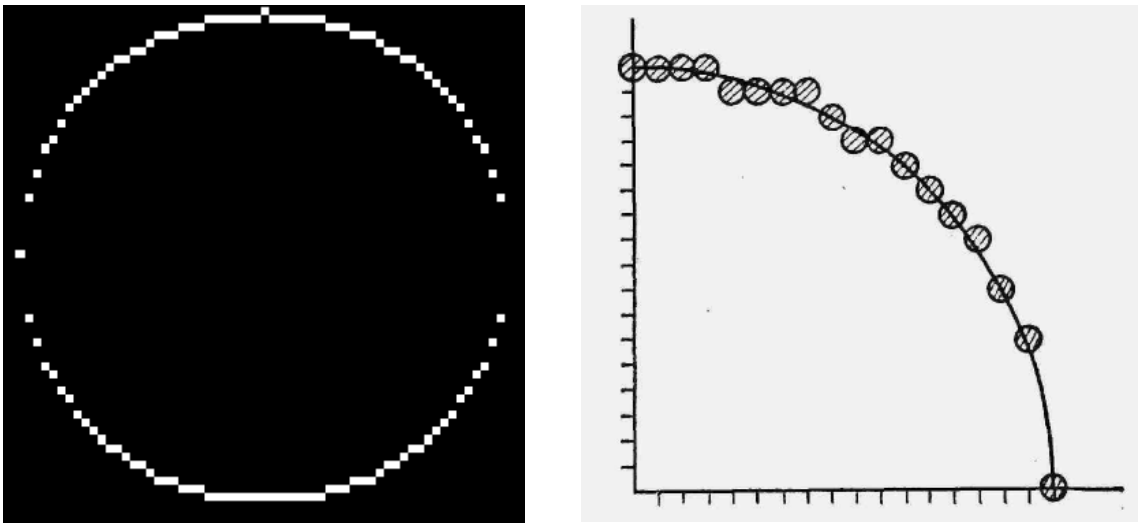
Трассировка работы алгоритма Брезенхама

Данный алгоритм нужно уточнить и дополнить для случаев, когда угловой коэффициент $\Delta y/\Delta x$ больше 1 и когда $x_1<x_0$ или $y_1<y_0$.

2.1.2. Окружность и эллипс

«Естественный» алгоритм рисования использует представление окружности в виде формулы $Y = \pm\sqrt{R^2 - (X - X_0)^2} + Y_0$:

```
For X:=X0-R to X<X0+R do begin
     Точка(X, Y0+sqrt(R*R-sqr(X-X0)));
     Точка(X, Y0-sqrt(R*R- sqr(X-X0)));
End;
```

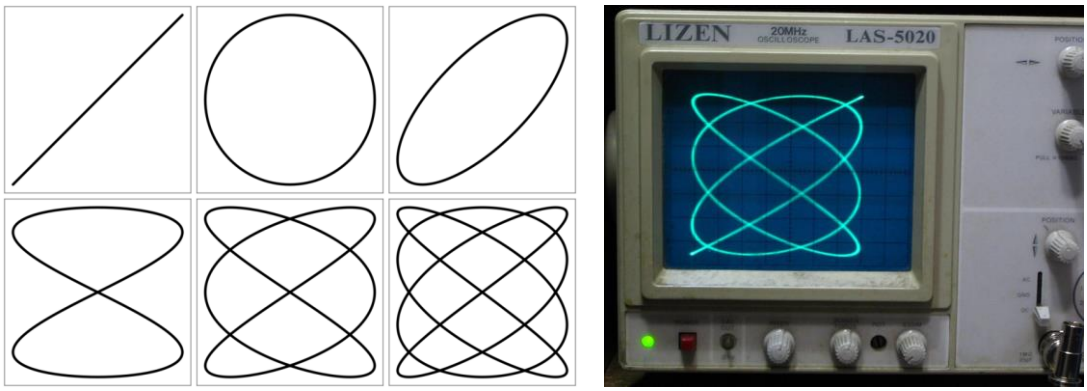


Недостатки прорисовки контура круга у «естественного» алгоритма

Параметрический способ рисования использует представление кривой в виде пары функций, зависящих от параметра *t*:

$x=x_0+R_1\cdot\cos(\omega_1\cdot t)$;
$y=y_0+R_2\cdot\sin(\omega_2\cdot t)$;

Здесь $t\in[0..2\pi]$. Если $R_1=R_2$ и $\omega_1=\omega_2$, то получается окружность; если $R_1\neq R_2$, то получается эллипс; если $\omega_1\neq\omega_2$, то получаются «фигуры Лиссажу».



Параметрически заданные кривые (окружность, эллипс, фигуры Лиссажу)

Ниже приведен пример программы на Паскале, реализующей построение окружности по приведенным формулам. В примере используются процедуры *moveto* и *lineto*, первая из которых устанавливает графический курсор на место с указанными

координатами, а вторая - проводит линию от текущего положения курсора до точки, координаты которой являются ее параметрами.

```
procedure my_circle(x,y,r:integer);
 var alfa: integer;
 begin
  moveto(x+r,y);
  for alfa:=1 to 360 do
  { угол измеряется в градусах }
  lineto(round(x+r*cos(alfa*pi/360)),
         round(y+r*sin(alfa*pi/360)))
  { здесь используется радианная мера углов }
 end;
```

Если вы располагаете временем и достаточно любопытны, то можете попробовать определить время, которое потребуется для изображения, к примеру, тысячи произвольных окружностей с помощью нашей процедуры, и сравнить его с временем, которое понадобится на то же самое с использованием стандартной процедуры *circle*. Вряд ли полученные результаты вас обрадуют. Кроме всего прочего, необходимо сделать следующее замечание: на самом деле строится не окружность, а правильный 360-угольник.

Алгоритм Брезенхама построения окружности не требует выполнять в циклах вычисления для чисел с плавающей точкой**.**

В алгоритме Брезенхама используются две идеи, позволяющие ускорить вывод на экран окружности:

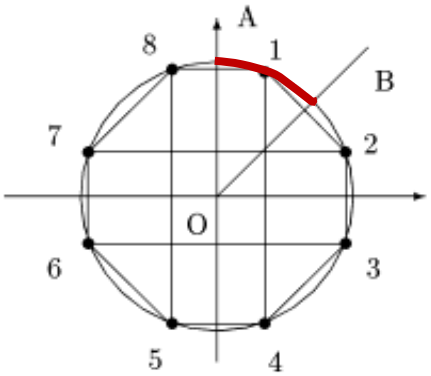
1) координаты точек окружности рассчитываются только для 1/8 части всей окружности. Для остальных 7/8 расчеты не производятся. Точки выводятся на экран с использованием симметрии окружности;

2) расчеты координат точек в цикле производятся для целых чисел.

Перейдем к более подробному описанию этих двух идей.

Во-первых – нет необходимости вычислять координаты точек всей окружности, достаточно вычислить 1/8 ее часть и последовательным применением преобразований симметрии получить из нее полную окружность. Мы станем строить 1/8 часть окружности, заключенную в сегменте AOB.

Каждая точка этого фрагмента должна быть еще семь раз отражена с помощью преобразований симметрии для получения полной окружности.



Симметричное отражение точки окружности

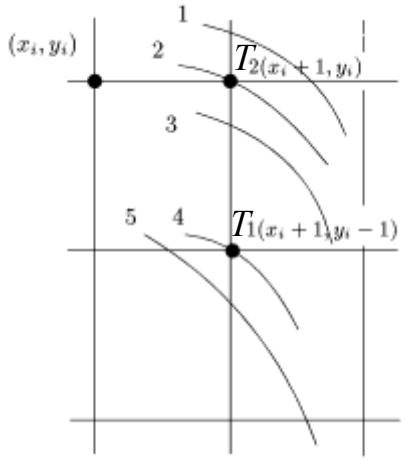
Процедура `Draw8Pixels` выводит 8 симметричных точек окружности, кроме того, именно эта процедура отвечает за расположение центра окружности. Главная процедура вполне может считать, что она строит окружность с центром в начале координат.

```
Procedure Draw8Pixels;
begin
  Точка(x+x0, y+y0, color);
  Точка(x+x0, -y+y0, color);
  Точка(-x+x0, y+y0, color);
  Точка(-x+x0, -y+y0, color);
  Точка(y+x0, x+y0, color);
  Точка(y+x0, -x+y0, color);
  Точка(-y+x0, x+y0, color);
  Точка(-y+x0, -x+y0, color);
end;
```

Время, затрачиваемое на прорисовку 8 симметричных точек окружности можно вычислить по следующей формуле:

$t_{\text{построение 8 точек окружности}}=t_{\text{расчет координат одной точки}}+8\cdot t_{\text{вывод на экран одной точки}}$.

Приступим к разбору второй – ключевой идеи алгоритма. Пусть мы находимся в некоторой промежуточной фазе построения. Мы только что поставили точку (x_i, y_i) и теперь должны сделать выбор между точками T_1 (x_i+1, y_i-1) и T_2 (x_i+1, y_i). (Так как мы строим часть окружности, заключенную в сектор AOB, следовательно, подняться выше мы не можем и спуститься вниз более чем на одну точку не можем тоже.)



Возможное расположение окружности относительно пикселов экрана

Реальная окружность может быть расположена относительно точек T_1 и T_2 одним из пяти способов 1-5.

Рассчитаем квадраты расстояний от центра окружности до точек T_1 и T_2:

$R_1^2 = (x_i+1)^2+(y_i-1)^2$,

$R_2^2 = (x_i+1)^2+(y_i)^2$.

Рассмотрим две погрешности Δ^i_{T1} и Δ^i_{T2}:

$\Delta^i_{T1} = R_1^2 - R^2 = (x_i+1)^2+(y_i-1)^2 - R^2$,

$\Delta^i_{T2} = R_2^2 - R^2 = (x_i+1)^2+(y_i)^2 - R^2$

и контрольную величину $\Delta^i = \Delta^i_{T1}+\Delta^i_{T2}$.

При выборе точки, следующей за (x_i, y_i), станем руководствоваться следующим критерием:

если $\Delta^i > 0$, выберем точку T_1;

если $\Delta^i \le 0$, выберем точку T_2.

Обоснуем разумность такого выбора. Рассмотрим знаки погрешностей Δ^i_{T1} и Δ^i_{T2} и их влияние на знак контрольной величины Δ^i для всех пяти возможных положений окружности.

Для положения 1.

$\Delta^i_{T1} < 0$, $\Delta^i_{T2} < 0 \Rightarrow \Delta^i = \Delta^i_{T1}+\Delta^i_{T2} < 0 \Rightarrow$ выбирается T_2.

Для положения 2.

$\Delta^i_{T1} < 0$, $\Delta^i_{T2} = 0 \Rightarrow \Delta^i < 0 \Rightarrow$ выбирается T_2.

Для положения 3 возможны варианты (учитывая, что $\Delta^i_{T1} < 0$, $\Delta^i_{T2} > 0$).

Вариант 3.1: $|\Delta^i_{T1}| \ge |\Delta^i_{T2}| \Rightarrow \Delta^i < 0 \Rightarrow$ выбирается T_2.

Вариант 3.2: $|\Delta^i_{T1}| < |\Delta^i_{T2}| \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Для положения 4.

$\Delta^i_{T1} = 0$, $\Delta^i_{T2} > 0 \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Для положения 5.

$\Delta^i_{T1} > 0$, $\Delta^i_{T2} > 0 \Rightarrow \Delta^i > 0 \Rightarrow$ выбирается T_1.

Получим выражение для контрольной величины Δ^i:

$\Delta^i = \Delta^i_{T1}+\Delta^i_{T2} = (x_i+1)^2+(y_i-1)^2 - R^2+(x_i+1)^2+(y_i)^2 - R^2 = 2x_i^2+2y_i^2+4x_i-2y_i+3-2R^2$.

Выражение для Δ^{i+1} существенным образом зависит от выбора точки на предыдущем шаге. Необходимо рассмотреть два случая: $y_{i+1} = y_i$ и $y_{i+1} = y_i-1$.

Δ^{i+1} [при $y_{i+1} = y_i$] $= 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 =$

$= 2(x_i+1)^2+2y_i^2+4(x_i+1)-2y_i+3-2R^2 = \Delta^i+4x_i+6$.

Δ^{i+1} [при $y_{i+1} = y_i-1$] $= 2x^2_{i+1}+2y^2_{i+1}+4x_{i+1}-2y_{i+1}+3-2R^2 =$

$= 2(x_i+1)^2+2(y_i-1)^2+4(x_i+1)-2(y_i-1)+3-2R^2 = \Delta^i +4(x_i-y_i)+10$.

Теперь, когда получено рекуррентное выражение для Δ^{i+1} через Δ^i, остается получить Δ^1 (контрольную величину в начальной точке.) Величина Δ^1 не может быть получена рекуррентно, ибо не определено предшествующее значение, зато легко может быть найдена непосредственно

$x_1 = 0$, $y_1 = R \Rightarrow$

$\Delta^1_{T1} = (0+1)^2+(R-1)^2-R^2 = 2-2R$,

$\Delta^1_{T2} = (0+1)^2+R^2-R^2 = 1$,

$\Delta^1 = \Delta^1_{T1} +\Delta^1_{T2} = 3-2R$.

Таким образом, алгоритм построения окружности основан на последовательном выборе точек. В зависимости от знака контрольной величины Δ^i выбирается следующая точка и нужным образом изменяется сама контрольная величина. Процесс начинается в точке $(0, R)$, а первая точка, которую ставит процедура `Draw8Pixels`, имеет координаты $(x_0, y_0+ R)$. При $x = y$ процесс заканчивается.

Схема алгоритма приведена ниже.

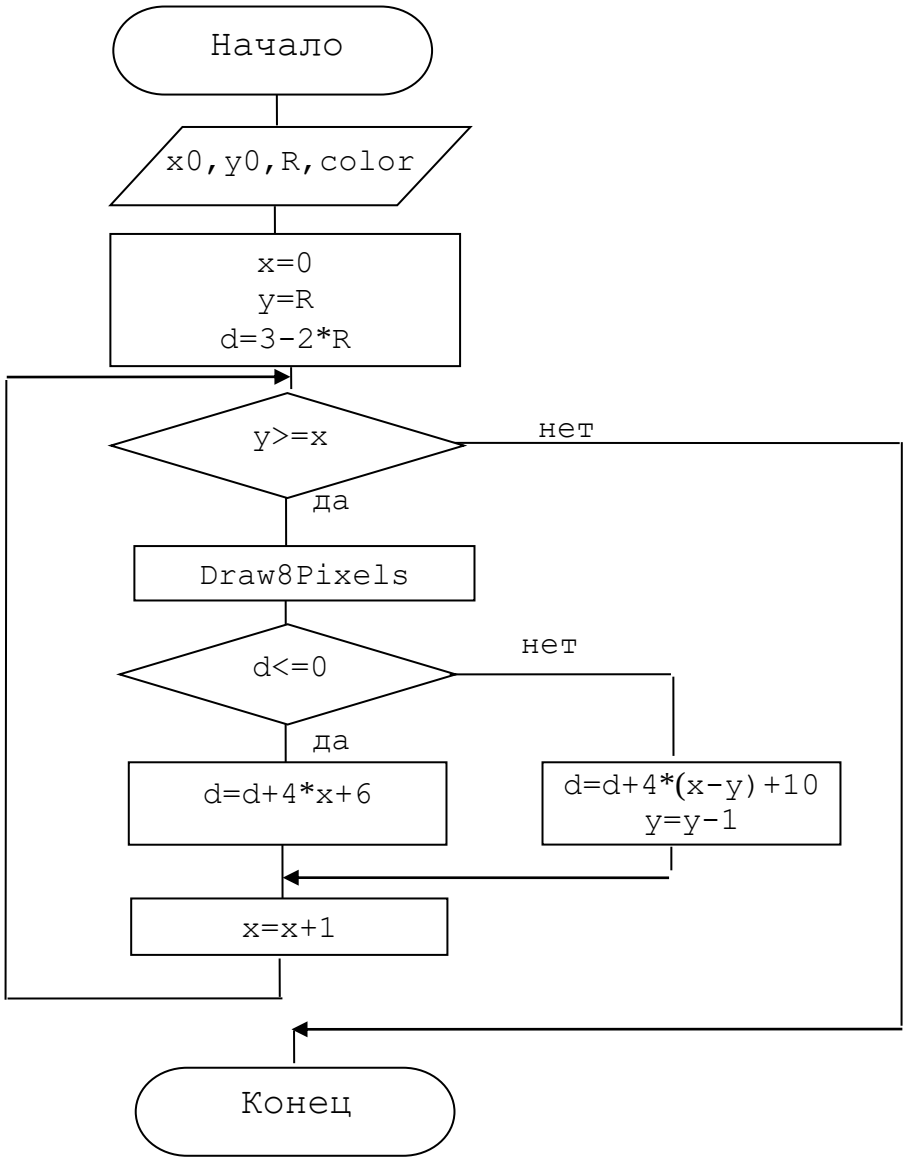


Схема алгоритма Брезенхама рисования окружности

Также существует разновидность алгоритма Брезенхама, позволяющая изображать эллипсы без использования вещественной арифметики.

2.1.3. Кривые и поверхности Безье

Разработаны математиком Пьером Безье (*Bezier*). Кривые и поверхности Безье были использованы в 60-х годах компанией «Рено» для компьютерного проектирования формы кузовов автомобилей. В настоящее время они широко используются в компьютерной графике.

Кривые Безье описываются *в параметрической форме* следующими формулами:

$$x = P_x(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} x_i ,$$

$$y = P_y(t) = \sum_{i=0}^{m} C_m^i t^i (1-t)^{m-i} y_i ,$$

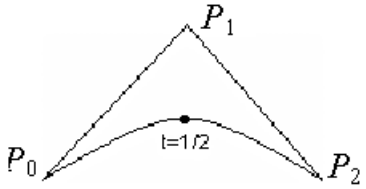
где $C_m^i = \frac{m!}{i!(m-i)!}$ - сочетание m по i, а x_i и y_i – координаты точек-ориентиров. Значение m на единицу меньше количества точек-ориентиров. Рассмотрим примеры кривых

Безье при разных значениях *m*.

1. *m*=1. (2 точки) Кривая вырождается в отрезок прямой линии $P(t) = (1-t)\cdot P_0+t\cdot P_1$

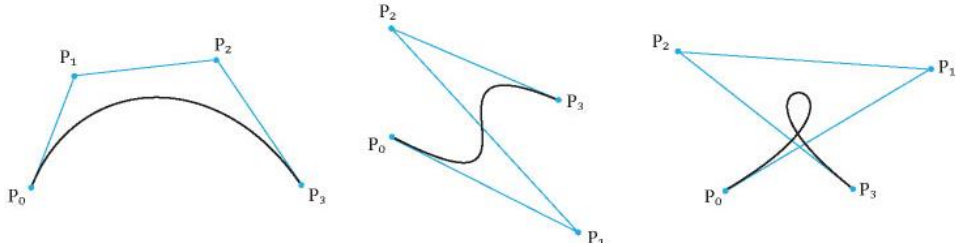
Кривая Безье 1-го порядка

2. *m*=2. (3 точки)
   $P(t) = (1-t)^2\cdot P_0 + 2t\,(1-t)\cdot P_1 + t^2\cdot P_2$

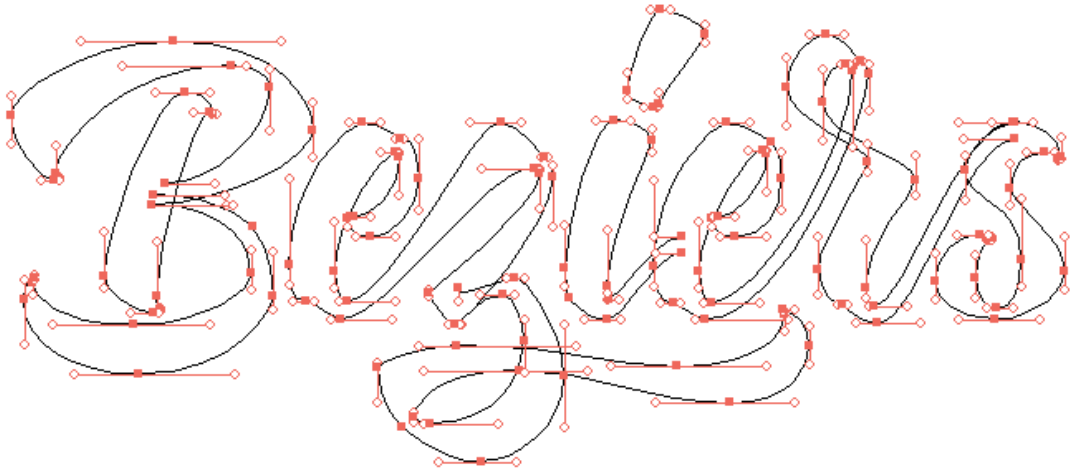


Кривая Безье 2-го порядка

3. *m* = 3. (4 точки)
   $P(t) = (1-t)^3\cdot P_0 + 3t(1-t)^2\cdot P_1 + 3t^2(1-t)\cdot P_2 + t^3\cdot P_3$



Кривые Безье 3-го порядка



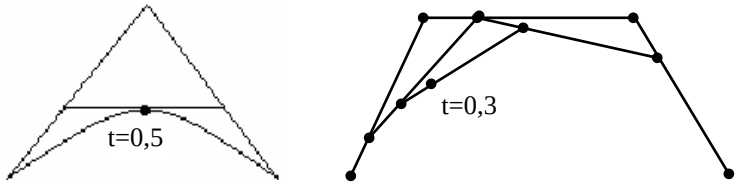
Шрифт, созданный с помощью кривых Безье 3-го порядка

Геометрический алгоритм построения кривой Безье позволяет вычислять координаты *x* и *y* по значению параметра *t*:

1) каждая сторона многоугольника делится пропорционально *t*;

2) точки деления соединяются отрезками прямых, образуя новый многоугольник с числом сторон на единицу меньшим;

3) стороны нового контура снова делятся пропорционально *t* и соединяются до тех пор, пока не будет получена единственная точка деления. Это и будет точка, принадлежащая кривой Безье.

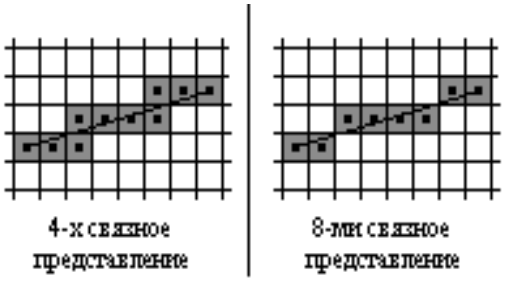


Геометрический способ построения кривых Безье

ЛЕКЦИЯ 4 *Алгоритмы закрашивания замкнутых фигур: рекурсивный алгоритм с «затравкой», алгоритм «короеда», модифицированный алгоритм с «затравкой», заполнение полигонов.*

2.2. Растровые алгоритмы закрашивания фигур

Будем считать, что фигура ограничена *контуром*, т.е. непрерывной линией, цвет которой отличен от цвета внутренней части фигуры.



К понятию связности контура

Связность. Будем называть два пиксела связными, если они являются "смежными".

4-смежность. Два пиксела являются 4-смежными, если они расположены рядом по горизонтали или вертикали.

8-смежность. Два пиксела являются 8-смежными, если они расположены рядом по горизонтали, вертикали или диагонали.

2.2.1. Рекурсивный алгоритм с «затравкой»

Шаг 1. Ставится точка нужного цвета («затравка») где-то внутри фигуры, она считается текущей.

Шаг 2. Проверяются поочередно все «соседи» для текущей точки, и если их цвет отличается от цвета закраски и цвета контура, то каждая такая точка закрашивается, и сама становится текущей (рекурсивный переход на шаг 1).

Соседние точки определяются в зависимости от того, какое условие связности определенно в условие задачи.

Этот алгоритм рекурсивный и приводит к большим затратам стековой памяти, поэтому его целесообразно использовать только для закраски небольших фигур (не более 1 тыс. пикселов).

Помимо встроенного механизма рекурсии в данном алгоритме можно также использовать структуру данных стек.

Стек содержит упорядоченный набор элементов и поддерживает две основные операции: добавить элемент и извлечь элемент. Вторая операция возвращает элемент, добавленный последним, и удаляет его из набора элементов.

Программно стек может быть реализован на основе одномерного массива или списка.

Схема алгоритма такой программы (назовем ее алгоритмом «короеда») будет выглядеть следующим образом:

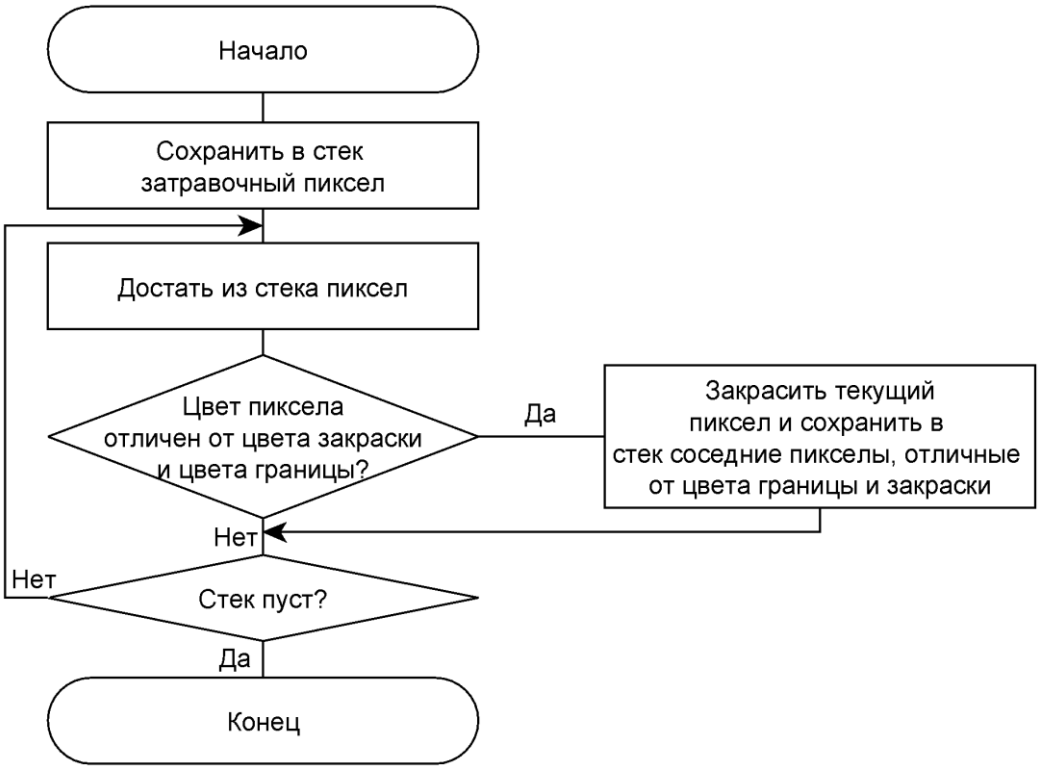
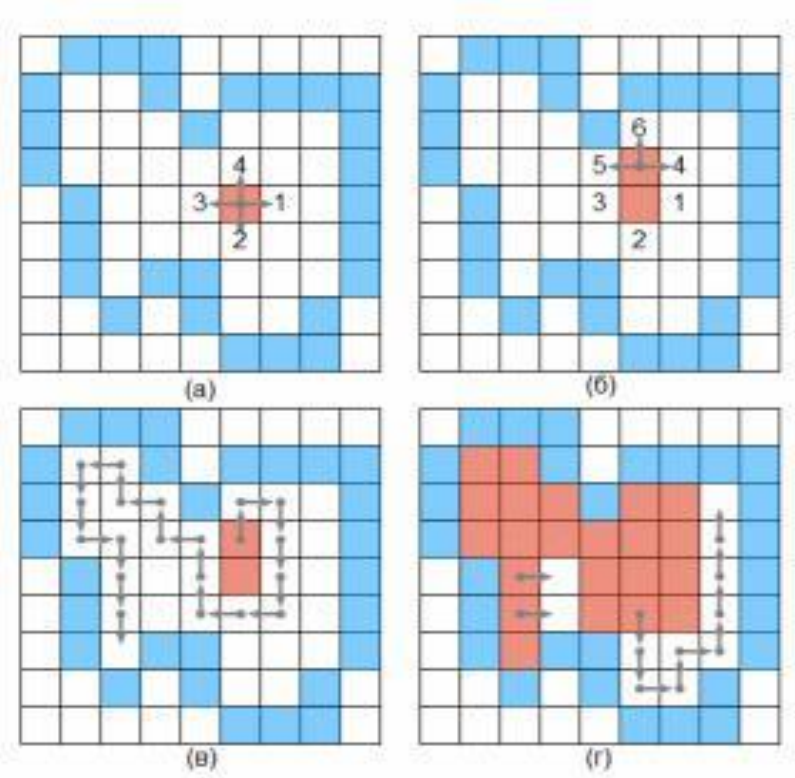


Схема алгоритма закраски «короедом»



Порядок обхода пикселей в алгоритме «короеда»

Модифицированный алгоритм с «затравкой»

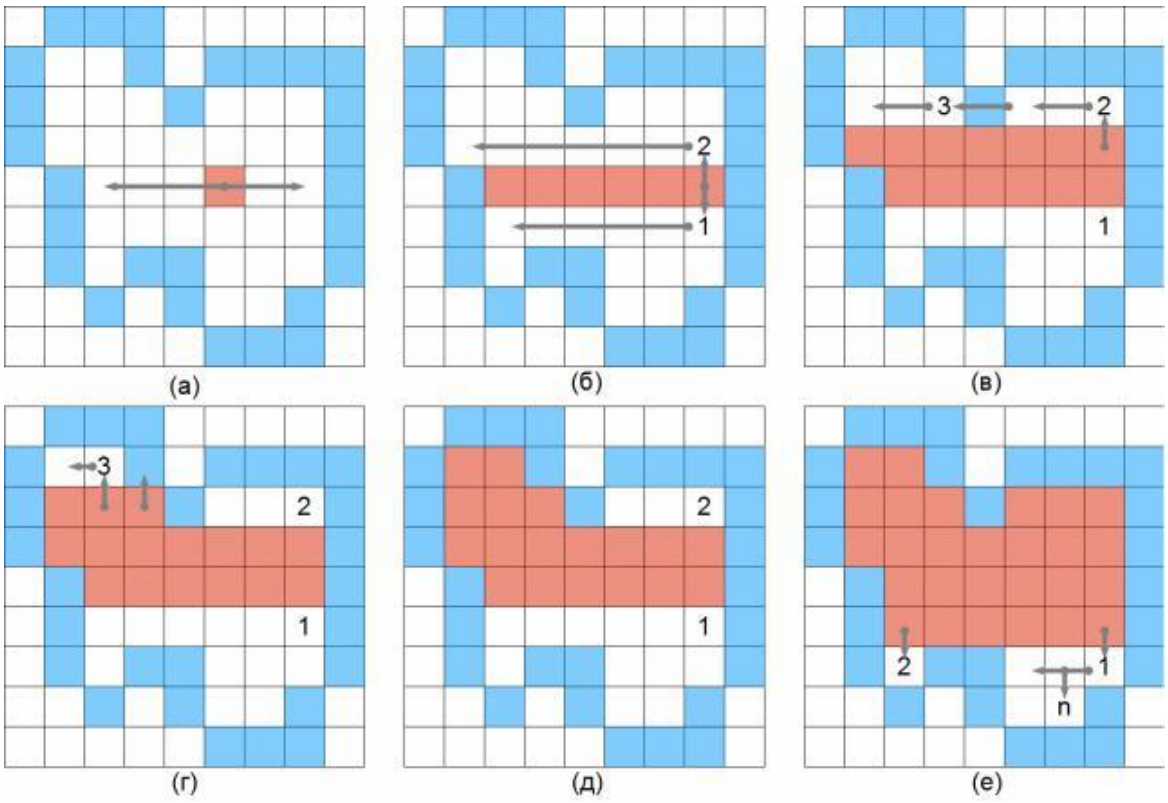
Заметим, что на каждой строке множество точек, подлежащих закраске, состоит из интервалов, принадлежащих внутренности области. Эти интервалы отделены друг от друга интервалами из точек, принадлежащих границе или внешности области. Кроме того, если набор точек образует связный интервал, принадлежащий внутренней части области, то точки над и под этим интервалом либо являются граничными, либо принадлежат внутренней части области. Последние могут служить затравочными для строк, лежащих выше и ниже рассматриваемой строки. Суммируя все это, можно предложить следующий алгоритм.

Шаг 1. Координата затравки помещается в стек, затем до исчерпания стека выполняются пункты 2-4.

Шаг 2. Координата очередной затравки извлекается из стека и выполняется максимально возможное закрашивание вправо и влево по строке с затравкой, т.е. пока не попадется граничный пиксель. Пусть это XLeft и XRight, соответственно.

Шаг 3. Анализируется строка ниже закрашиваемой в пределах от XLeft до XRight и в ней находятся крайние правые пиксели всех незакрашенных фрагментов. Их координаты заносятся в стек.

Шаг 4. То же самое проделывается для строки выше закрашиваемой.



Порядок обхода пикселей модифицирванного алгоритма с «затравкой»

2.2.2. Заполнение полигонов

Контур полигона (многоугольника) определяется вершинами, которые соединены отрезками прямых. Основная идея алгоритма – закрашивание фигур отрезками прямых линий.

Алгоритм заполнения полигонов отрезками прямых следующий:

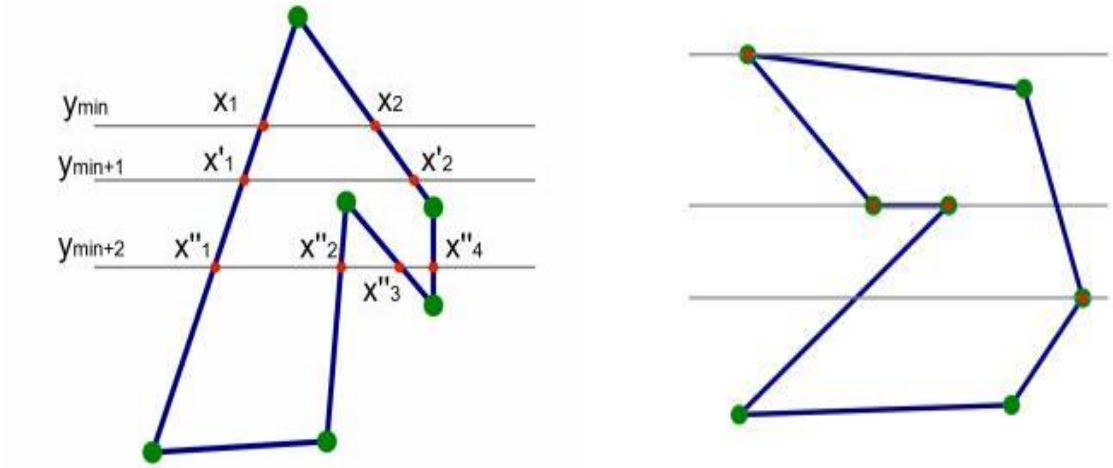
1)Найти *miny* и *maxy* среди всех вершин P_i.

2) Выполнить цикл по y от min до max.
{
3) Нахождение точек пересечения всех отрезков контура с горизонталью y. Координаты x_j точек пересечения записать в массив.
4) Сортировка массива $\{x_j\}$ по возрастанию.
5) Вывод горизонтальных отрезков с координатами:
$(x_0, y) - (x_1, y)$
$(x_2, y) - (x_3, y)$
...
$(x_{2k}, y) - (x_{2k+1}, y)$
}

В данном алгоритме используется свойство топологии контура фигуры: любая прямая линия пересекает замкнутый контур четное количество раз.

Координата пересечения отрезков p_ip_k с горизонталью y вычисляется по формуле:

$$x=x_i+(y_k-y)(x_k-x_i)/(y_k-y_i).$$



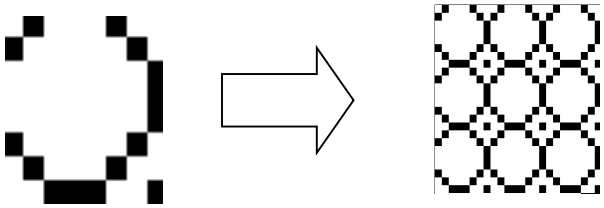
Закраска полигонов

2.2.3. Заполнение областей узорами

Пусть нужный узор хранится в массиве `byte pattern[Width][Height]`.

Пиксельная карта в массиве `pattern[][]` может быть целым изображением, которым мы хотим раскрасить заданную область, или это может быть маленький **элемент мозаики**, который следует выкладывать с повторением.

Для обеспечения повторяемости узора пикселу с координатами (x, y) нужно присваивать цвет `pattern[x mod Width][y mod Height]`.



Заполнение области узором