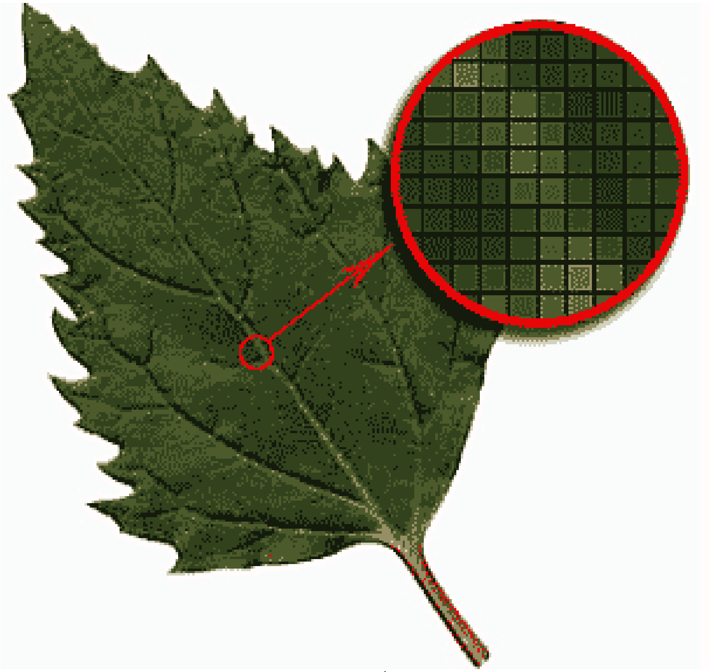
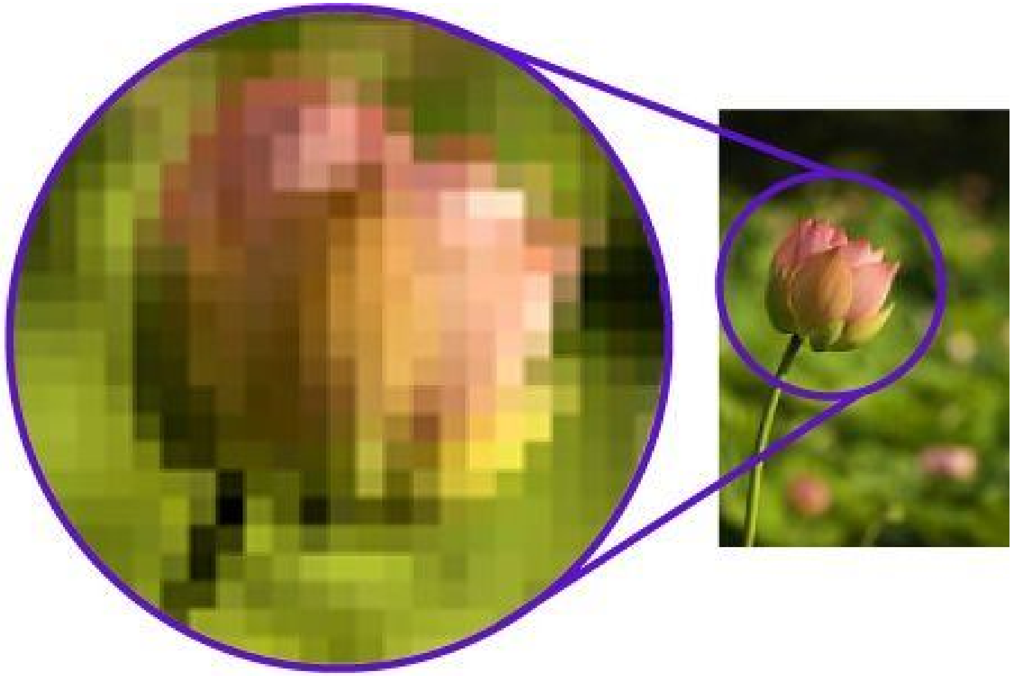
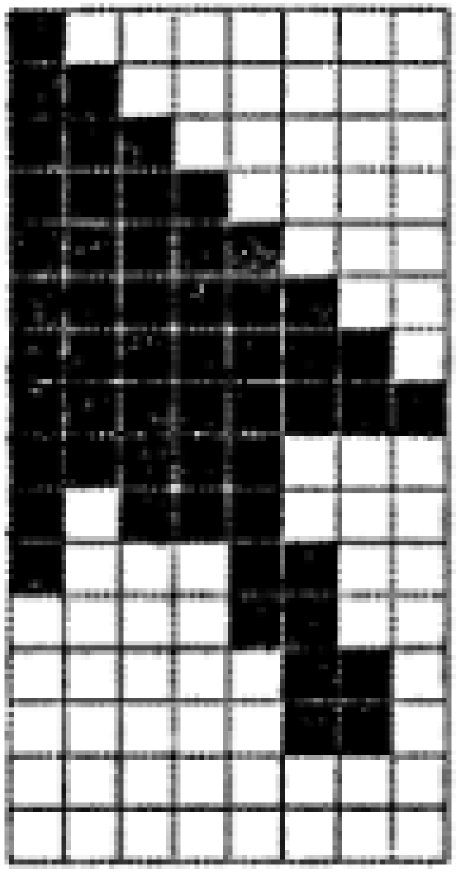
Растровое изображение





Растровое изображение хранится в компьютере в виде массива числовых величин. Массив является прямоугольным, с определенных числом строк и столбцов. Каждая числовая величина представляет значение пикселя, записанного в этом месте. Этот массив называют «пиксельной картой» («pixel map»).

Двухуровневое изображение и его битовая карта



1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0
1	1	1	1	1	1	0	0
1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	0	0	0
1	0	1	1	1	0	0	0
1	0	0	0	1	1	0	0
0	0	0	0	1	1	0	0
0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Если в растровом изображении имеются пикселы только с двумя значениями, то такое изображение называется двухуровневым или черно-белым.

Полутоновые изображения

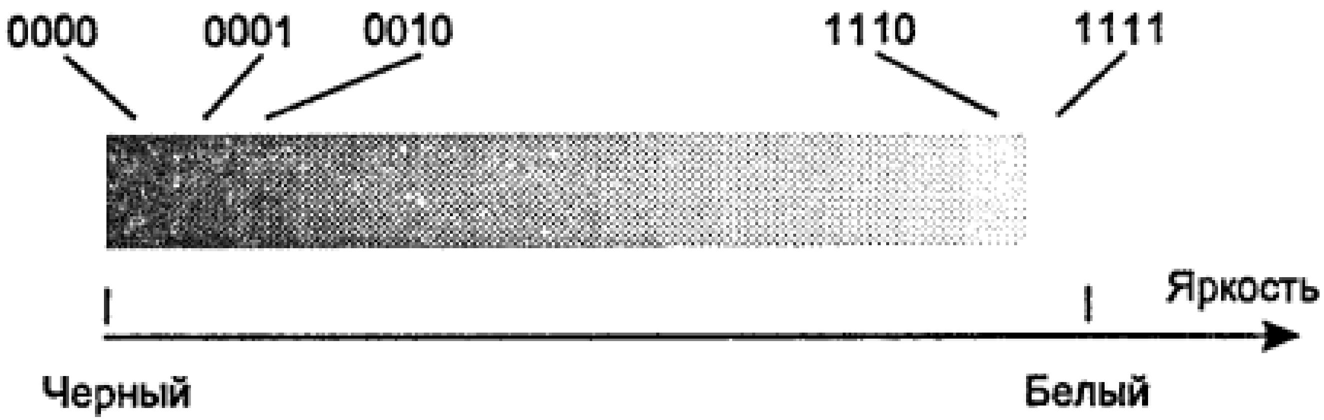
Полутоновые изображения классифицируются по «глубине пикселя» («pixel depth»), которая равна числу бит, необходимых для представления уровней их полутонов (оттенков серого).

Поскольку n-битовая величина имеет 2^n возможных значений, то в изображении с глубиной пикселей, равной n, может быть 2^n оттенков серого.

Полутоновые изображения

Чаще всего используются такие величины:

- два бита на пиксель обеспечивают 4 оттенка серого;
- четыре бита на пиксель обеспечивают 16 оттенков серого;
- восемь бит на пиксель обеспечивают 256 оттенков серого



Шестнадцать уровней яркости серого цвета

Цветные изображения

Каждый пиксель цветного изображения имеет свой «код цвета» («color value») — числовое значение, которое каким-либо образом представляет цвет.

Чаще всего цвет описывается комбинацией величин красного, зеленого и синего цветов. Значение каждого пикселя представляет собой упорядоченную тройку, например (23,14,51), которая описывает интенсивности красной, зеленой и синей составляющих — в указанном порядке.

Число бит, используемых для представления цвета каждого пикселя, зазывают глубиной цвета.

Цветные изображения

Каждая величина в тройке (красный, зеленый и синий) занимает определенное число бит, глубина цвета является суммой этих трех величин.

Глубина цвета, равная трем, использует по одному биту на каждый компонент.

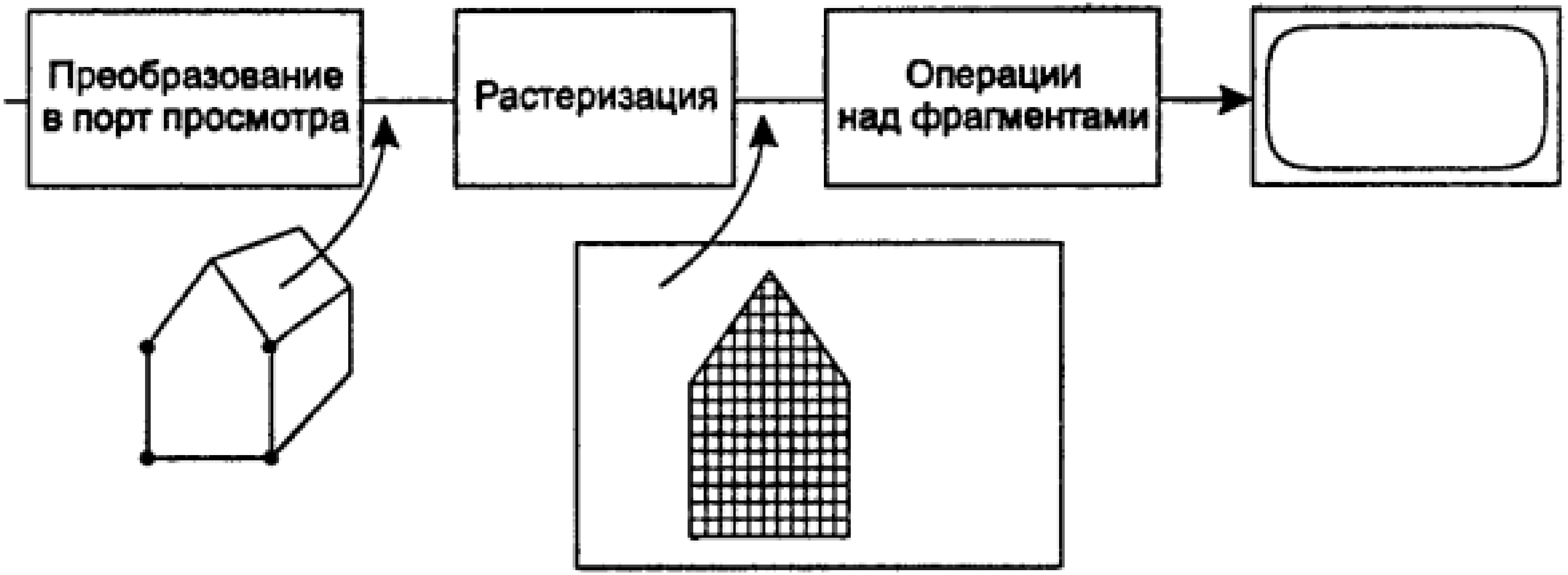
Код цвета	Изображение
0, 0, 0	Черный
0, 0, 1	Синий
0, 1, 0	Зеленый
0, 1, 1	Голубой
1, 0, 0	Красный
1, 0, 1	Пурпурный
1, 1, 0	Желтый
1, 1, 1	Белый

Цветные изображения

Многие изображения имеют глубину цвета равную восьми, по три бита под красную и зеленую составляющие и два под синюю.

Если используют по одному байту на каждую составляющую, то получаются изображения, обеспечивающие реалистичное цветовоспроизведение, их называют полноцветными изображениями (true color).

Растеризация в OpenGL



Назначение нужного цвета каждому отдельному пикселю «внутри» прямой или полигона называется «преобразованием развертки» или растеризацией.

Растровые алгоритмы

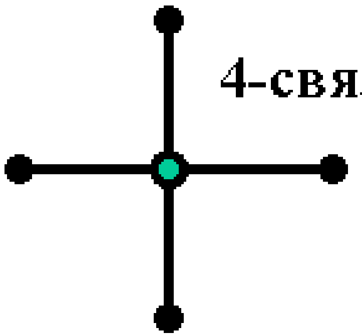
- Алгоритмы растеризации
  - Алгоритмы перевода графических примитивов в растровую форму
  - Алгоритмы заполнения областей и многоугольников
- Алгоритмы обработки растровых изображений
  - Регулировка яркости и контрастности
  - Масштабирование изображений
  - Геометрические преобразования
  - Алгоритмы фильтрации

Связность

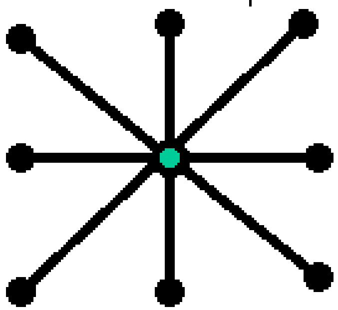
<u>Связность</u> – возможность соединения двух пикселей растровой линией, т.е. последовательным набором пикселей.

Пикселы Р1 и Р2 называются 4-СВЯЗНЫМИ, если у них отличаются только x-координаты или только y-координаты, причем только на 1: $|x1 - x2| + |y1 - y2| <= 1$

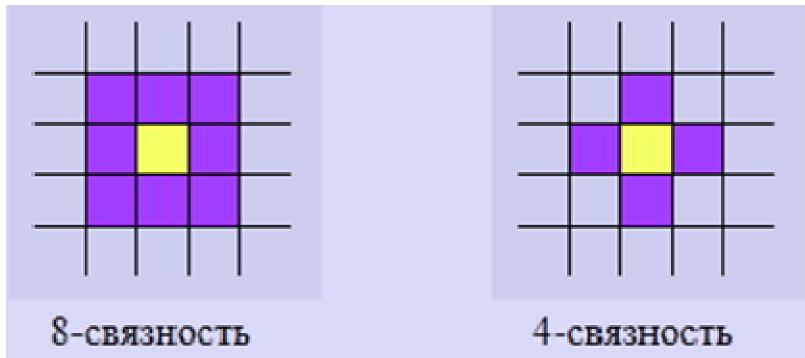
Пикселы Р1 и Р2 называются 8-СВЯЗНЫМИ, если у них отличаются x-координаты и/или y-координаты, но не более, чем на 1: $|X1 - X2| <= 1; \ |Y1 - Y2| <= 1$



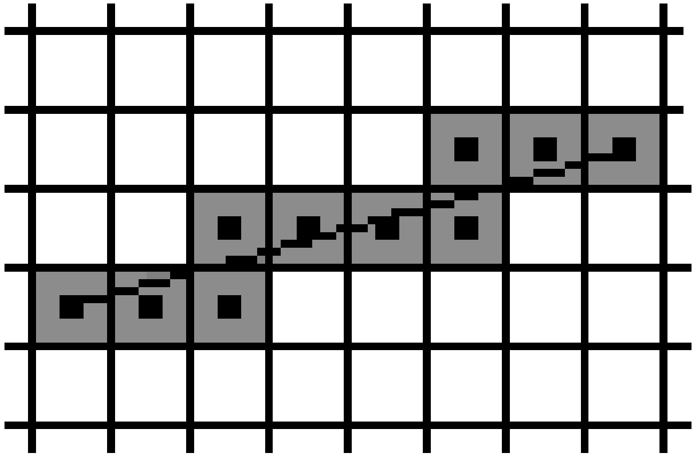
4-связность



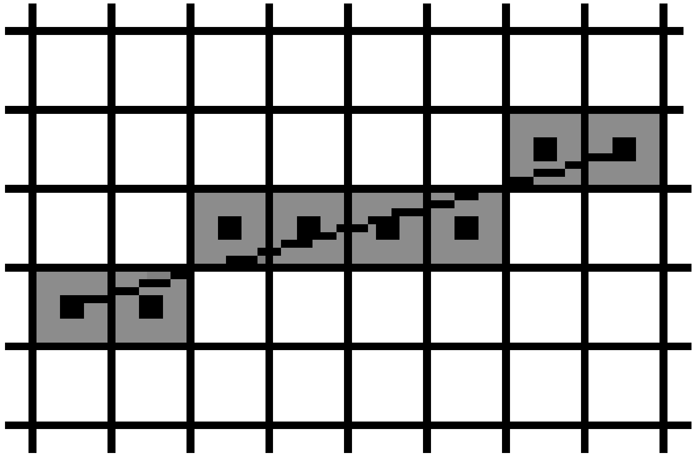
8-связность



Четырехсвязная и восьмисвязная линии



4-х связное
представление



8-ми связное
представление

Рисование отрезка прямой линии

```
void drawDot(GLint x, Glint y)
{
 glBegin(GL_POINTS); glVertex2i(x,y);
 glEnd();
 glFlush();
}
…
for (int x=x1; x<=x2;x++)//отрезок горизонтальной линии
drawDot(x,y1);
…
for (int y=y1; y<=y2;y++)//отрезок вертикальной линии
drawDot(x1,y);
```

Рисование отрезка прямой линии

Пусть конечные точки отрезка имеют целочисленные координаты, и уравнение прямой, содержащей отрезок: $y = kx + b$

Будем также считать, что тангенс угла наклона прямой лежит в пределах от 0 до 1. Тогда для изображения отрезка на растре достаточно для всех целых *x*, принадлежащих отрезку, выводить на экран точки с координатами $(x, \mathrm{Round}(y))$

```
void line(int x1, int y1, int x2, int y2)
{
double k=((double)(y2-y1))/(x2-x1);
double b=y1-k*x1;
for (int x=x1;x<=x2;x++)
drawDot(x,(int)(k*x+b));
}
```

Рисование отрезка прямой линии

Поскольку $k = \Delta y / \Delta x$, то один шаг по целочисленной сетке на оси x будет соответствовать $\Delta x = 1$. Отсюда получаем, что y будет увеличиваться на величину k ($y_i = kx_i + b$; $y_{i+1} = kx_{i+1} + b$; $y_{i+1} - y_i = kx_{i+1} + b - kx_i - b = k(x_{i+1} - x_i) = k$).

Итерационная последовательность выглядит следующим образом: $x_{i+1} = x_i + 1$, $y_{i+1} = y_i + k$.

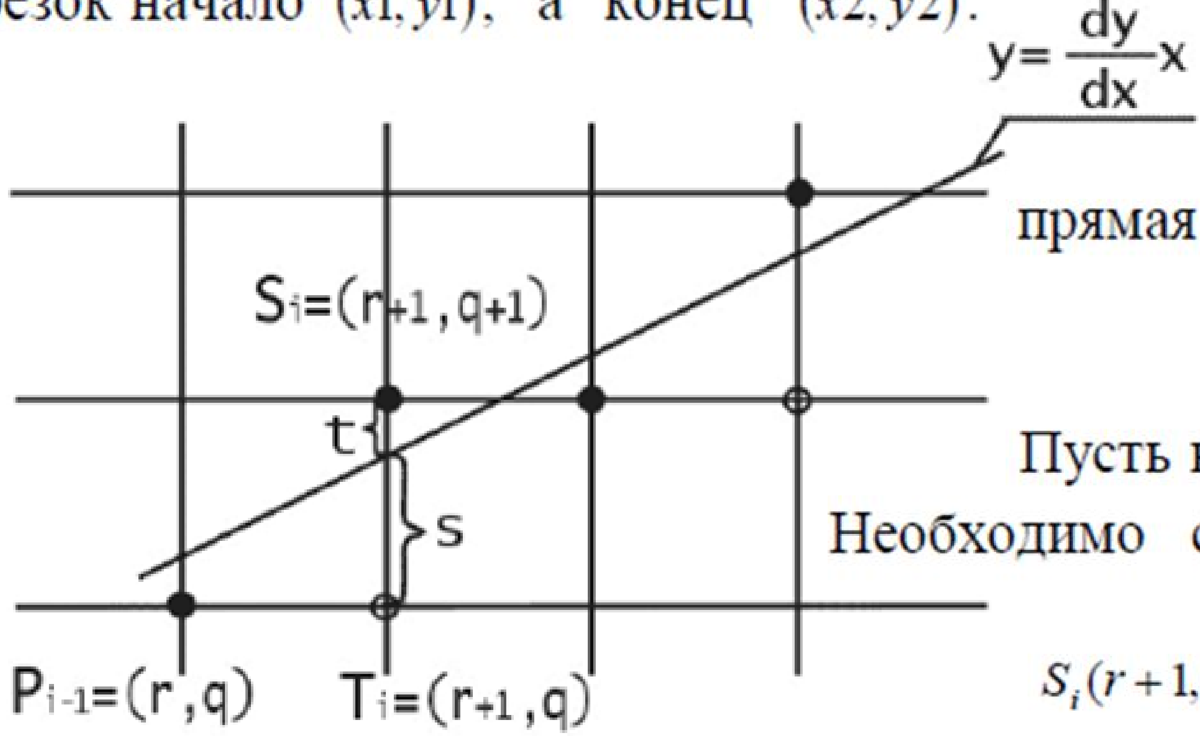
```
void line(int x1, int y1, int x2, int y2, int color)
{
double k = ((double)(y2 – y1)) / (x2 – x1);
double y = y1;
for (int x = x1; x <= x2; x++, y += k)
drawDot(x, (int)y);
}
```

Алгоритм Брезенхема рисования отрезка прямой линии

начало отрезка совпадает с началом координат,

отрезок начало $(x1, y1)$, а конец $(x2, y2)$.

$dx = (x2 - x1)$, $dy = (y2 - y1)$.



прямая имеет вид $y = \frac{dy}{dx}x$, где $\frac{dy}{dx} \in [0,1]$,

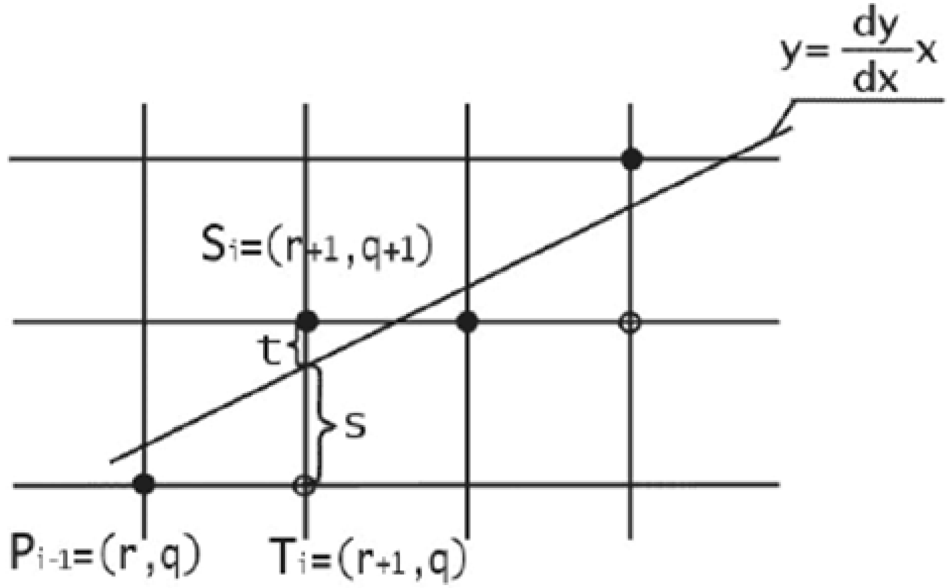
т.е. $dy < dx$.

Пусть на $(i-1)$-м шаге $P_{i-1} = (r, q)$.

Необходимо сделать выбор следующей точки

$S_i(r+1, q+1)$ или $T_i(r+1, q)$.

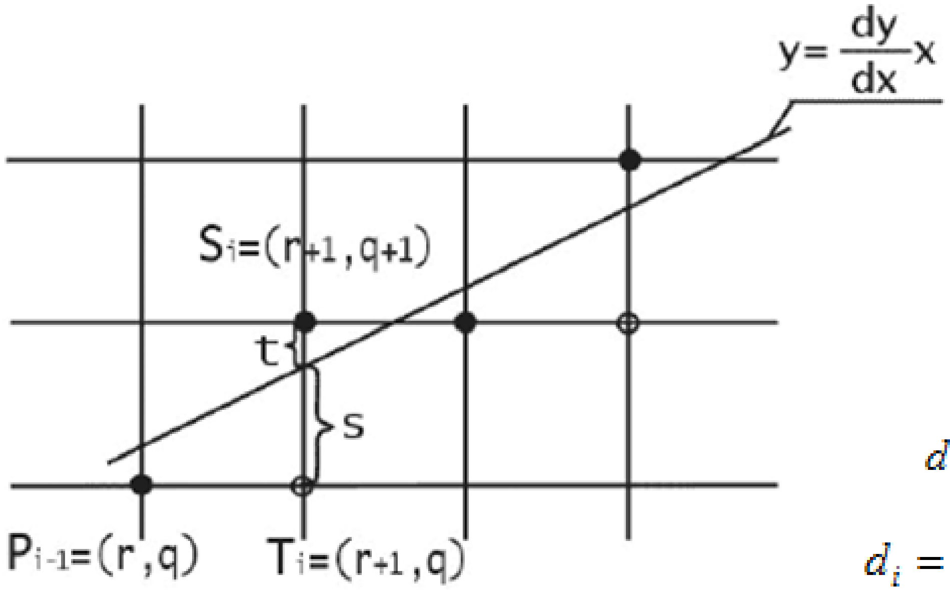
Алгоритм Брезенхема рисования отрезка прямой линии



Если $(s-t)<0$, то $P_i = T_i = (r+1, q)$ и тогда $x_i = x_{i-1}+1$, $y_i = y_{i-1}$,

если же $(s-t)\geq 0$, то $P_i = S_i = (r+1, q+1)$ и тогда $x_i = x_{i-1}+1$, $y_i = y_{i-1}+1$

Алгоритм Брезенхема рисования отрезка прямой линии



$$s=\frac{dy}{dx}(r+1)-q\,,\quad t=q+1-\frac{dy}{dx}(r+1),\Rightarrow$$

$$s-t=2\frac{dy}{dx}(r+1)-2q-1\ \Rightarrow$$

$$dx(s-t)=2(r\cdot dy-q\cdot dx)+2dy-dx\,.$$

$$d_i=dx(s-t)=2(r\cdot dy-q\cdot dx)+2dy-dx\,.\qquad r=x_{i-1}\qquad q=y_{i-1},$$

$$d_i=2(x_{i-1}\cdot dy-y_{i-1}\cdot dx)+2dy-dx=2x_{i-1}dy-2y_{i-1}dx+2dy-dx$$

$$d_{i+1}=2x_idy-2y_idx+2dy-dx$$

$$d_{i+1}-d_i=2x_idy-2y_idx+2dy-dx-2x_{i-1}dy+2y_{i-1}dx-2dy+dx=2x_idy-2x_{i-1}dy+2y_{i-1}dx-2y_idx$$

$d_{i+1}-d_i=2dy(x_i-x_{i-1})-2dx(y_{i-1}-y_i)$ зная, что $x_i=x_{i-1}+1$ (т.е. $x_i-x_{i-1}=1$)

$$d_{i+1}=d_i+2dy-2dx(y_i-y_{i-1}).$$

Алгоритм Брезенхема рисования отрезка прямой линии

Пусть на предыдущем шаге $d_i < 0$, тогда $(y_i - y_{i-1}) = 0$ и $d_{i+1} = d_i + 2dy$ и $y_{i+1} = y_i$. Если же на предыдущем шаге $d_i \geq 0$, то $(y_i - y_{i-1}) = 1$ и $d_{i+1} = d_i + 2(dy - dx)$ и $y_{i+1} = y_i + 1$.

Осталось узнать как вычислить d_1. Так как при $i = 1$:

$(x_0, y_0) = (0,0)$, $\Rightarrow d_1 = 2dy - dx$. (при подстановке в первое определение $d_i = dx(s - t) = 2(r \cdot dy - q \cdot dx) + 2dy - dx$)

Алгоритм Брезенхема рисования отрезка прямой линии

1. Рассчитаем $dx = (x_2 - x_1)$ и $dy = (y_2 - y_1)$.

2. $i = 1$, $d_i = 2dy$-dx, y_i=y_1, x_i=x_1.

3.Рисуем точку с координатами (x_i, y_i).

4. Если $d_i < 0$, тогда $d_{i+1} = d_i + 2dy$ и $y_{i+1} = y_i$. Если $d_i \geq 0$, то $d_{i+1} = d_i + 2(dy - dx)$

и $y_{i+1} = y_i + 1$.

5. x_{i+1}=x_i+1; $i = i$+1.

6. Если $x_i < x_2$, то перейти на пункт 3, иначе − конец.

Алгоритм Брезенхема

Если $dy > dx$, то необходимо будет использовать этот же алгоритм, но пошагово увеличивая y ($y_{i+1} = y_i + 1$) и на каждом шаге вычислять x ($x_i = x_i + 1$ или $x_i = x_i$), пока $y_i < y_2$.

Реализация этого алгоритма может выглядеть следующим образом:

```
void drawDot(GLint x, GLint y) // рисуем точку
{
 glBegin(GL_POINTS);
 glVertex2i(x,y);
 glEnd();
 glFlush();
}
```

Алгоритм Брезенхема

```
void line(GLint x1, GLint y1, GLint x2, GLint y2) //
алгоритм Брезенхема для отрезка
{
int dy=abs(y2-y1);
int dx=abs(x2-x1);
int tmp_x, tmp_y,x,y, flag;

if (dx==0) {             // вертикальная линия
if (y2<y1) {tmp_y=y2;y2=y1; y1=tmp_y;}
for (y=y1;y<=y2;y++)
drawDot(x1,y);
return;
          }
```

```
if (dy<dx){                    // наклонная линия
flag=0;
if (x2<x1) {tmp_x=x2; x2=x1; x1=tmp_x; tmp_y=y2;y2=y1;
y1=tmp_y;}
if (y2<y1){flag=1;}
int di;
di=2*dy-dx;
y=y1;
x=x1;
do {
    drawDot(x,y);
    if (di<0) {di=di+2*dy; x++;}
    else  {di=di+2*(dy-dx);  x++;  if  (flag==0)  {y++;}
else {y--;}}
     } while (x<x2);
return;
          }
```

Растровая развертка окружности

$$x^2 + y^2 = R^2.$$

Чтобы изобразить часть окружности, будем изменять x с единичным шагом от 0 до R и на каждом шаге вычислять y.

$$y = \pm \sqrt{R^2 - x^2}$$

Растровая развертка окружности

использование параметрического представления окружности

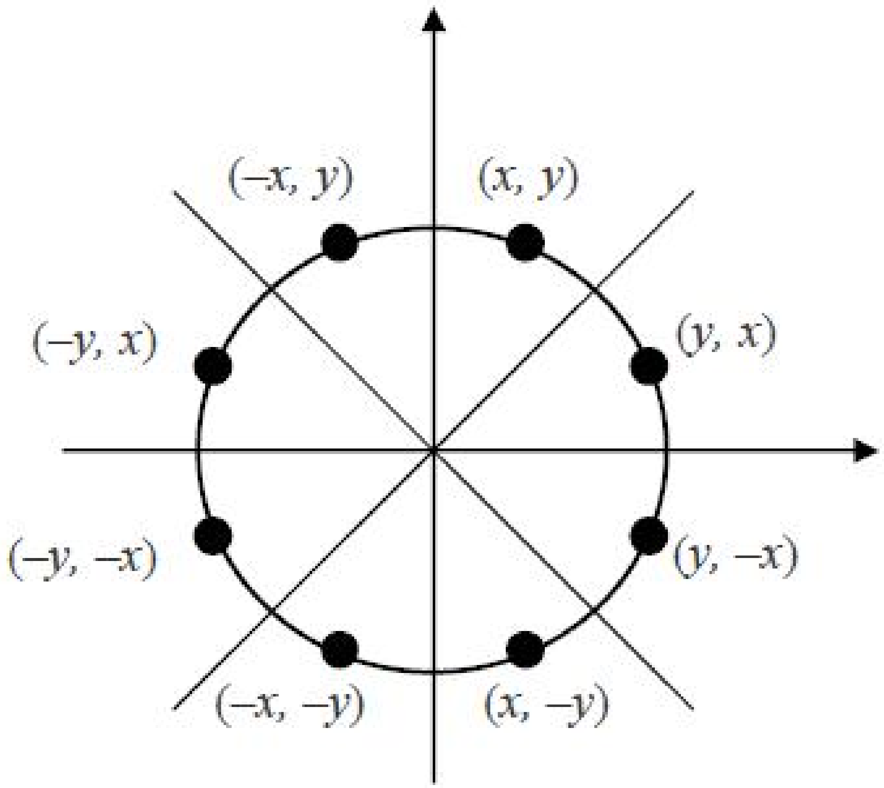
$$x = R\, cos\alpha,$$

$$y = R\, sin\alpha$$

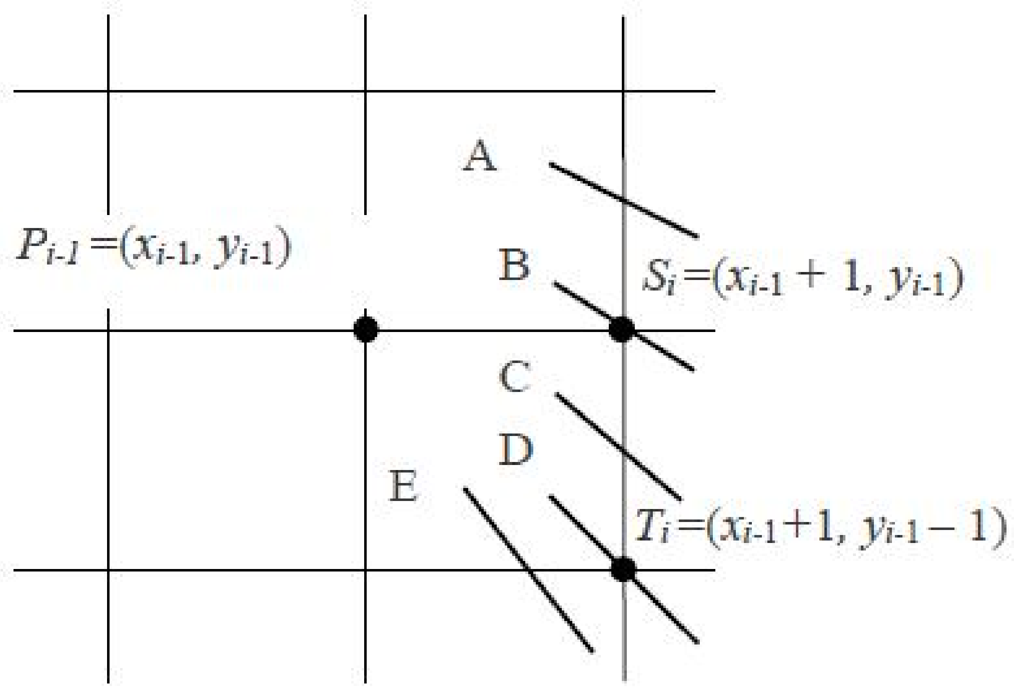
при пошаговом изменении угла α от 0°до 360°.

Растровая развертка окружности

```
void cir_pix(int x0, int y0, int x, int y) // 8 точек
на окружности
{
drawDot(x0+x,y0+y);
drawDot(x0+y,y0+x);
drawDot(x0+x,y0-y);
drawDot(x0+y,y0-x);
drawDot(x0-x,y0-y);
drawDot(x0-y,y0-x);
drawDot(x0-x,y0+y);
drawDot(x0-y,y0+x);
}
```



Алгоритм Брезенхема



Предположим, что точка P_{i-1} была выбрана как ближайшая к окружности при $x = x_{i-1}$. Теперь найдем, какая из точек (S_i или T_i) расположена ближе к окружности при $x = x_{i-1} + 1$.

Алгоритм Брезенхема

Заметим, что ошибка при выборе точки P_i (x_i, y_i) была равна

$$D(P_i) = (x_i^2 + y_i^2) - R^2.$$

Запишем выражение для ошибок, получаемых при выборе точки S_i или T_i:

$$D(S_i) = [(x_{i-1} + 1)^2 + (y_{i-1})^2] - R^2$$

$$D(T_i) = [(x_{i-1} + 1)^2 + (y_{i-1} - 1)^2] - R^2.$$

Если $|D(S_i)| \geq |D(T_i)|$, то T_i ближе к реальной окружности, иначе выбирается S_i.

Алгоритм Брезенхема

Введем $d_i = |D(S_i)| - |D(T_i)|$.

T_i будет выбираться при $d_i \geq 0$, в противном случае будет устанавливаться S_i.

Опуская алгебраические преобразования, запишем d_i и d_{i+1} для разных вариантов выбора точки S_i или T_i.

$$D_1 = 3 - 2\,R.$$

Если выбирается S_i (когда $d_i < 0$), то $d_{i+1} = d_i + 4\,x_{i-1} + 6$.

Если выбирается T_i (когда $d_i \geq 0$), то $d_{i+1} = d_i + 4\,(x_{i-1} - y_{i-1}) + 10$.

Алгоритм Брезенхема

Алгоритм формулируется следующим образом. Дано (x_0, y_0) – центр окружности, R – радиус.

1. Берем первую точку $i=1$, $x_i = 0$, $y_i = R$, рассчитываем $d_i = 3 - 2R$.

2. Вызов функции рисования 8-ми точек на окружности по координатам одной точки, учитывая смещение центра окружности от начала координат.

3. Если $d_i < 0$, то $d_{i+1} = d_i + 4x_i + 6$, $x_{i+1}=x_i+1$, , $y_{i+1} = y_i$.

Если $d_i \geq 0$, то $d_{i+1} = d_i + 4(x_{i-1} - y_{i-1}) + 10$, $x_{i+1}=x_i+1$, $y_{i+1} = y_i - 1$.

4. $i = i + 1$.

5. Если $x_i > R/\sqrt{2}$, то перейти на пункт 2, иначе − конец.

Кривая Безье

$$x = P_x(t), \quad y = P_y(t).$$

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

$$C_m^i = \frac{m!}{i!(m-i)!}.$$

Кривая Безье

1. $m = 1$ (по двум точкам).

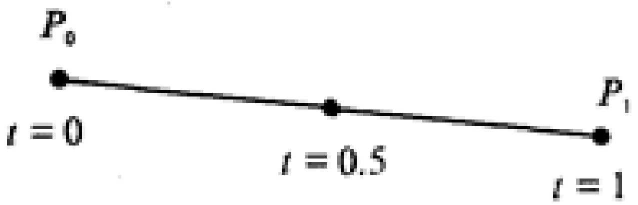
$$P(t) = (1-t)P_0 + tP_1$$

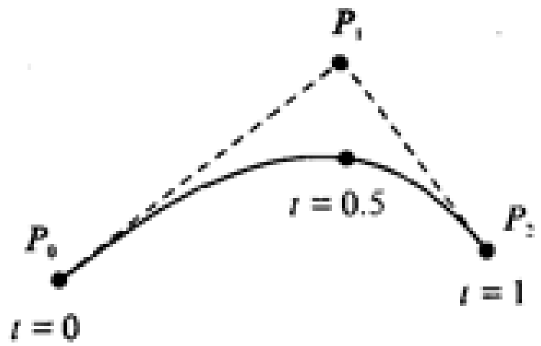
$$x(t) = (1-t)\cdot x_0 + t\cdot x_1$$

$$y(t) = (1-t)\cdot y_0 + t\cdot y_1$$

2. $m = 2$ (по трем точкам, рис. 12):

$$P(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2.$$

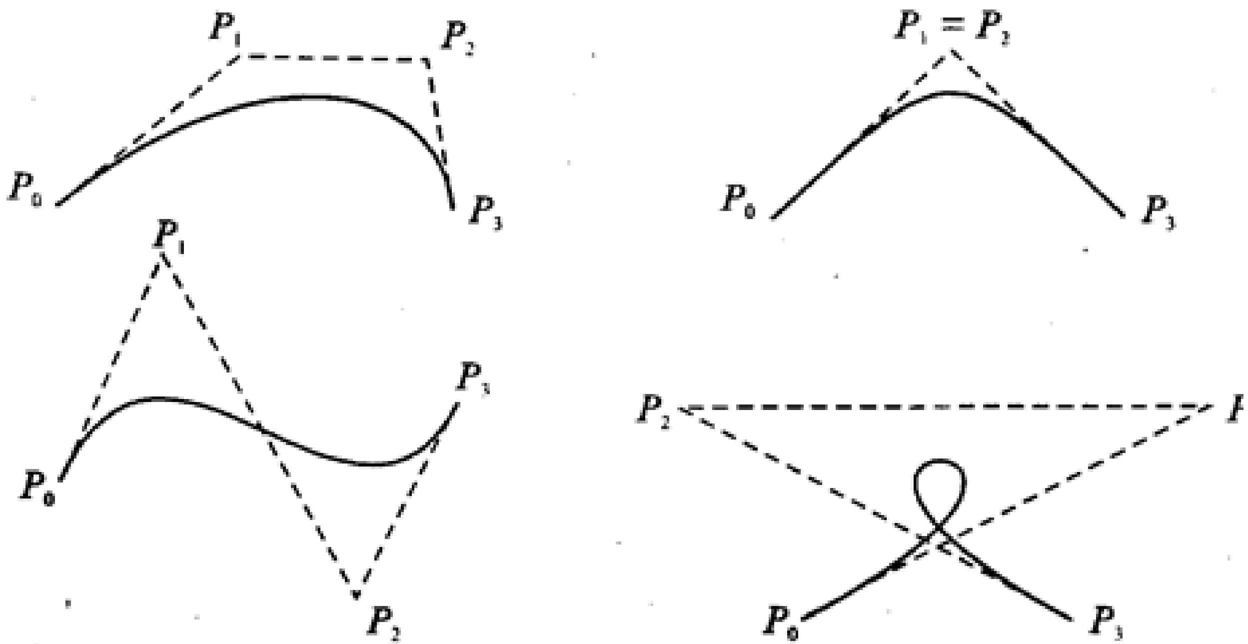




Кривая Безье

3. $m = 3$ (по четырем точкам, кубическая.

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3.$$



Геометрический алгоритм для кривой Безье

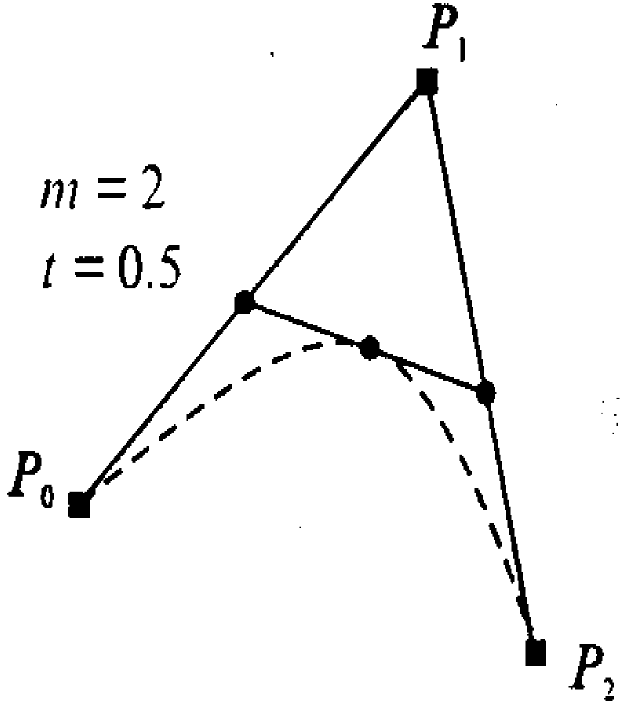
Этот алгоритм позволяет вычислить координаты (x, y) точки кривой Безье по значению параметра t.

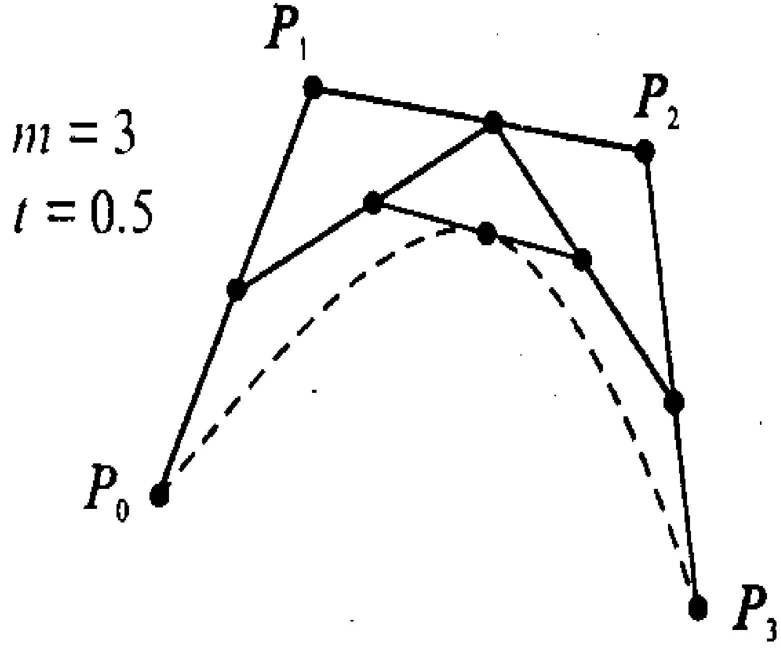
1. Каждая сторона контура многоугольника, который проходит по точкам-ориентирам, делится пропорционально значению t.

2. Точки деления соединяются отрезками прямых и образуют новый многоугольник. Количество узлов нового контура на единицу меньше, чем количество узлов предшествующего контура.

3. Стороны нового контура снова делятся пропорционально значению t. И так далее. Это продолжается до тех пор, пока не будет получена единственная точка деления. Эта точка и будет точкой кривой Безье .

Геометрический алгоритм для кривой Безье





Кривая Безье

Пример. Пусть заданы вершины многоугольника Безье $P_0[1\ 1]$, $P_1[2\ 3]$, $P_2[4\ 3]$ и $P_3[3\ 1]$. Найдем семь точек, лежащих на кривой Безье.

В общем виде многочлен Безье имеет вид:

$$\mathbf{P}(t) = (1-t)^m \mathbf{P}_0 + \sum_{i=1}^{m-1} C_m^i \, t^i (1-t)^{m-i} \mathbf{P}_i + t^m \mathbf{P}_m.$$

В нашем случае $m = 3$ (по четырем точкам, кубическая)

$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3\,t^2(1-t)\,P_2 + t^3 P_3.$$

Кривая Безье

$P(0) = P0 = [1\ 1],$

$P(0.15) = 0.614P_0 + 0.325\ P_1 + 0.058\ P_2 + 0.003\ P_3 = [1.5\ 1.765],$

$P(0.35) = 0.275\ P_0 + 0.444\ P_1 + 0.239\ P_2 + 0.042\ P_3 = [2.248\ 2.367],$

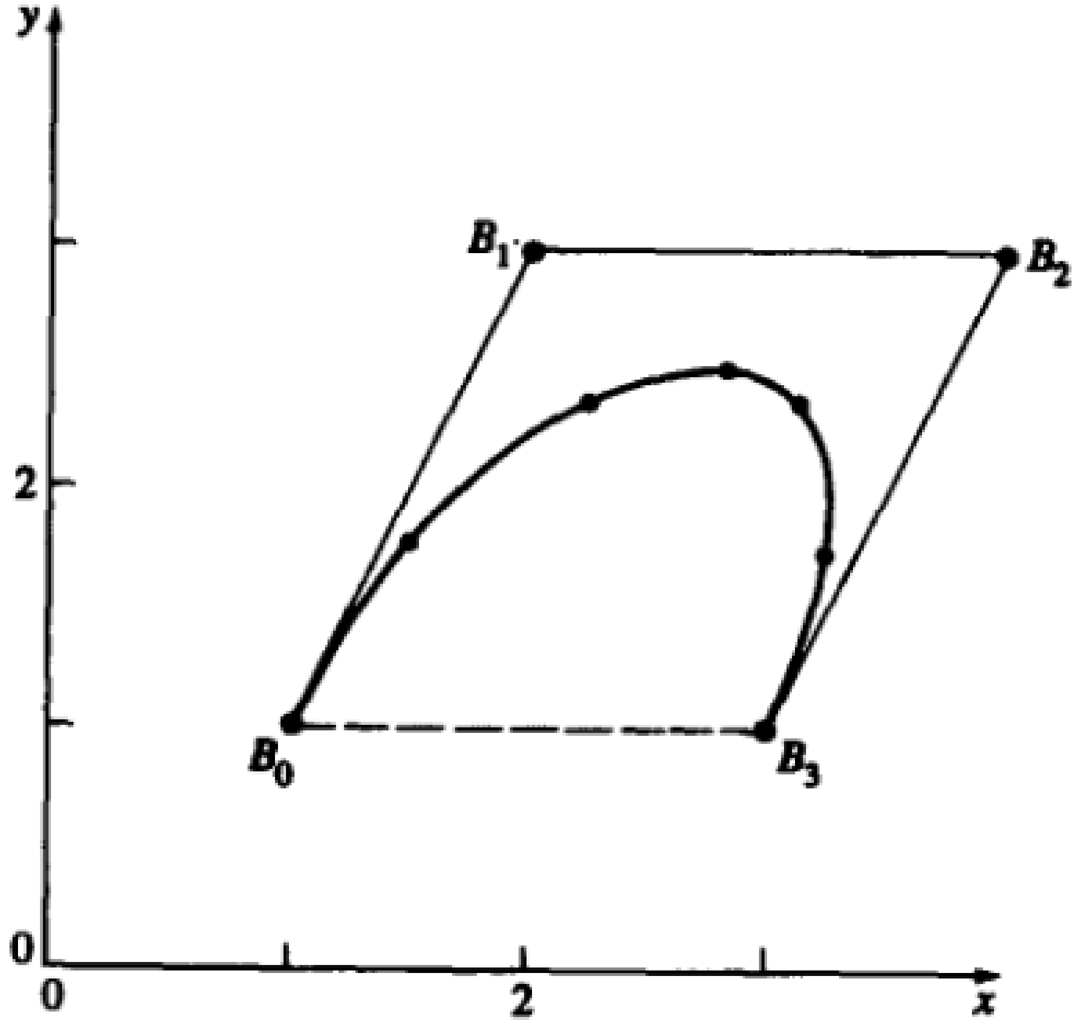
$P(0.5) = 0.125\ P_0 + 0.375\ P_1 + 0.375\ P_2 + 0.125\ P_3 = [2.75\ 2.5],$

$P(0.65) = 0.042\ P_0 + 0.239\ P_1 + 0.444\ P_3 + 0.275\ P_3 = [3.122\ 2.367],$

$P(0.85) = 0.003\ P_0 + 0.058\ P_1 + 0.325\ P_3 + 0.614\ P_3 = [3.248\ 1.765],$

$P(1) = P_3 = [3\ 1].$

Кривая Безье



Матричная форма кривой Безье

$$P(t) = [T][N][G]$$

$$P(t) = [T][N][G] = [t^3 \quad t^2 \quad t \quad 1] \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}.$$

$$P(t) = [t_4 \quad t_3 \quad t_2 \quad t \quad 1] \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \\ -4 & 12 & -12 & 4 & 0 \\ 6 & -12 & 6 & 0 & 0 \\ -4 & 4 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Матричная форма кривой Безье

$$P(t) = [T][N][G]$$

Матрица N в общем виде

$$\begin{bmatrix} \binom{n}{0}\binom{n}{n}(-1)^n & \binom{n}{1}\binom{n-1}{n-1}(-1)^{n-1} & \cdots & \binom{n}{n}\binom{n-n}{n-n}(-1)^0 \\ \binom{n}{0}\binom{n}{n-1}(-1)^{n-1} & \binom{n}{1}\binom{n-1}{n-2}(-1)^{n-2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ \binom{n}{0}\binom{n}{1}(-1)^1 & \binom{n}{1}\binom{n-1}{0}(-1)^0 & \cdots & 0 \\ \binom{n}{0}\binom{n}{0}(-1)^0 & 0 & \cdots & 0 \end{bmatrix}$$

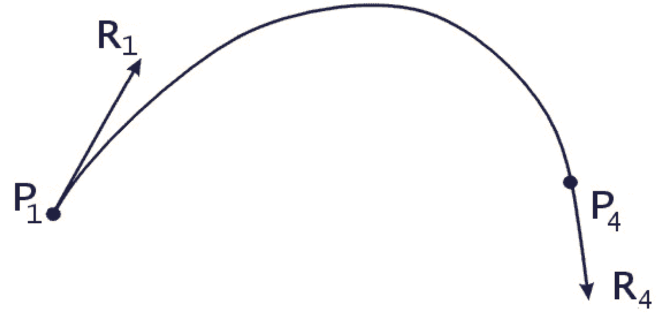
Сплайн Эрмита

$$x(t) = TM_h G_{hx}$$

$$y(t) = TM_h G_{hy}$$

$$z(t) = TM_h G_{hz}$$

$$= \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix},$$



B-сплайны

$$x(t) = TM_s G_{sx},$$

…

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

$$G_s^i = \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}, \quad 2 \le i \le n-2.$$