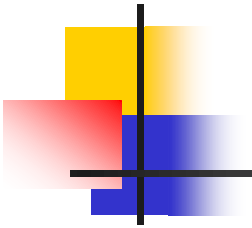
Компьютерная графика

лекция 2

Растровые алгоритмы и характеристики растра

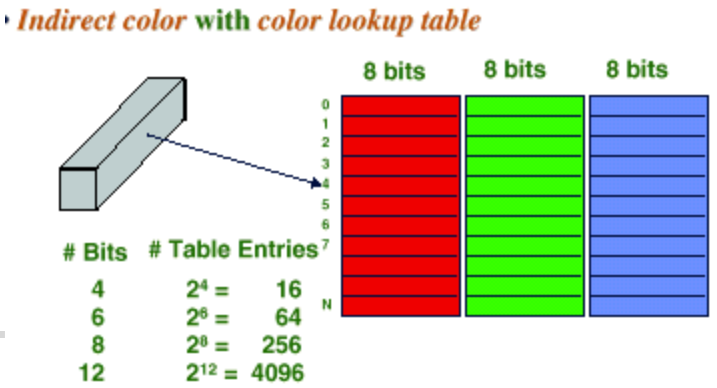


Глубина цвета

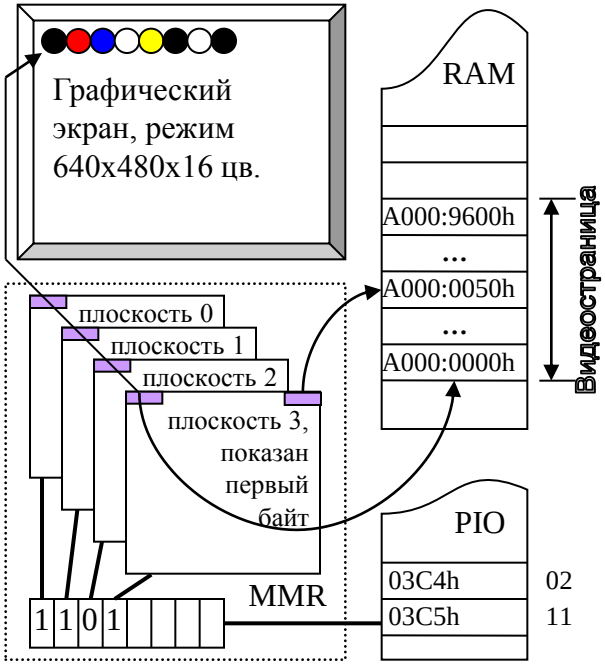
Количество цветов (глубина цвета) – одна из основных характеристик изображения или устройства графического вывода. Согласно психофизиологическим исследованиям глаз человека способен различать 350 000 цветов.
Однако в компьютерной графике в настоящее время используются изображения с формально гораздо большей глубиной цвета (16.7 млн цветов), тем не менее следует учитывать, что для синтезированных цветов на каждый из компонентов цвета в этом случае отводится только 256 градаций, которые достаточно хорошо различимы глазом человека.

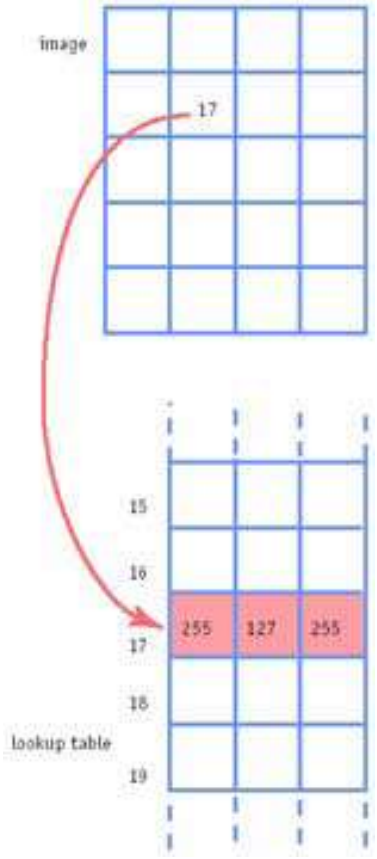
Классифицируем изображения следующим образом:
бинарные – 1 бит на пиксель – обычно чёрно-белые изображения
полутоновые – 1 байт на пиксель – изображение в градациях серого
Hi Color – 16 бит на пиксель – 65536 цветов
True Color – от 24 бит на пиксель (16,7 млн. цветов) до 48 бит на пиксель

Палитра



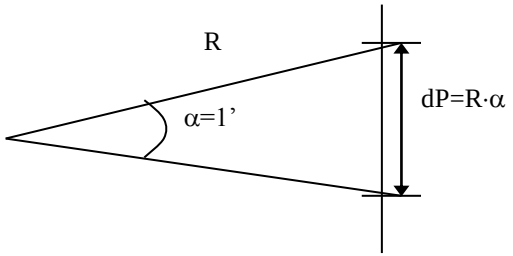
Палитра (palette) – набор цветов, используемых в изображении или при отображении видеоданных. Палитру можно воспринимать как таблицу кодов цветов (обычно в виде RGB-троек байтов). Палитра устанавливает взаимосвязь между кодом цвета и его компонентами в выбранной цветовой модели. Палитра может принадлежать изображению, части изображения, операционной системе или видеокарте.





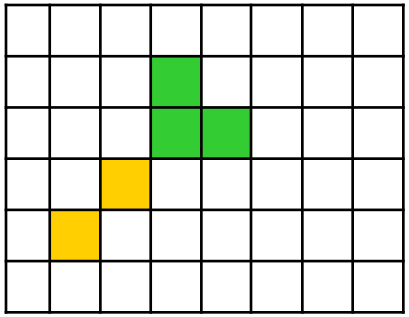
Геометрические характеристики растра

Разрешающая способность характеризует расстояние между соседними точками растрового изображения и измеряется обычно в ***dpi (dots per inch)***



dpi = 25.4 / dP

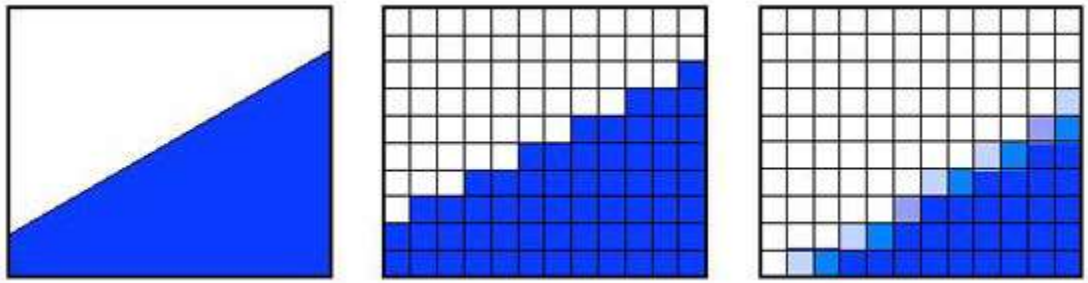
Связность – соседство двух пикселей в растровом изображении

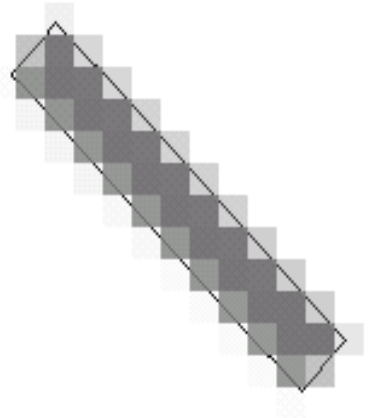


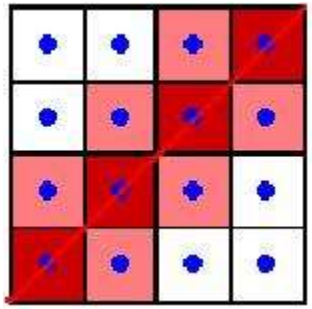
Расстояние R, мм	*Размер dP, мм*	*Разр. способность, dpi*	*Комментарии*
500	**0.14**	**181**	Расстояние, на котором рекомендуется работать с монитором.
300	**0.09**	**282**	Расстояние, на котором просматриваются бумажные документы.

Методы улучшения растровых изображений: anti-aliasing (устранение ступенчатого эффекта)

$$C_x = \frac{C \cdot S_1 + C_f \cdot S_2}{S_1 + S_2}$$





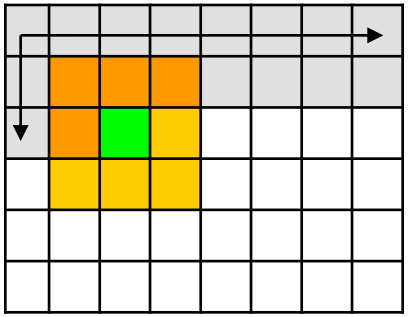


Мультисэмплинг 4x – один из вариантов антиалисинга: пиксель делится на 4 равные части.

Методы улучшения растровых изображений: smoothing (сглаживающие фильтры)

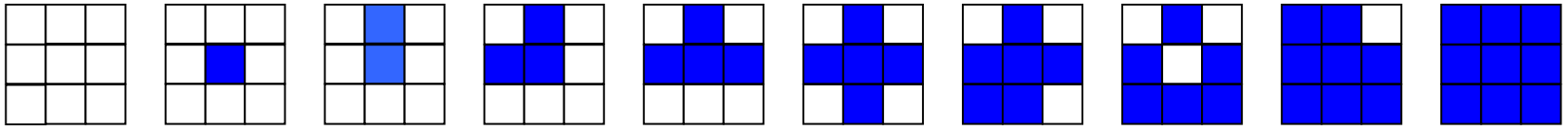
$$F_{x,y} = \frac{1}{K} \sum_{i=0}^{L} \sum_{j=0}^{L} C_{x+i,y+j} \cdot M_{i,j}$$

рекурсивная фильтрация



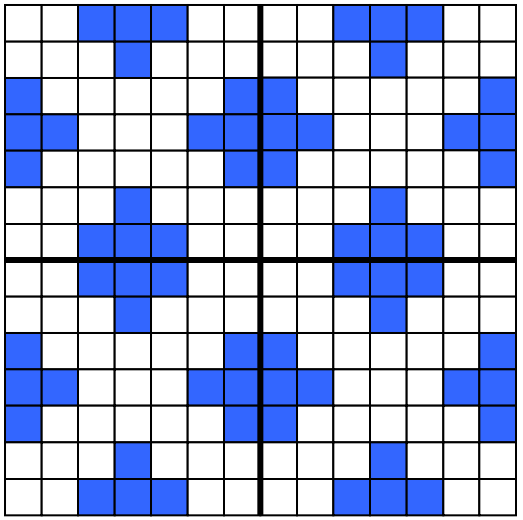
$$M_1 = \frac{1}{4} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad M_2 = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Методы улучшения растровых изображений: dithering (эмуляция оттенков цвета)



Для ячейки с размерами *n* x *n* можно получить n^2+1 различных градаций

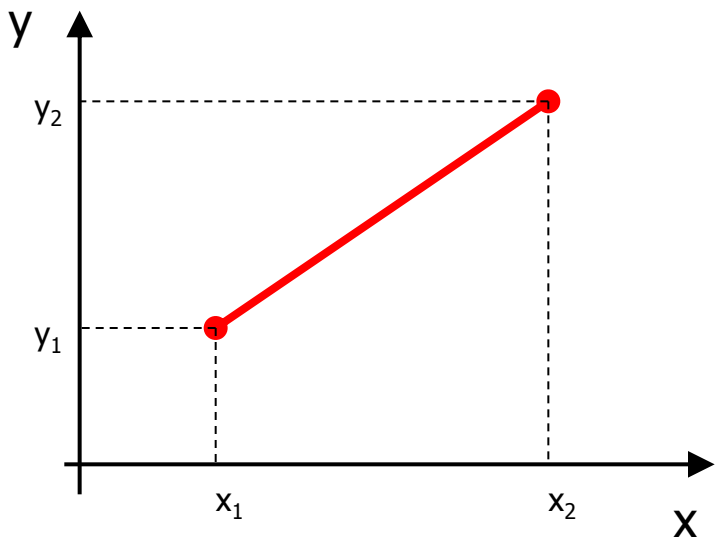
$$C = \frac{m_1 C_1 + m_2 C_2}{m_1 + m_2}, \quad m_1 + m_2 = n^2.$$



При регулярном расположении одинаковых ячеек образуются *паразитные текстуры*, *муар*. Поэтому наряду с ячейками с фиксированным рисунком используются методы *частотно-модулированного дизеринга* (равномерное псевдослучайное распределение пикселей по ячейке) и *диффузного дизеринга* (распределение в каждой ячейке создаётся случайным образом).

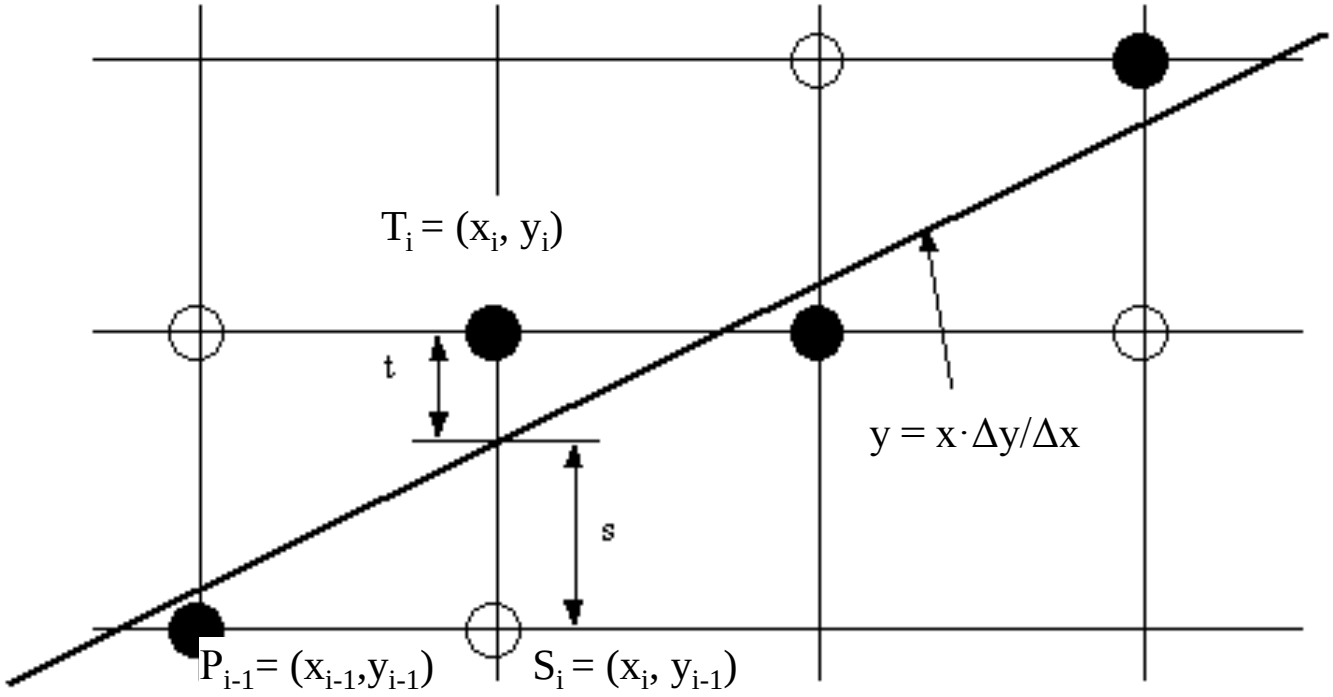
Растровые алгоритмы рисования прямых и эллипсов: постановка проблемы

$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$

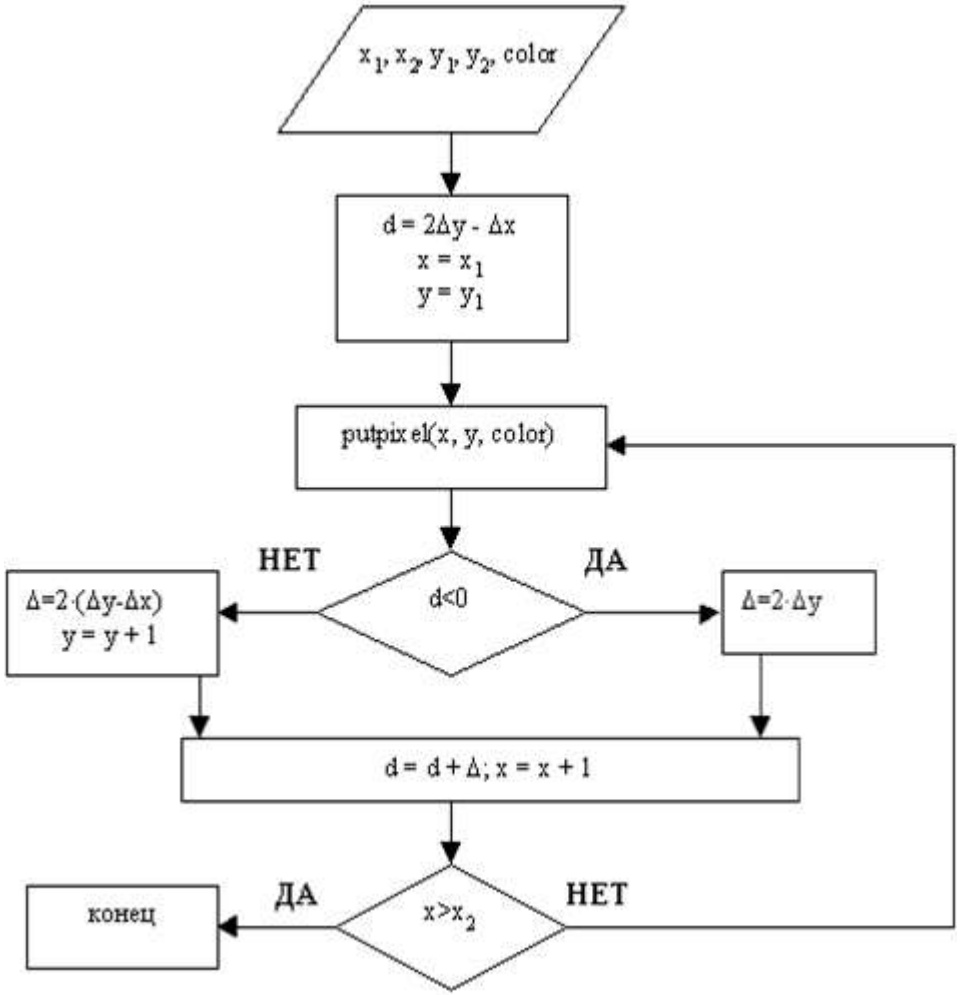



```
float k = (float) (y2-y1) / (float) (x2-x1);
float yy = y1;
for (x=x1; x<=x2; x++)
{
  yy += k; y = yy
  SetPixel(x,y);
}
```

Растровые алгоритмы рисования прямых: инкрементный алгоритм Брезенхема



Растровые алгоритмы рисования прямых: инкрементный алгоритм Брезенхема



Для 8-связных пикселей:

```
    int dx = x2 - x1;
int dy = y2 - y1;
int d = 2 * dy - dx;
    int d1 = 2*dy;
    int d2 = 2*(dy - dx);
    SetPixel(x1, y1, color);

    for (int x = x1 + 1; int y = y1; x <= x2; x++)
    {
  if (d < 0) {
              d += d1;
      } else {
    d += d2;
    y += 1;
  }

    SetPixel( x, y, color);
}
```

Растровые алгоритмы рисования прямых: инкрементный алгоритм Брезенхема

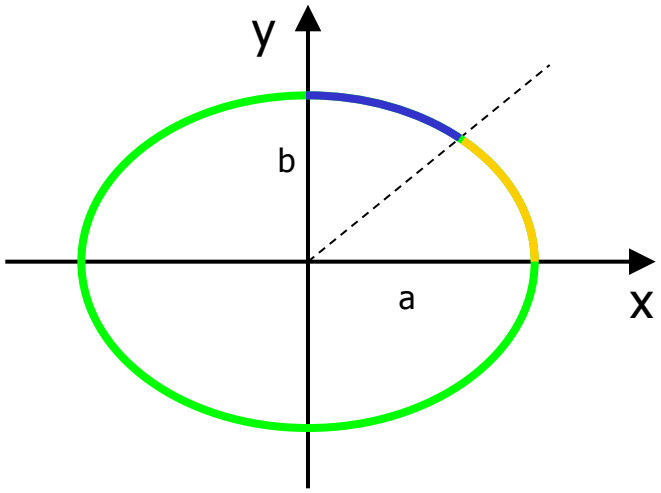
Для 4-связных пикселей:

```
void Line4 (int x1, int y1, int x2, int y2, int color)
{
   int dx = x2 - x1;
   int dy = y2 - y1;
   int d = 0;
   int d1 = dy << 1;
   Int d2 = - ( dx << 1 );
   SetPixel (x1, y1, color);
   for ( int x = x1, y = y1, i = 1; i <= dx + dy; i++ ) {
         if ( d > 0 ) {
               d += d2; y += 1;
         } else {
               d += d1; x += 1;
         }
         SetPixel ( x, y, color);
   }
}
```

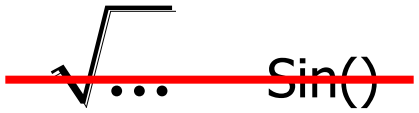
$b^2x^2 + a^2y^2 = a^2b^2$

Растровые алгоритмы рисования прямых: рисование окружностей и эллипсов

$y^2 = r^2 - x^2$

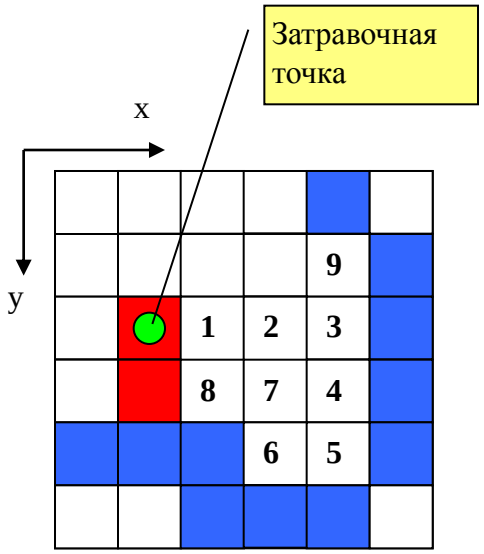



```
void Circle(int Xc,int Yc,int radius,int c)
{
        int x,y,d;
        d = 3 - ( radius << 1);
        x = 0; y = radius;
        While ( x < y ) {
                Set8Pixels( Xc, Yc, x, y, c );
                If ( d > 0 ) {
                        d = d + ( x << 2 ) + 6;
                } else {
                        d = d + ( (x-y) << 2 ) + 10;
                        y--;
                }
        x++;
        }
}
```



Растровые алгоритмы закрашивания: простое рекурсивное заполнение

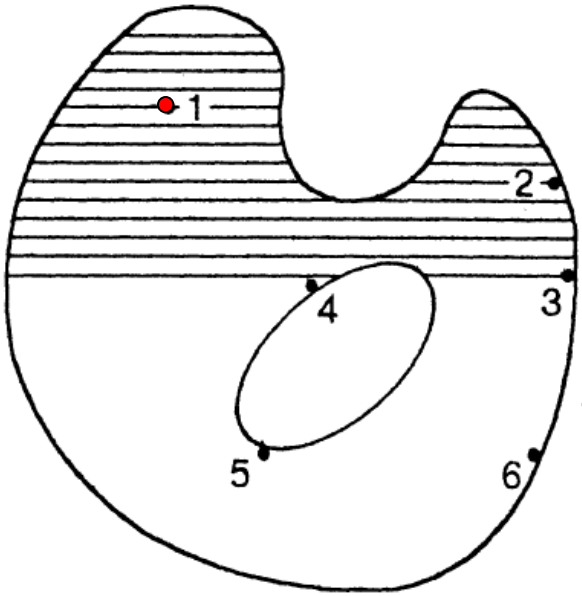
В закрашиваемой области указывается затравочная точка. Область может содержать полости и быть сколь угодно сложной формы. Необходимо лишь, чтобы внешняя граница и границы полостей были 8-связны и их цвет отличался от цвета заполнения.

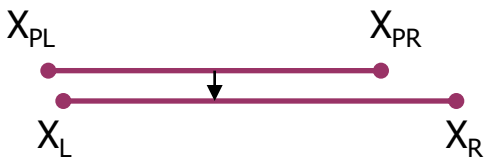



```
void PixelFill (int x, int y, int BC, int color)
{
   int c = GetPixel ( x, y );
   If (( c != BC ) && ( c != color ))
   {
         SetPixel ( x, y, color );
         PixelFill ( x – 1, y, BC, color );
         PixelFill ( x + 1, y, BC, color );
         PixelFill ( x, y – 1, BC, color );
         PixelFill ( x, y + 1, BC, color );
   }
}
```

Растровые алгоритмы закрашивания: алгоритм закрашивания линиями

1. Имеется затравочная точка с координатами (x_{st}, y_{st}) и начальное направление действия рекурсивных вызовов dir=1. Параметры левой и правой границы вначале совпадают с координатой затравочной точки: $x_{PL} = x_{st}$, $x_{PR} = x_{st}$. Вызывается процедура LineFill, в которой:
2. Находятся x_L и x_R – левая и правая границы, между которыми **проводится текущая горизонтальная линия**.
3. Делается приращение y=y+dir и, между x_L и x_R, анализируется цвет пикселей **над** линией. Если он **не** совпадает с цветом заполнения, процедура LineFill вызывается рекурсивно с dir = 1 и $x_{PL} = x_L$, $x_{PR} = x_R$.
4. Делается приращение y=y–dir и, начиная от x_L до предыдущего значения x_{PL} анализируется цвет пикселей **под** линией. Если цвет пикселя отличается от цвета заполнения, процедура вызывается рекурсивно с dir = –1 и $x_{PL} = x_L$, $x_{PR} = x_R$.
5. Продолжая по той же горизонтали от предыдущего x_{PR} до x_R анализируется цвет **под** линией. Если цвет какого-либо пикселя отличается от цвета заполнения, процедура вызывается рекурсивно с dir = –1 и $x_{PL} = x_L$, $x_{PR} = x_R$.





Растровые алгоритмы закрашивания: алгоритм закрашивания линиями

```
int  LineFill ( int x, int y, int dir, int PrevXl, int PrevXr )
{
  int  xl = x;
  int  xr = x;
  int  c;

       // find line segment
  do    c = getpixel ( --xl, y );
  while ( ( c != BorderColor ) && ( c != Color ) );

  do    c = getpixel ( ++xr, y );
  while ( ( c != BorderColor ) && ( c != Color ) );

  xl++;  xr--;

  line ( xl, y, xr, y );  // fill segment
       // fill adjacent segments in the same direction
  for ( x = xl; x<= xr; x++ )
  {
    c = getpixel ( x, y + dir );
    if ( ( c != BorderColor ) && ( c != Color ) )
      x = LineFill ( x, y + dir, dir, xl, xr );
  }

  for ( x = xl; x < PrevXl; x++ )
  {
    c = getpixel ( x, y - dir );
    if ( ( c != BorderColor ) && ( c != Color ) )
      x = LineFill ( x, y - dir, -dir, xl, xr );
  }

  for ( x = PrevXr; x < xr; x++ )
  {
    c = getpixel ( x, y - dir );
    if ( ( c != BorderColor ) && ( c != Color ) )
      x = LineFill ( x, y - dir, -dir, xl, xr );
  }

  return xr;
}
```