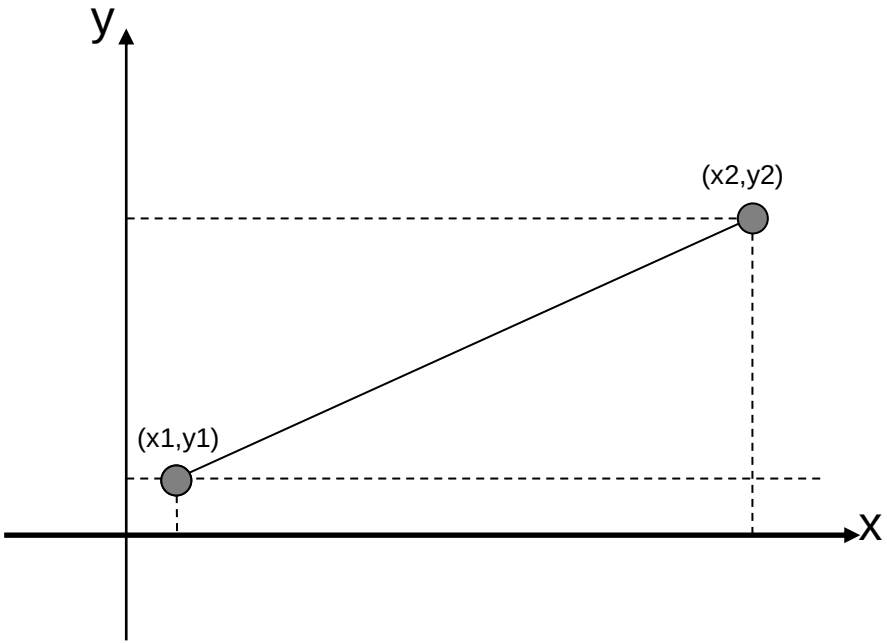
Растровая графика

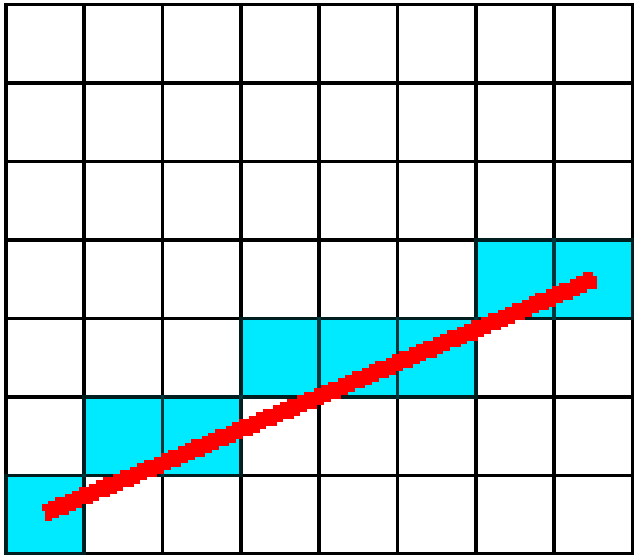
материалы занятий: https://compsciclub.ru/courses/graphics2018/2018-autumn/classes/
дублируются на сайте: http://www.school30.spb.ru/cgsg/cgc2018/

Примитивы

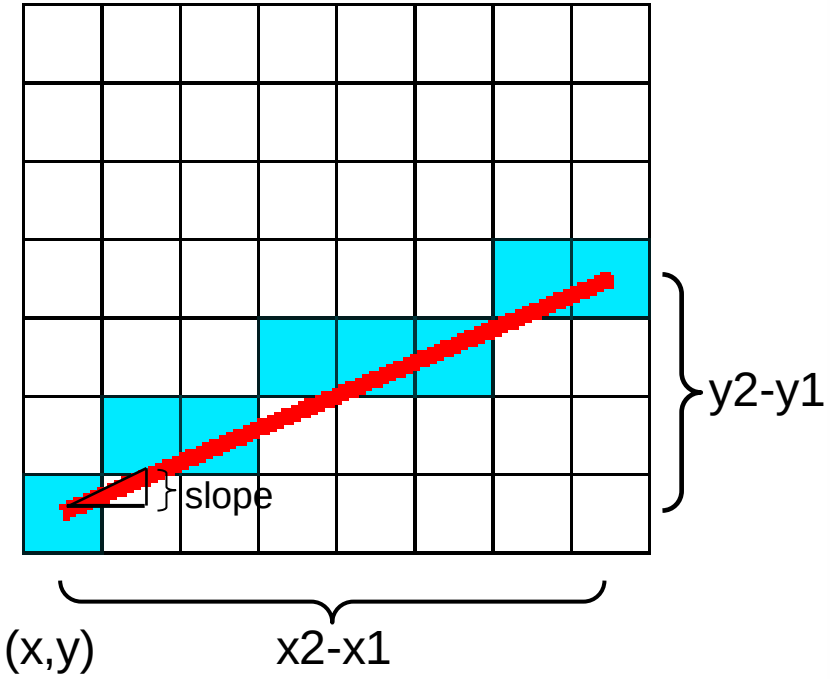
- *Точки*
- *Линии*
- *Прямоугольники (со сторонами, параллельными границам экрана)*
- *Многоугольники*
- *Шрифты*
- *Заливка областей*
- *Плоское отсечение*

Line



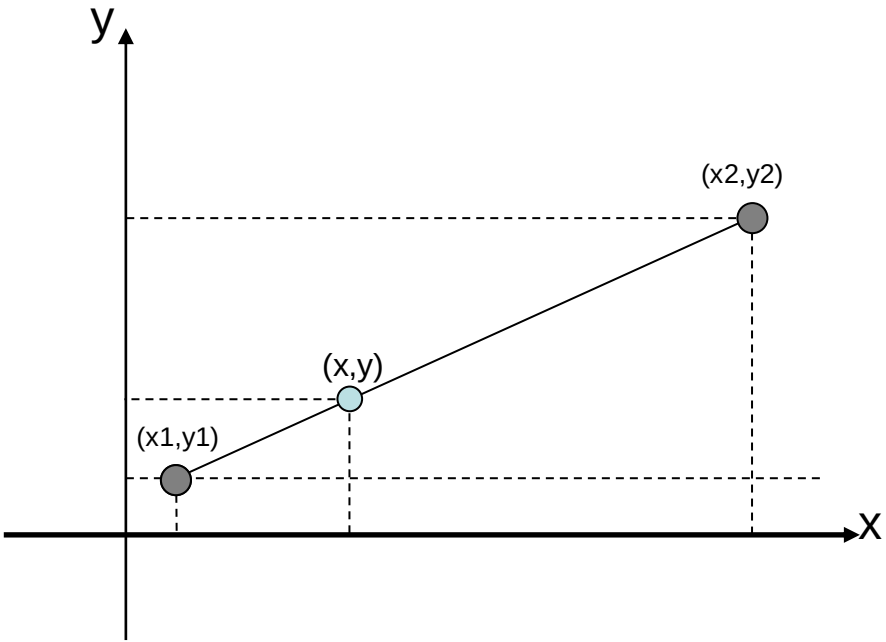


Line: Digital Differential Analyzer (DDA)




```
int x;
float
  y,
  slope = (y2 - y1) / (x2 - x1);

x = x1; y = y1;

while (x <= x2)
{
  SetPixel(x, (int)y);
  y = y + slope;
  x = x + 1;
}
```

Line: Алгоритм Брезенхема (метод центральной точки)



$$\frac{x - x1}{x2 - x1} = \frac{y - y1}{y2 - y1}$$

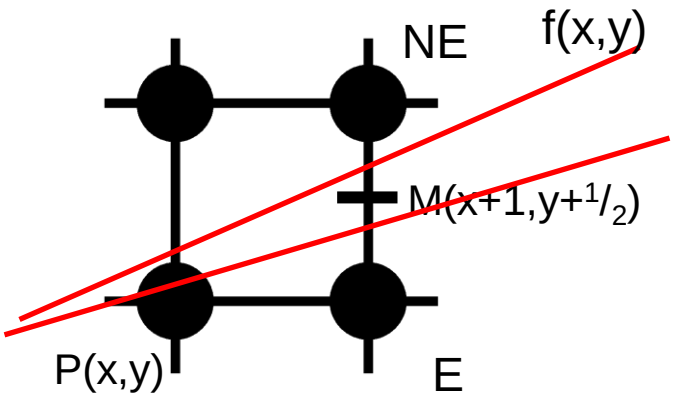
$$(x - x1) \cdot (y2 - y1) - (y - y1) \cdot (x2 - x1) = 0$$

$$dy \cdot x - dx \cdot y - (x1 \cdot dy - y1 \cdot dx) = 0$$

$$f(x, y) = dy \cdot x - dx \cdot y - (x1 \cdot dy - y1 \cdot dx)$$

$$\begin{cases} f(x, y) > 0 & \text{точка } (x,y) \text{ «ниже» прямой} \\ f(x, y) = 0 & \text{точка } (x,y) \text{ «лежит» на прямой} \\ f(x, y) < 0 & \text{точка } (x,y) \text{ «выше» прямой} \end{cases}$$

Line: Алгоритм Брезенхема (метод центральной точки)

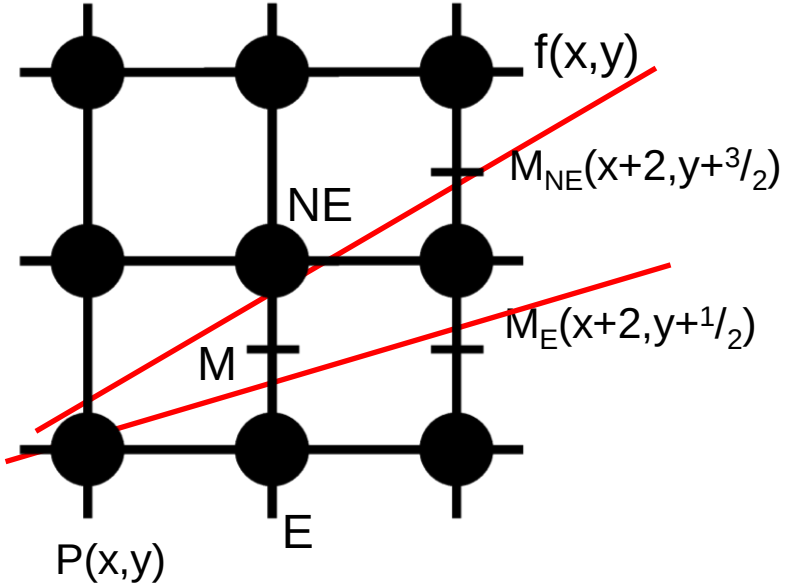


Подставляем точку M в функцию f:

- если f(M) > 0 выбираем точку NE
- если f(M) <= 0 выбираем точку E

```
int x, y;

x = x1; y = y1;

SetPixel(x, y);
while (x <= x2)
{
  if (f(x + 1, y + 0.5) > 0)
    y = y + 1;
  x = x + 1;
  SetPixel(x, y);
}

/* f(x, y) = dy * x - dx * y - (x1 * dy - y1 * dx)
 * dx = x2 - x1; dy = y2 - y1;
 */
```

Line: Алгоритм Брезенхема (метод центральной точки)



Подставляем точку M в функцию f:

- если f(M) > 0 выбираем точку NE
- если f(M) <= 0 выбираем точку E

Изменения значения f(M) при переходе к новым точкам (E или NE):

$$f(M) = f(x+1, y+\tfrac{1}{2}) = dy\cdot(x+1) - dx\cdot(y+\tfrac{1}{2}) - C =$$

$$dy\cdot x + dy - dx\cdot y - \tfrac{dx}{2} - C$$

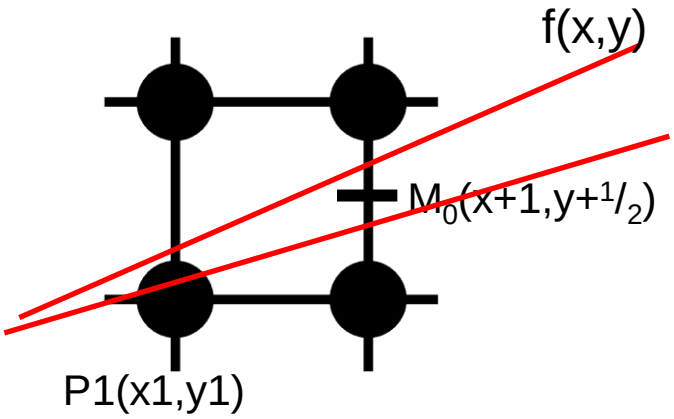
$$f(M_E) = f(x+2, y+\tfrac{1}{2}) = dy\cdot(x+2) - dx\cdot(y+\tfrac{1}{2}) - C =$$

$$dy\cdot x + 2\cdot dy - dx\cdot y - \tfrac{dx}{2} - C = \underline{f(M) + dy}$$

$$f(M_{NE}) = f(x+2, y+\tfrac{3}{2}) = dy\cdot(x+2) - dx\cdot(y+\tfrac{3}{2}) - C =$$

$$dy\cdot x + 2\cdot dy - dx\cdot y - \tfrac{3\cdot dx}{2} - C = \underline{f(M) + dy - dx}$$

Line: Алгоритм Брезенхема (метод центральной точки)



Известны приращения f.

Найдем первоначальное значение для точки (x1,y1)

$$f(M_0) = f(x1+1, y1+\tfrac{1}{2}) =$$

$$dy \cdot (x1+1) - dx \cdot (y1+\tfrac{1}{2}) - (x1 \cdot dy - y1 \cdot dx) =$$

$$dy - dx/2$$

Line: Алгоритм Брезенхема (метод центральной точки)

Сохранились вещественные числа.

Сделаем замену: 2f = e

Тогда помеченные строки изменяться на:

```
e = 2 * dy - dx;
```

```
e > 0
```

```
e = e + 2 * dy - 2 *dx;
```

```
e = e + 2 * dy
```

и e – целое число.

```
int x, y, dx, dy;
float f;

dx = x2 - x1;
dy = y2 - y1;
f = dy - dx / 2.0;

x = x1; y = y1;
SetPixel(x, y);
count = dx;
while (count > 0)
{
  count = count - 1;

  if (f > 0)
  {
    y = y + 1;
    f = f + (dy - dx);
  }
  else
    f = f + dy;
  x = x + 1;
  SetPixel(x, y);
}
```

Line: Алгоритм Брезенхема (метод центральной точки)

```
int x, y, dx, dy, incrE, incrNE, e;

dx = x2 - x1;
dy = y2 - y1;
e = 2 * dy - dx;
incrE = 2 * dy;
incrNE = 2 * dy - 2 * dx;

x = x1; y = y1;
SetPixel(x, y);
count = dx;
while (count > 0)
{
  count = count - 1;

  if (f > 0)
  {
    y = y + 1;
    f = f + incrNE;
  }
  else
    f = f + incrE;
  x = x + 1;
  SetPixel(x, y);
}
```

Line: Алгоритм Брезенхема (метод центральной точки)

Fixed Point – вещественные числа с фиксированной точкой.

Рассмотрим 4-байтное целое:

2b целая часть	2b дробная часть

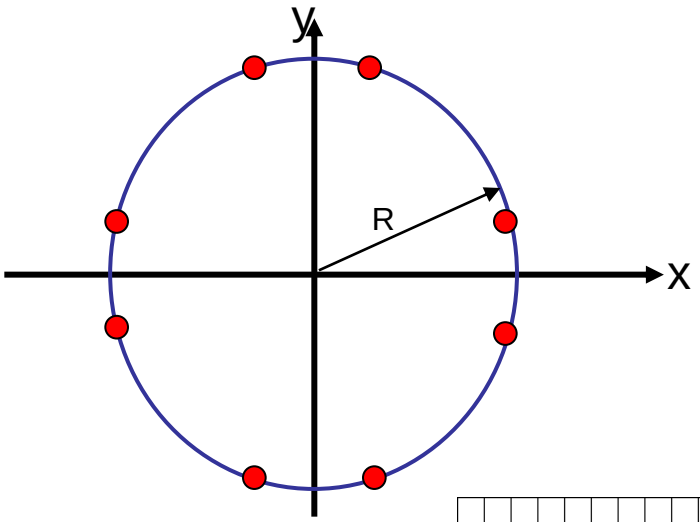
Точность $^{1}/_{65536}$

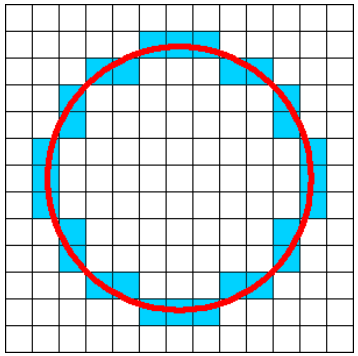
Если x и y fixed point, то

- сложение не изменяется (x+y)
- вычитание не изменяется (x-y)
- целая часть – «двоичный сдвиг» вправо на 16 бит (x >> 16)
- из целого: x = a << 16

```
int x;
long
  y,
  slope = ((y2 - y1) << 16) / (x2 - x1);

x = x1; y = y1 << 16;

SetPixel(x1, y1);
count = x2 - x1;
while (count > 0)
{
  count = count - 1;

  y = y + slope;
  x = x + 1;
  SetPixel(x, y >> 16);
}
```

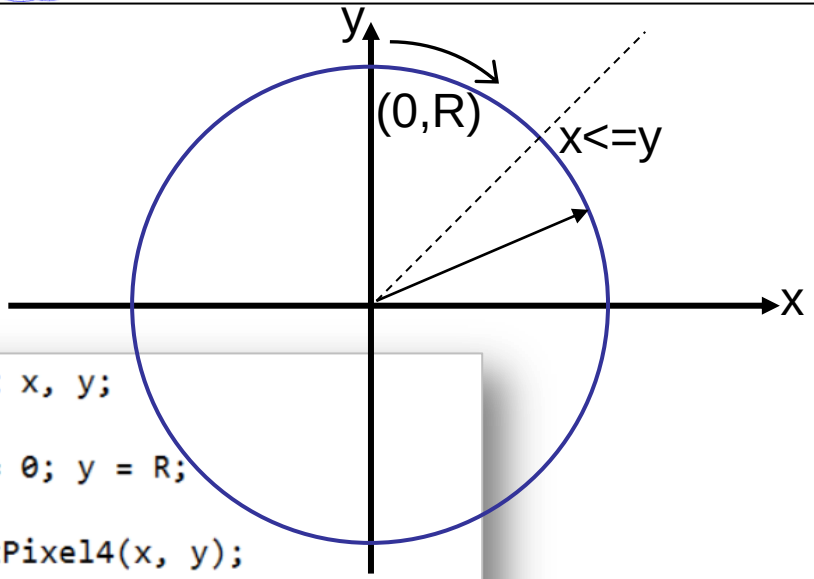
Circle





```
SetPixel4(x, y):
  SetPixel(Cx, Cy + R);
  SetPixel(Cx, Cy - R);
  SetPixel(Cx + R, Cy);
  SetPixel(Cx - R, Cy);

SetPixel8(x, y):
  SetPixel(Cx + x, Cy + y);
  SetPixel(Cx - x, Cy + y);
  SetPixel(Cx + x, Cy - y);
  SetPixel(Cx - x, Cy - y);
  SetPixel(Cx + y, Cy + x);
  SetPixel(Cx - y, Cy + x);
  SetPixel(Cx + y, Cy - x);
  SetPixel(Cx - y, Cy - x);
```

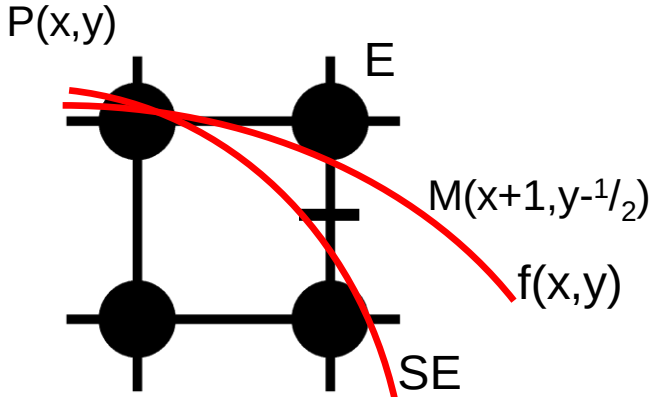
Circle: Алгоритм Брезенхема (метод центральной точки)




```
int x, y;

x = 0; y = R;

SetPixel4(x, y);
while (x <= y)
{
  if (f(x + 1, y - 0.5) < 0)
    y = y - 1;
  x = x + 1;
  SetPixel8(x, y);
}
```

$$f(x,y) = x^2 + y^2 - R^2$$

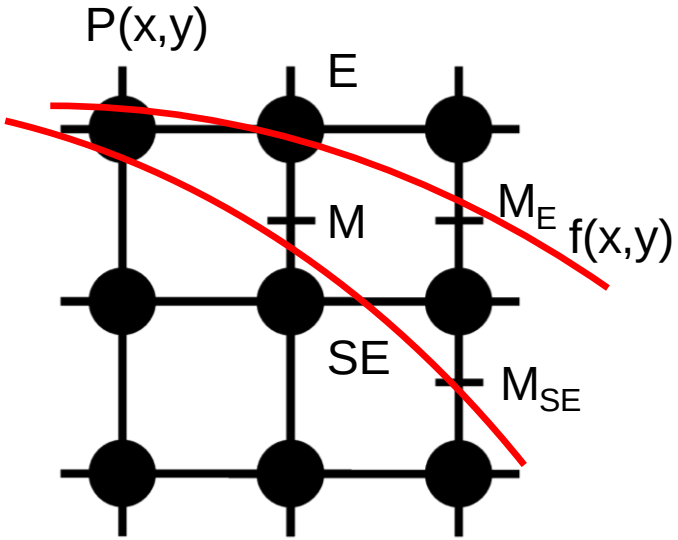
$$\begin{cases} f(x,y) > 0 & \text{точка } (x,y) \text{ вне круга} \\ f(x,y) = 0 & \text{точка } (x,y) \text{ на окружности} \\ f(x,y) < 0 & \text{точка } (x,y) \text{ внутри круга} \end{cases}$$



Подставляем точку M в функцию f:

- если f(M) >= 0 выбираем точку SE
- если f(M) < 0 выбираем точку E

Circle: Алгоритм Брезенхема (метод центральной точки)



Изменения значения f(M) при переходе к новым точкам (E или SE):

$$f(M) = f(x+1, y-\tfrac{1}{2}) = (x+1)^2 + (y-\tfrac{1}{2})^2 - R^2 =$$

$$x^2 + 2x + 1 + y^2 - y + \tfrac{1}{4} - R^2$$

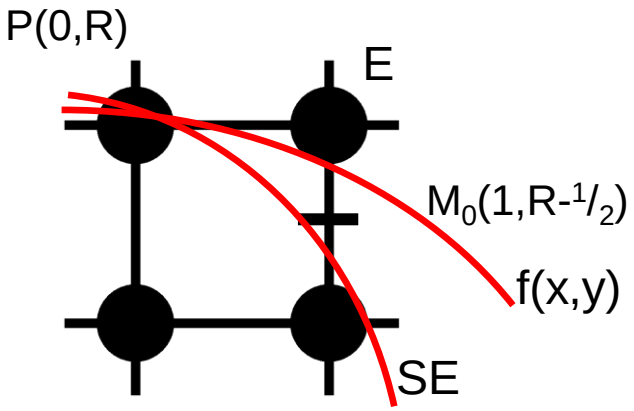
$$f(M_E) = f(x+2, y-\tfrac{1}{2}) = (x+2)^2 + (y-\tfrac{1}{2})^2 - R^2 =$$

$$x^2 + 4x + 4 + y^2 - y + \tfrac{1}{4} - R^2 = f(M) + 2x + 3$$

$$f(M_{SE}) = f(x+2, y-\tfrac{3}{2}) = (x+2)^2 + (y-\tfrac{3}{2})^2 - R^2 =$$

$$x^2 + 4x + 4 + y^2 - 3y + \tfrac{9}{4} - R^2 = f(M) + 2x - 2y + 5$$

Circle: Алгоритм Брезенхема (метод центральной точки)



Определили приращения f.

Найдем первоначальное значение для точки (x1,y1)

$$f(M_0) = f(1, R - \tfrac{1}{2}) = 1^2 + (R - \tfrac{1}{2})^2 - R^2 =$$

$$1 + R^2 - R + \tfrac{1}{4} - R^2 = \tfrac{5}{4} - R$$

```
int x, y, f = 1 - R;

x = 0; y = R;

SetPixel4(x, y);
while (x <= y)
{
  if (f > 0)
  {
    y = y - 1;
    f = f + 2 * (x - y) + 5;
  }
  else
    f = f + 2 * x + 3;
  x = x + 1;
  SetPixel8(x, y);
}
```

Все приращения - целые. Сравнение f с 0 строгое: ‘<‘.

Поэтому из первоначального f можно вычесть $^1/_4$.

Circle: Алгоритм Брезенхема (метод центральной точки)

Дополнительная оптимизация:

Просчитаем изменение приращений по направлениям E и SE (**`incrE=2*x+3`** и **`incrSE=2*(x-y)+5`**) для избавления от доступа к переменным.

- Если выбрана точка E, то ‘x’ увеличивается на 1 и:

  `incrE=incrE+2` и **`incrSE=incrSE+2`**

- Если выбрана точка SE, то ‘x’ увеличивается на 1, ‘y’ уменьшается на 1 и:

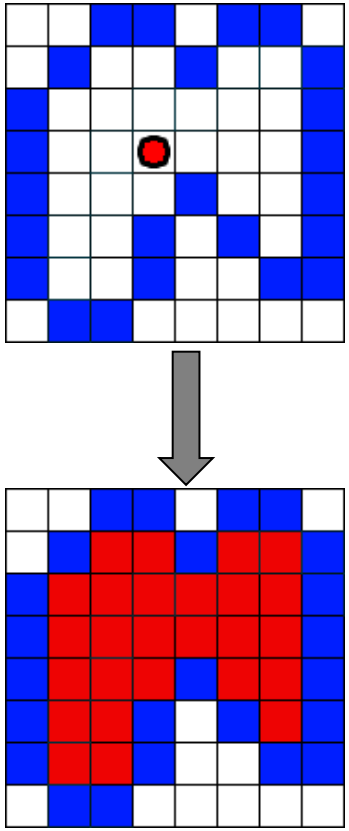
  `incrE=incrE+2` и **`incrSE=incrSE+4`**

Изначальные значения:

`incrE=3` и **`incrSE=5-2*R`**

```
int
  x = 0, y = R,
  f = 1 - R,
  incrE = 3,
  incrSE = 5 - 2 * R;

SetPixel4(x, y);
while (x <= y)
{
  if (f > 0)
  {
    y = y - 1;
    f = f + incrSE;
    incrSE = incrSE + 4;
  }
  else
  {
    f = f + incrE;
    incrSE = incrSE + 2;
  }
  incrE = incrE + 2;
  x = x + 1;
  SetPixel8(x, y);
}
```

Flood Fill



```
Push(x, y)  - сохранить на стеке
Pop(&x, &y) - получить со стека
Get(x, y)   - получить цвет точки
Color - цвет покраски

Back = Get(x, y);
if (Back == Color)
  return;
Push(x, y);
while (стек не пуст)
{
  Pop(&x, &y);
  if (точка(x, y) на экране &&
      Back == Color)
  {
    SetPixel(x, y, Color);
    Push(x + 1, y);
    Push(x - 1, y);
    Push(x, y + 1);
    Push(x, y - 1);
  }
}
```

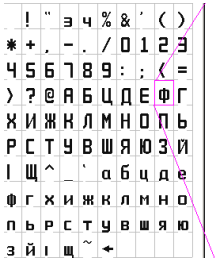
Flood Fill

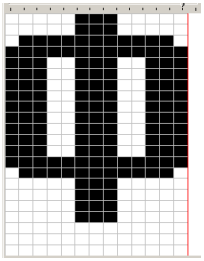
```
Back = Get(x, y);
if (Back == Color)
  return;
Push(x, y);
while (стек не пуст)
{
  Pop(&x, &y);
  /* Поиск границ */
  left = x - 1;
  while (left в экране && Get(left, y) == Back)
    left--;
  left++;
  right = x + 1;
  while (right в экране && Get(right, y) == Back)
    right++;
  right--;
```

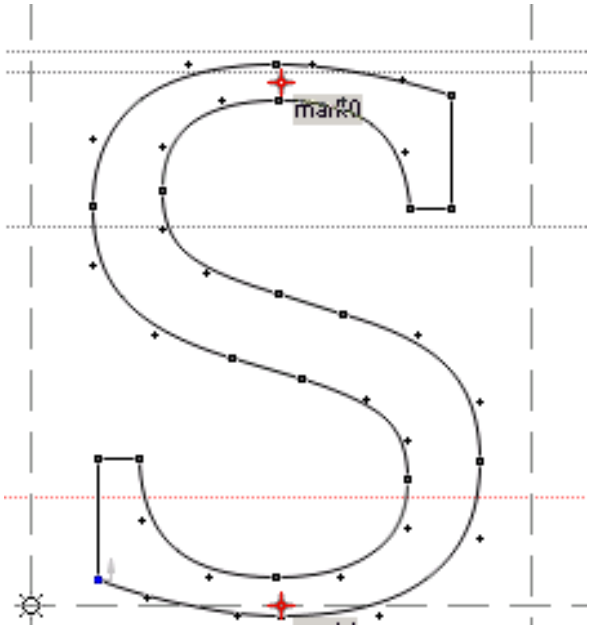
```
  /* Отрисовка уровня */
  Line(left, y, right, y, Color);
  /* Поиск смежных областей */
  pos = left; y = y - 1;
  while (pos <= right)
  {
    /* пропускаем поочередно зоны покраски/непокраски */
    while (pos <= right && Get(pos, y) != Back)
      pos++;
    if (Get(pos, y) == Back)
    {
      Push(pos, y);
      while (pos <= right && Get(pos, y) == Back)
        pos++;
    }
  }
  /* аналогично с уровнем y + 2 */
  . . .
}
```

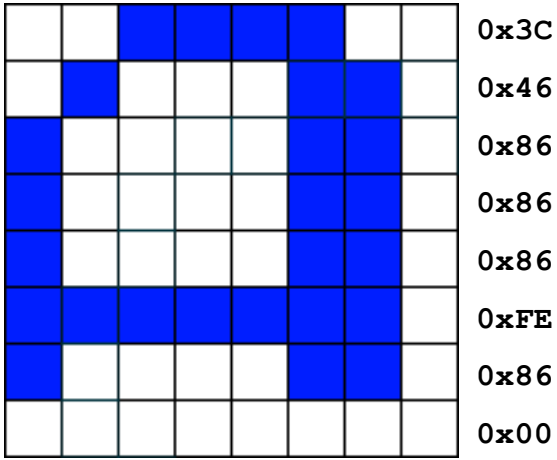
Шрифты

Растровые Векторные Контурные









Справа показана битовая кодировка каждой строки
(в шестнадцатеричном виде)

```
unsigned char *Table = ...;
int i, j;

/* H - высота символов в строках */
Table += КодСимвола * H;
for (i = 0; i < H; i++)
{
  for (j = 0; j < 8; j++)
    if ((Table & (0x80 >> j)) != 0)
      SetPixel(X + j, Y + i);
  Table++;
}
```

Задание

- Реализовать алгоритмы растровой графики на базе функции вывода точки:
  - Вывод отрезка прямой
  - Вывод окружности (дополнительно – круг)
  - Закраска области
  - Вывод строки с помощью растровых шрифтов (шрифты загружать из файла)
- Использовать программу из 1-го задания.

Литература

Дополнительно к литературе из 1-й лекции:

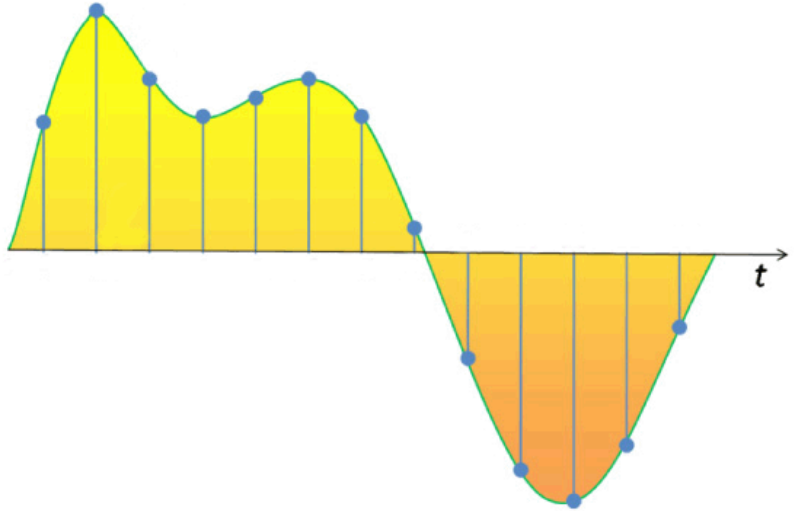
Роджерс Д., "Алгоритмические основы машинной графики", М.: Мир, 1989

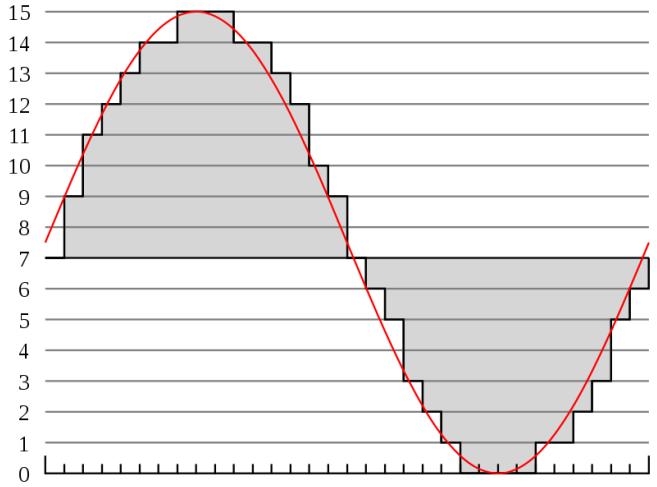
Цвет

материалы занятий: https://compsciclub.ru/courses/graphics2018/2018-autumn/classes/
дублируются на сайте: http://www.school30.spb.ru/cgsg/cgc2018/

Sampling & Quantization

- *Дискретизация сигнала – разбиение непрерывного сигнала на «выборки» (**sampling**, sampling rate)*
- *Квантование выборки – кодирование аналогового сигнала в дискретные величины (**quantization**)*





Sampling (разрешение)

8x8

16x16

32x32

64x64

128x128

256x256

Quantization (глубина цвета)

2 цвета

3 цвета

4 цвета

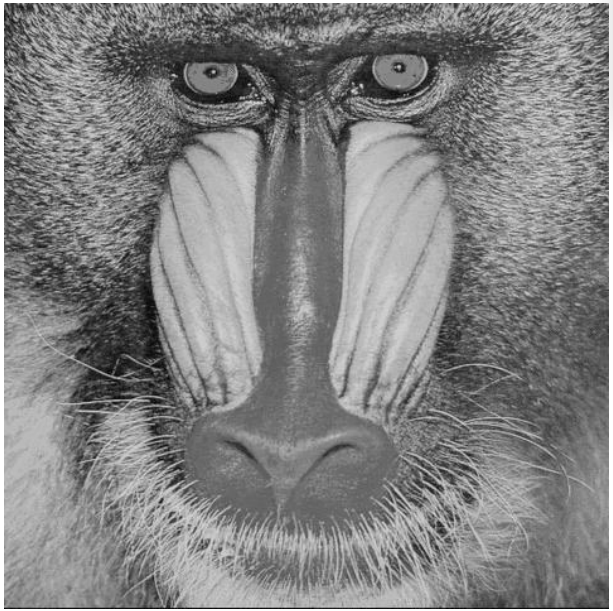
8 цветов

16 цветов

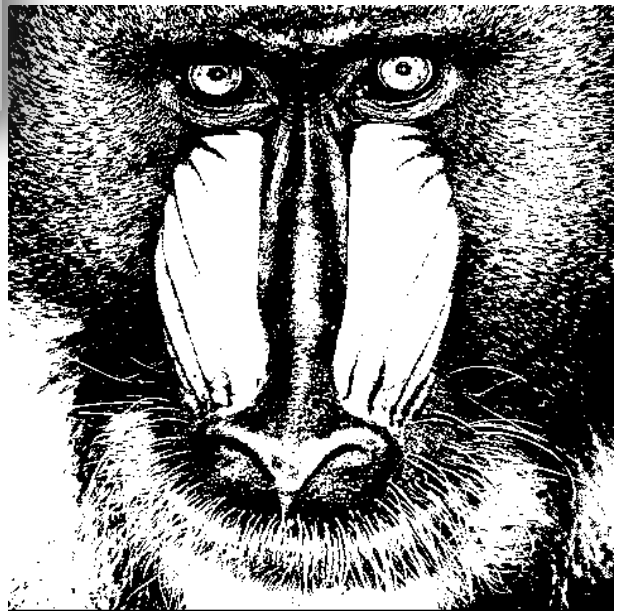
256 цветов

Fixed Thresholding

```
if (Img(x, y) > Threshold)
  color = 1;
else
  color = 0;
```



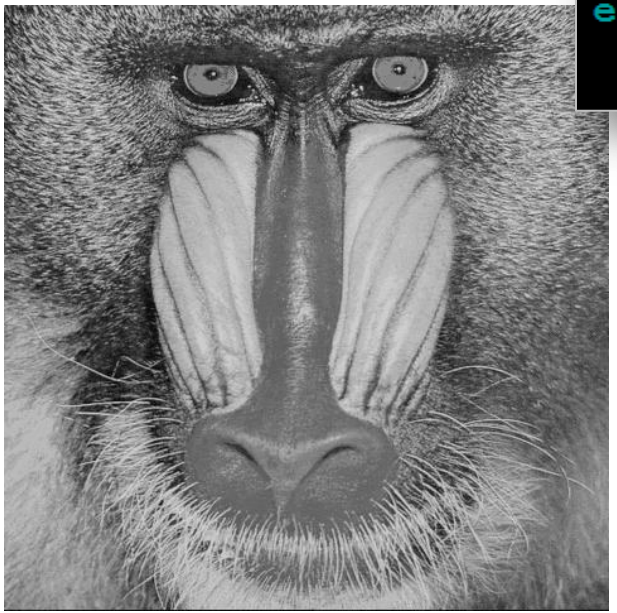
оригинал



порог = 128

Random Thresholding

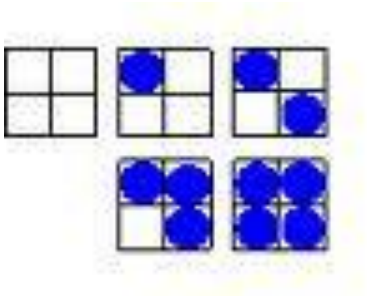
```
if (Img(x, y) > rand() % 255)
  color = 1;
else
  color = 0;
```



оригинал

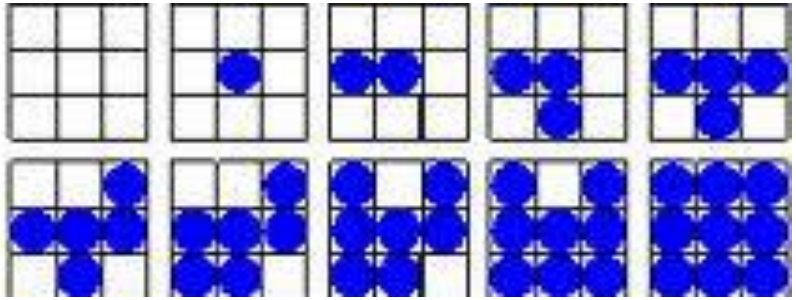
«случайный» порог

Patterning



5 уровней

(2x2)

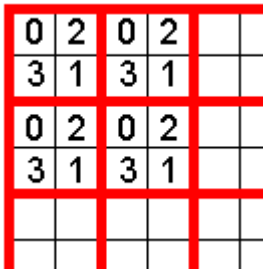


10 уровней

(3x3)

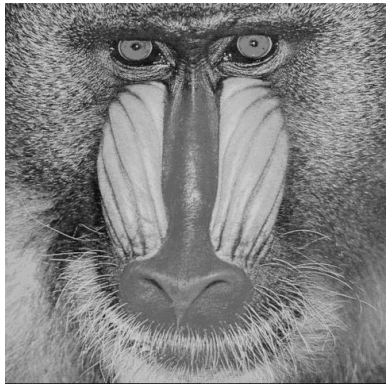
Ordered Dither

0 2
3 1

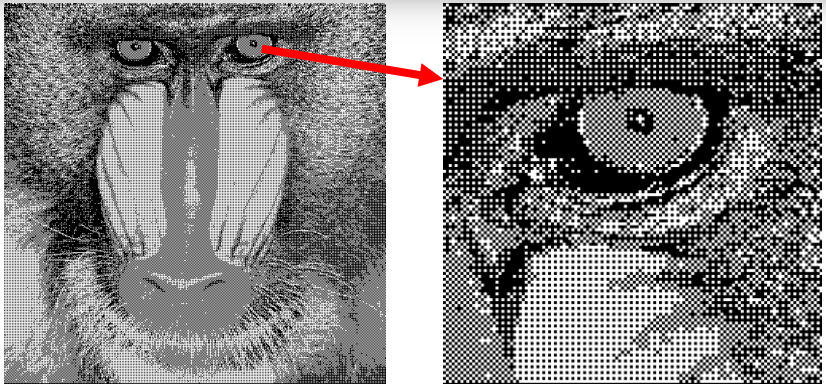


экран заполняется матрицами

```
int M_2x2[2][2] =
{
  {0, 2},
  {3, 1}
};

if (Img(x, y) * 5 / 256 > M_2x2[y % 2][x % 2])
  color = 1;
else
  color = 0;
```



оригинал



матрица 2x2

Метод Байера получения матриц смешивания

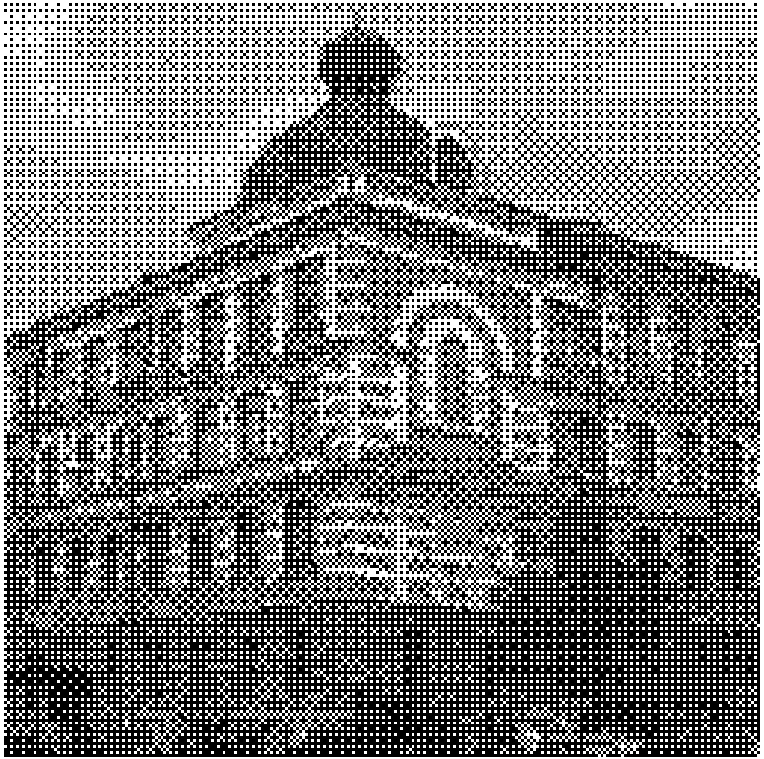
```
int M2[2][2] =
{
  {0, 2},
  {3, 1}
};
```

```
int M4[4][4] =
{
   { 0,  8,  2, 10},
   {12,  4, 14,  6},
   { 3, 11,  1,  9},
   {15,  7, 13,  5},
};
```

```
for (i = 0; i < 4; i++)
  for (j = 0; j < 4; j++)
  {
    M2n[0 + i][0 + j] = Mn[i][j] * 4 + 0;
    M2n[0 + i][4 + j] = Mn[i][j] * 4 + 2;
    M2n[4 + i][0 + j] = Mn[i][j] * 4 + 3;
    M2n[4 + i][4 + j] = Mn[i][j] * 4 + 1;
  }
```

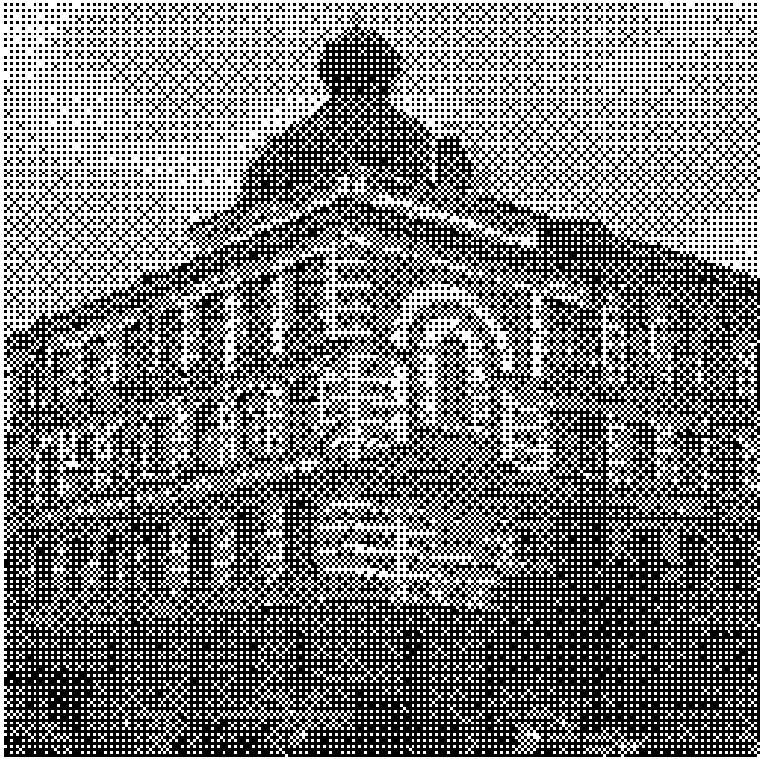
Примеры матриц Байера

2x2

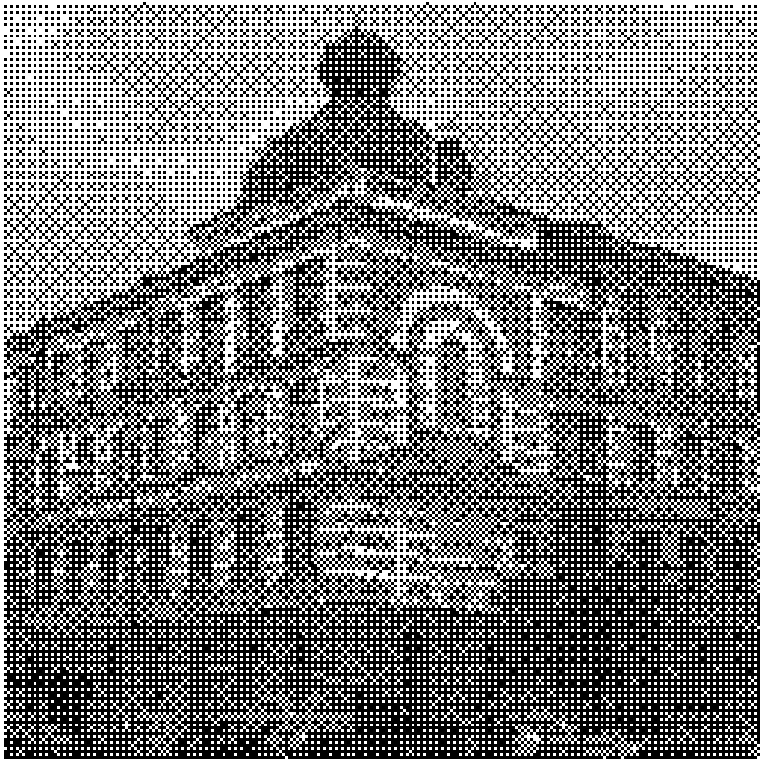


4x4

Примеры матриц Байера



8x8



16x16

Error Diffusion: алгоритм Флойда-Стейнберга

N=ближайший цвет I(x,y)

Pixel(x,y,N)

Error = значение цвета N – I(x,y)

Распространяем порции ошибки Error на еще не отрисованные точки

Переход к следующей точке

	*	7
3	5	1

1/16

Error Diffusion: примеры фильтров

*	3
3	2

1/8

False Floyd-Steinberg

		*	7	5
3	5	7	5	3
1	3	5	3	1

1/48

Jarvice, Judice, Ninke

		*	8	4
2	4	8	4	2
1	2	4	2	1

1/42

Stucki

Frankie Sierra

		*	5	3
2	4	5	4	2
	2	3	2	

1/32

		*	4	3
1	2	3	2	1

1/16

	*	2
1	1	

1/4

Подбор цвета: uniform palette

Универсальная палитра для любых изображений:

цвет задается по RGB каналам: ColorNo = B + SizeB * (G + SizeG * R)

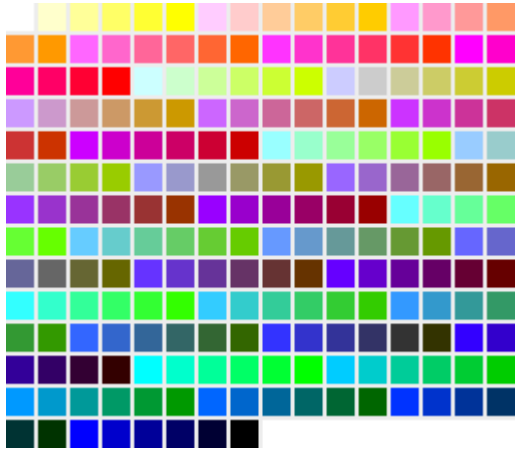
8

27

64

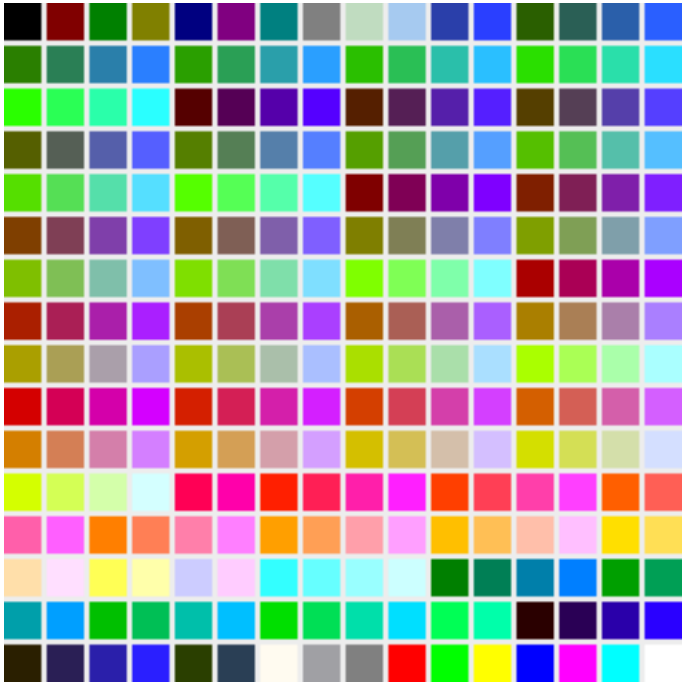
125

216

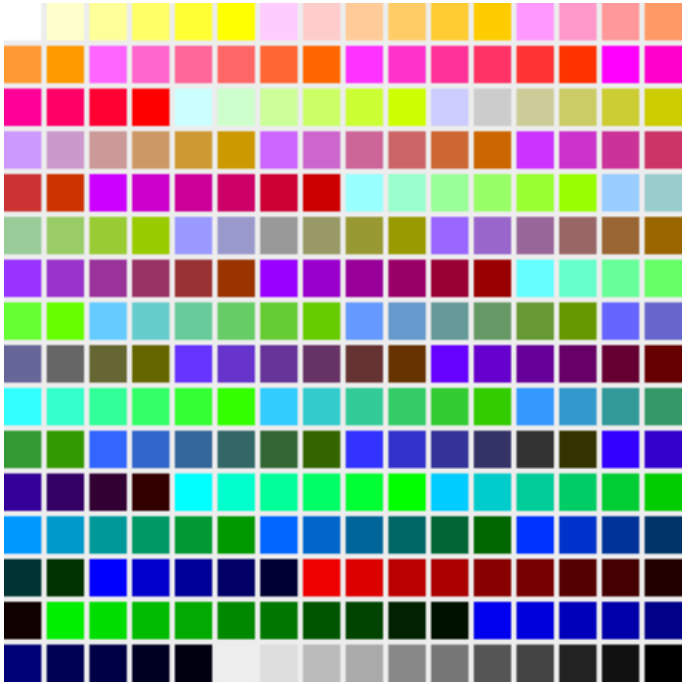


Подбор цвета: OS palette

Используются в индексированных графических режимах

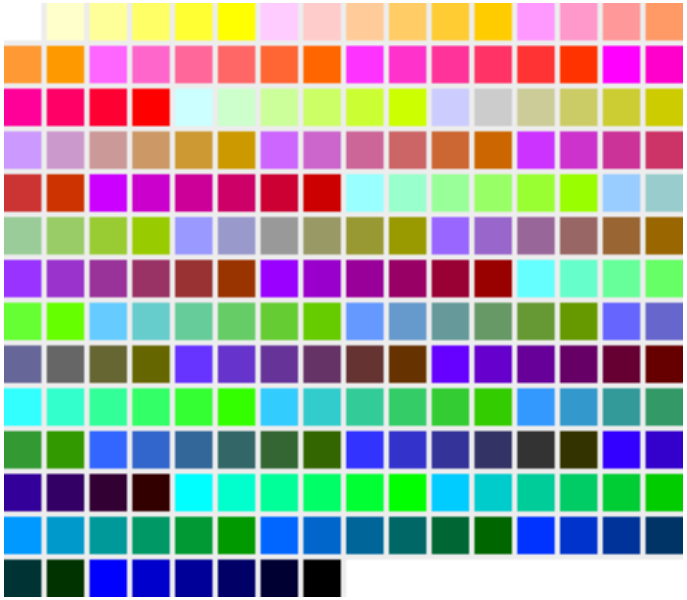


MS Windows



Mac OS

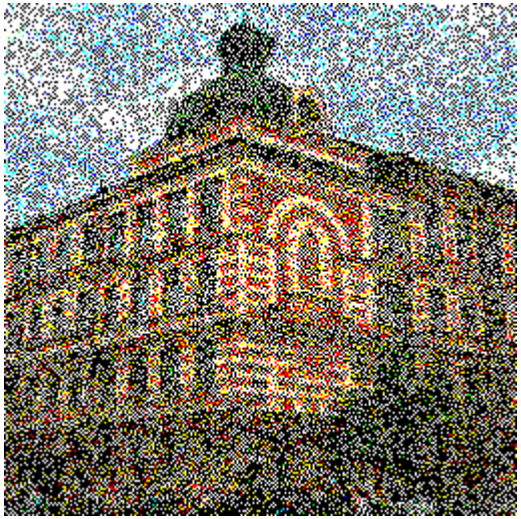
Подбор цвета: WEB & safe palette



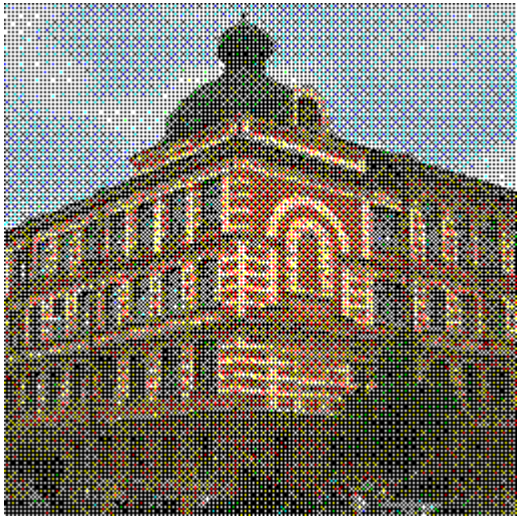
WEB палитра

оттенки по каналам
шаг: 0-51-102-163-204-255

Пример разных методов



random threshold

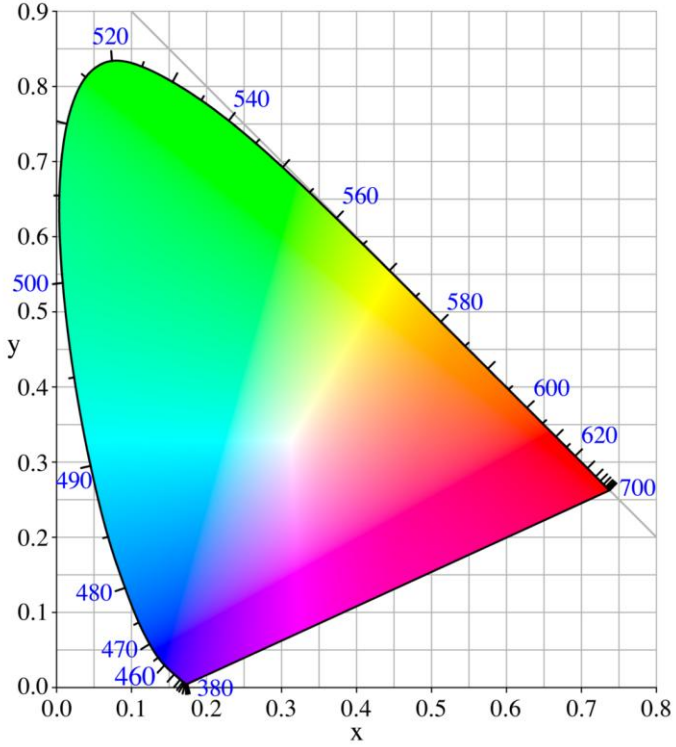


ordered dither

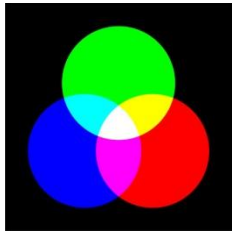
error diffusion

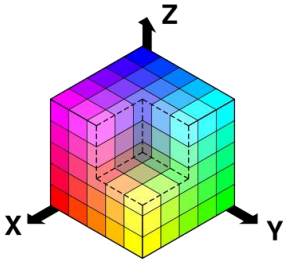
График МКО

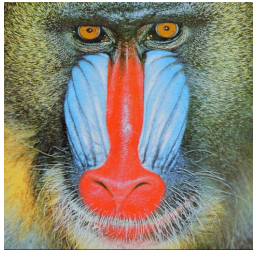
Международная Комиссия по Освещенности (Commission internationale de l'éclairage - CIE)

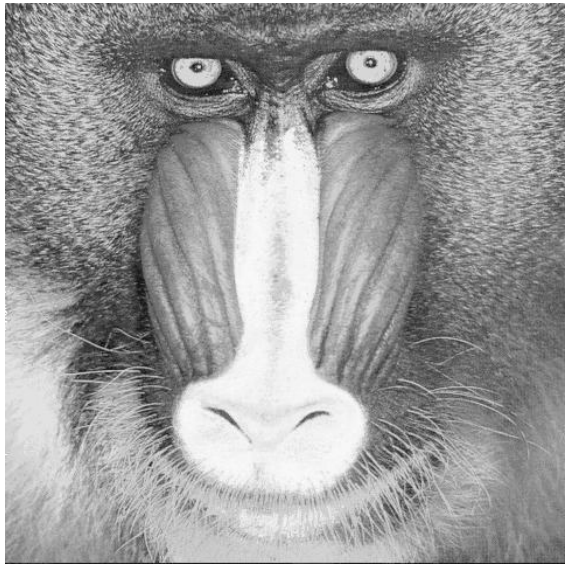


RGB

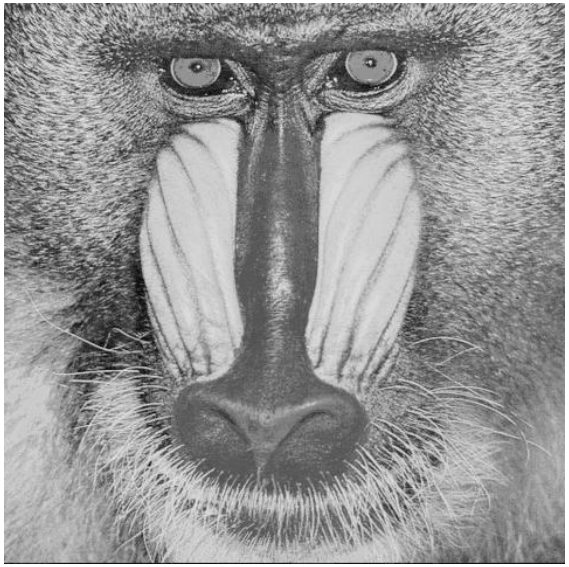




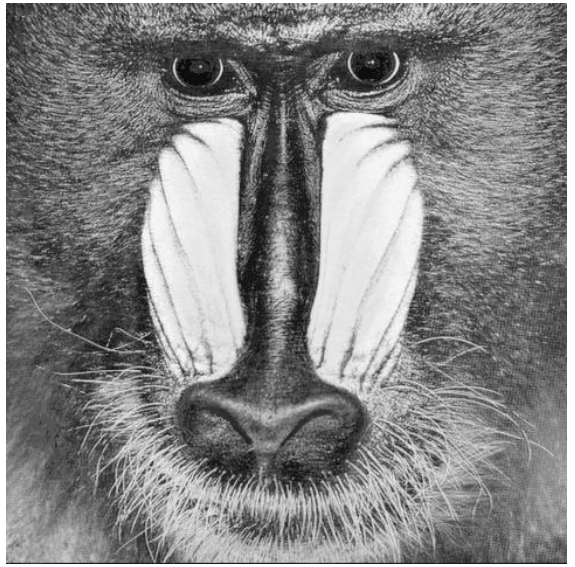




red

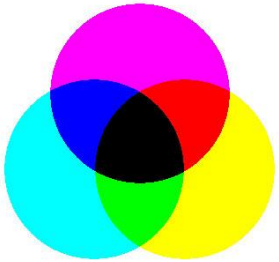


green

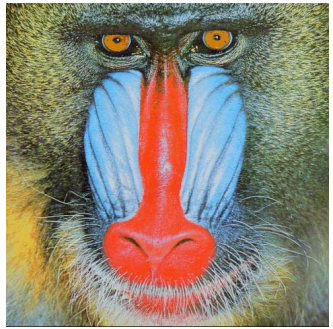


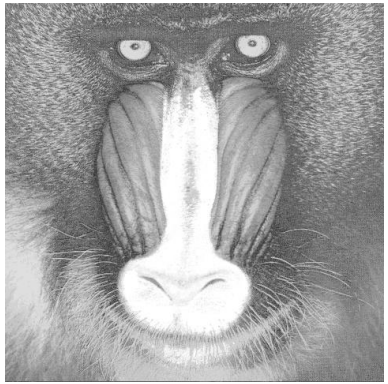
blue

CMYK

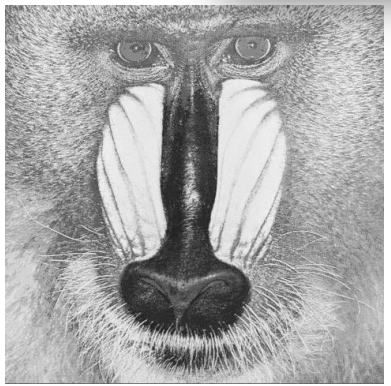


```
RGB2CMYK:
  C = 1.0 - R;
  M = 1.0 - G;
  Y = 1.0 - B;
  K = min(C, M, Y);
  C = C - K;
  M = M - K;
  Y = Y - K;

CMYK2RGB:
  R = 1.0 - min(1.0, C + K);
  G = 1.0 - min(1.0, M + K);
  B = 1.0 - min(1.0, Y + K);
```

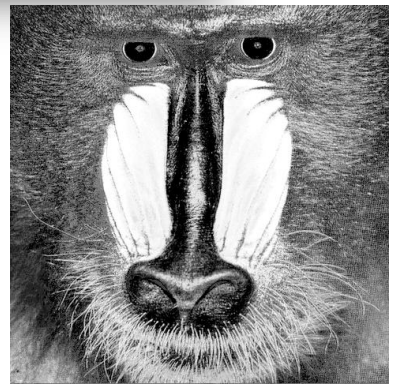




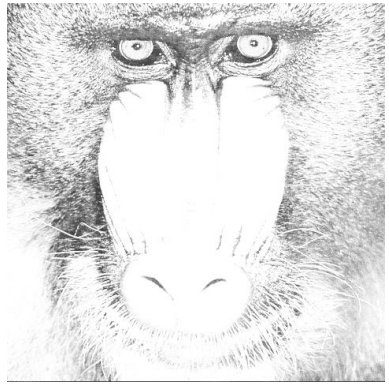
cyan



magenta

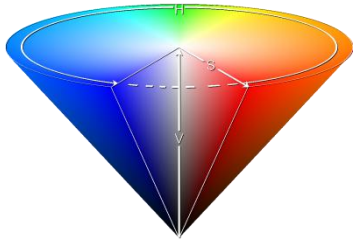


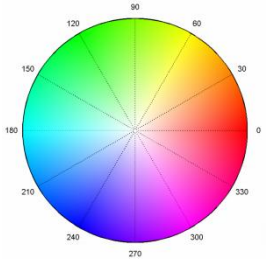
yellow

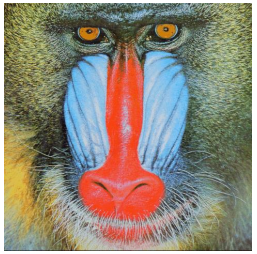


black

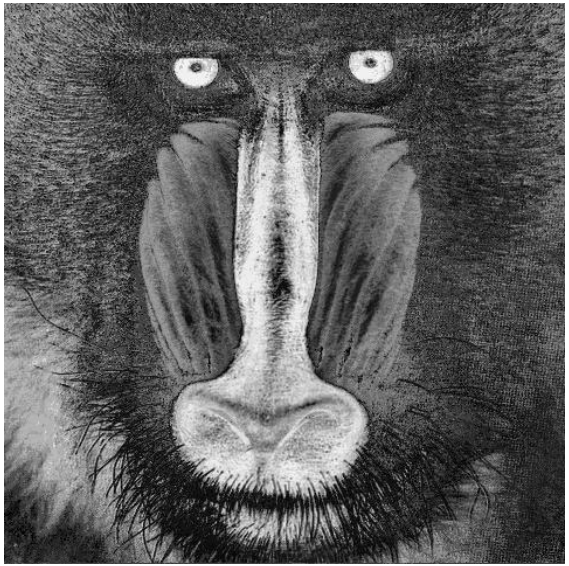
HSV



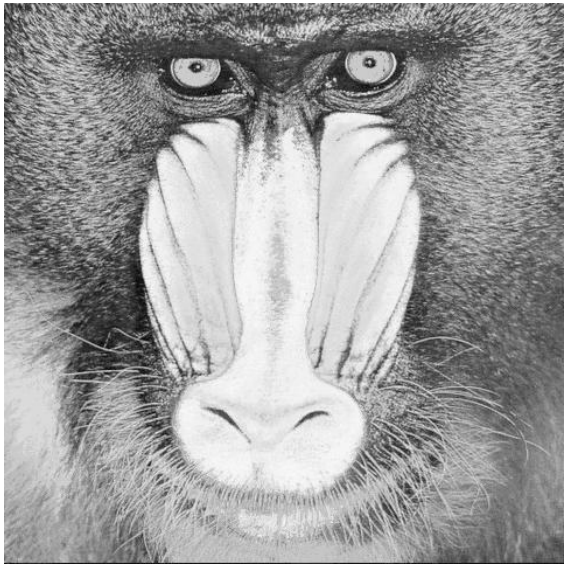




hue



saturation



value

HSV conversion

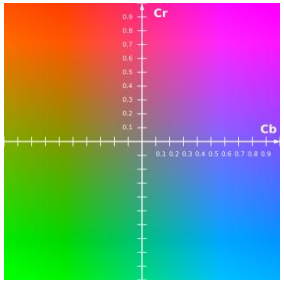
```
HSV2RGB:
  if (S == 0)
    return <V, V, V>;
  else
  {
    H = H / 60.0;
    n = (int)H;
    frac = H - n;
    c1 = V * (1.0 - S);
    c2 = V * (1.0 - S * frac);
    c3 = V * (1.0 - S * (1.0 - frac));
    if (n == 0)
      return <V, c3, c1>;
    if (n == 1)
      return <c2, V, c1>;
    if (n == 2)
      return <c1, V, c3>;
    if (n == 3)
      return <c1, c2, V>;
    if (n == 4)
      return <c3, c1, V>;
    if (n == 5)
      return <V, c1, c2>;
  }
```

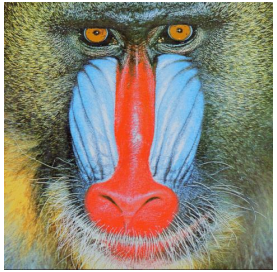
```
RGB2HSV
  maxc = max(R, G, B);
  minc = min(R, G, B);
  delta = maxc - minc;

  S = 0;
  if (maxc > 0)
    S = delta / maxc;
  V = maxc;
  if (S == 0)
    H = 0; /* неопределено */
  else
  {
    rc = (maxc - R) / delta;
    gc = (maxc - G) / delta;
    bc = (maxc - B) / delta;
    if (R == maxc)
      H = bc - gc;      /* Y-M */
    else if (G == maxc)
      H = 2 + rc - bc; /* C-Y */
    else
      H = 4 + gc - rc; /* M-C */
    H = H * 60.0;
  }
```

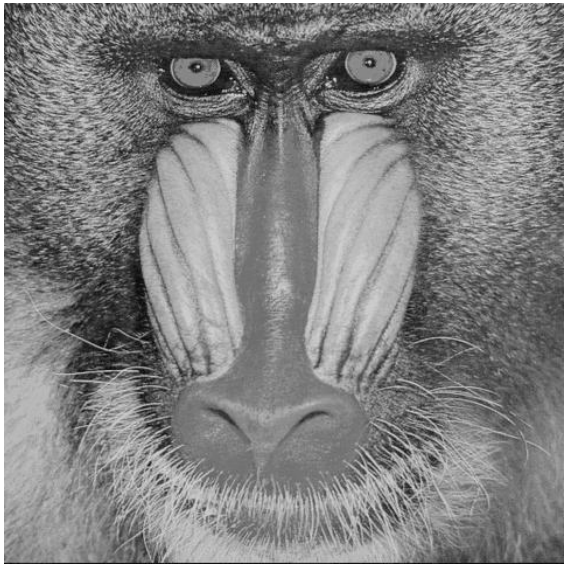
YCrCb

$$\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} = \begin{vmatrix} 0.299 & 0.587 & 0.114 \\ 0.5 & -0.4187 & -0.0813 \\ 0.1687 & -0.3313 & 0.5 \end{vmatrix} * \begin{vmatrix} R \\ G \\ B \end{vmatrix} + \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix}$$

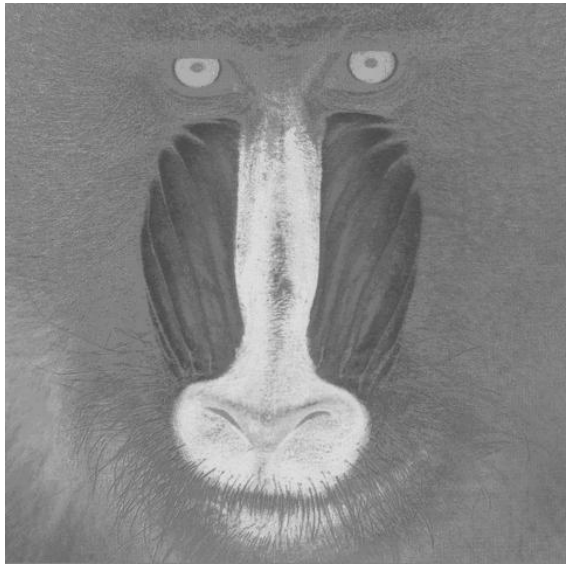
$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right)$$



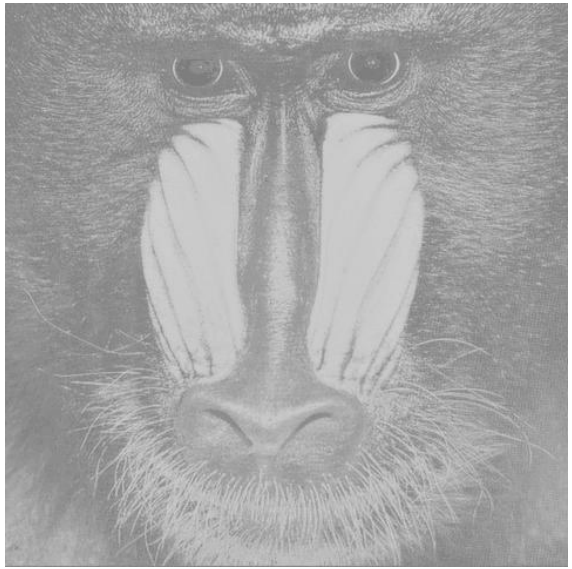




Y



Cr



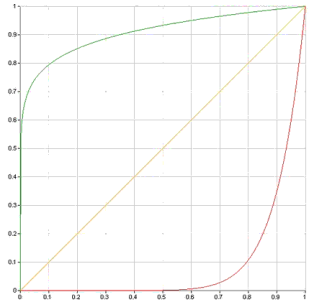
Cb

Коррекция цвета

Таблицы «подстановки» (Look Up Tables – LUT) `Color = LUT[Color];`

```
Примеры LUT, (квантование 256 уровней):
негатив:
        LUT[i] = 255 - i;
изменение яркости на Di:
        LUT[i] = Clamp(i + Di, 0, 255);
изменение контрастности изображения (диапазон A..B):
        LUT[i] = Clamp(255 * (i - A) / (B - A), 0, 255);
гамма коррекция:
    LUT[i] = 255 * pow(i / 255.0, 1 / gamma);
```

$$I_{вых} = I_{вх}^{\frac{1}{\gamma}}$$



Задание

- Реализовать полутонирование (*dither* и *error diffusion*) для монохромных изображений (результат выводить на экран или в файл)
- Реализовать программу, выполняющую коррекцию цвета (используя LUT) в полноцветном изображении путем изменения цветов в разных моделях (рассмотреть RGB и HSV). Результат демонстрировать на примере любого растрового изображения.

Ссылки

- P. Heckbert, "Color image quantization for frame buffer display,", *Computer Graphics*, 16(3), pp. 297-307 (1982).
- R. Ulichney, “Digital Halftoning,” *The MIT Press*, 1993.
- R. Floyd and L. Steinberg, “An adaptive algorithm for spatial gray scale,” *SID 1975 Symp. Dig. Tech. Papers*, pp. 36-37, 1975.
- B. E. Bayer, “An optimum method for two-level rendition of continuous-tone pictures,” *IEEE International Conference on Communications*, vol. 1, pp. 26-11 to 26-15, 1976.

Литература

Гонсалес Р., Вудс Р., "Цифровая обработка изображений", 3-е издание, исправленное и дополненное. — М.: Техносфера, 2012

Ганс Юрген Шлихт, "Цифровая обработка цветных изображений: Сканирование. Печать. Видео. Мультимедиа под Windows", М.: ЭКОМ, 1997

Литература (форматы изображений)

Дж. Мюррей, У. ван Райпер,"Энциклопедия форматов графических файлов", К.: Издательская группа BHV, 1997

Тим Кенцл, "Форматы файлов Internet", СПб: Питер, 1997.

Климов А.С. "Форматы графических файлов". К.: НИПФ "ДиаСофт Лтд.", 1995.