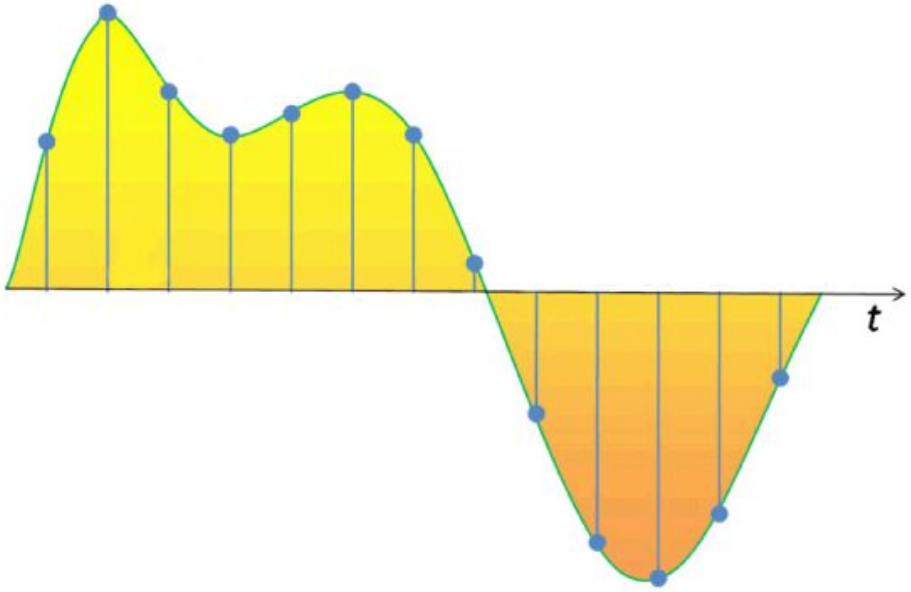
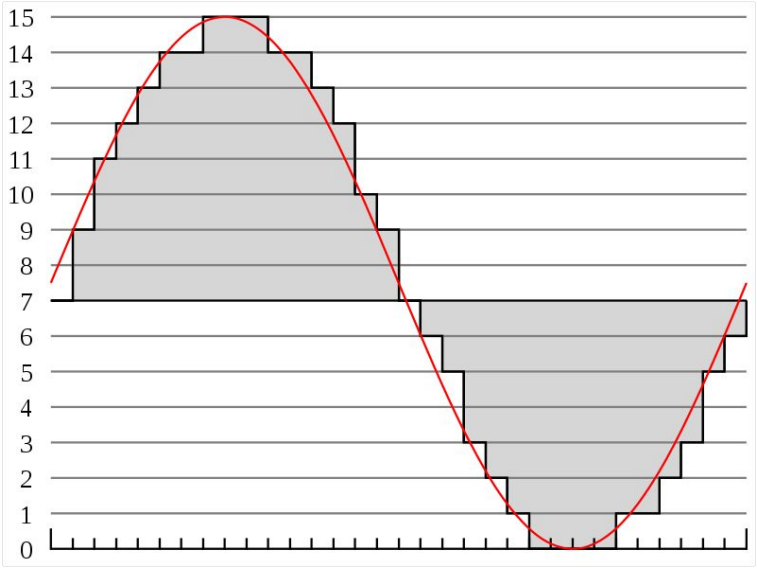
Цвет

Sampling & Quantization

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





Sampling & Quantization

При оцифровке сигнала $x(t)$ производятся две операции - дискретизация и квантование. Дискретизация -- это замена сигнала x(t) с непрерывным временем t на дискретизованный сигнал -- последовательность чисел $x(t_i)$ для дискретного набора моментов времени $t_1, t_2, ..., t_i, ...$(чаще всего интервалы между моментами времени $\Delta t = t_i - t_{i-1}$ берутся одинаковыми). При дискретизации, конечно, часть информации о сигнале теряется. Но если сигнал $x(t)$ за время Δt не сильно изменяется, числа $x(t_i)$ и $x(t_{i-1})$ близки друг к другу, то поведение $x(t)$ между временами t_i и t_{i-1} нетрудно восстановить (сигнал практически линейно изменяется во времени от $x(t_{i-1})$ до $x(t_i)$). При дискретизации мы теряем частотные составляющие сигнала с частотами порядка $f > 1/\Delta t$ и выше.

При дискретизации время из аналогового как бы становится цифровым -- моменты времени t_i можно нумеровать, кодировать. Производится замена непрерывного времени t на нечто, которое может принимать не все значения, а только некоторые, а именно $t_1, t_2, ..., t_i, ...$ Квантование сигнала -- это нечто похожее, только данная процедура производится не со временем, а со значением сигнала x. Выбирается некий набор возможных значение сигнала $x_1, x_2, ..., x_n, ...$ и каждому $x(t_i)$ сопоставляется ближайшее число из этого набора.

Приведем конкретный пример дискретизации и квантования:

Пусть сигнал $x(t)$ такой, что $x(t) = \sqrt{t}$, шаг дискретизации $\Delta t = 0.1$ (т.е. набор моментов времени $t = 0, 0.1, 0.2, \ldots$), значение сигнала x мы будем записывать с точностью до одной сотой (т.е. набор значений сигнала $x = 0, \pm 0.01, \pm 0.02, \ldots$). После дискретизации сигнала получим

x = 0. 0.3162... 0.4472... 0.5477... 0.6324... ...
t = 0 0.1 0.2 0.3 0.4 ...

Учитывая точность хранения значений x, после квантования получаем

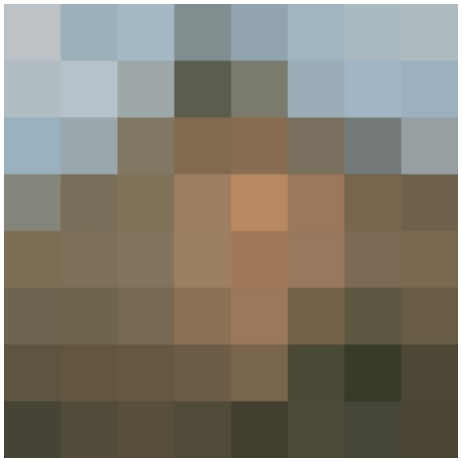
x = 0. 0.32... 0.45... 0.55... 0.63... ...
t = 0 0.1 0.2 0.3 0.4 ...

При дискретизации мы теряем высокие ($f > 1/\Delta t$) частоты сигнала, при квантовании мы теряем маленькие (меньше $\Delta x = x_n - x_{n-1}$) изменения сигнала. Кроме того, получившийся после квантования сигнал $x_n(t_i)$ отличается от реального (но уже дискретизованного) сигнала $x(t_i)$ на величину порядка шага квантования (или кванта) Δx. Это различие носит название шума квантования, и оно принципиально неустранимо.

Для примера, описанного выше, имеем

$x(t_i)$ = 0. 0.3162... 0.4472... 0.5477... 0.6324... ...
$x_n(t_i)$ = 0. 0.32... 0.45... 0.55... 0.63... ...
t_i = 0 0.1 0.2 0.3 0.4 ...
шум квантования $\approx$ 0. 0.00377 0.00279 0.00228 −0.00246 ...

Sampling (разрешение)



8x8

16x16

32x32

64x64

128x128

256x256

Quantization (глубина цвета)

2 цвета

3 цвета

4 цвета

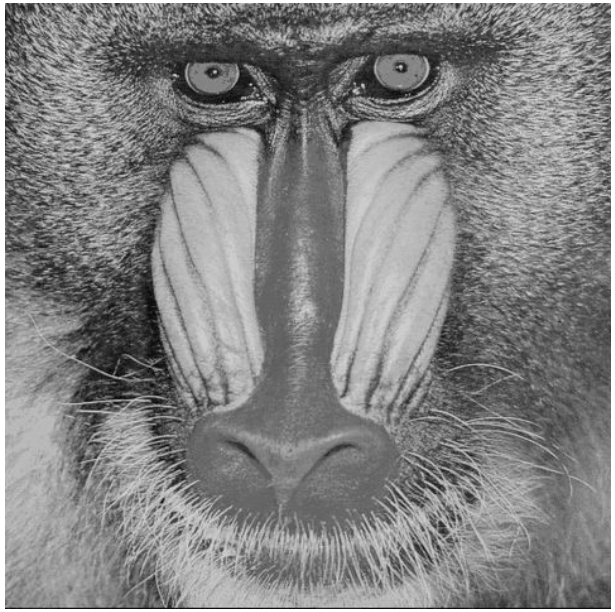
8 цветов

16 цветов

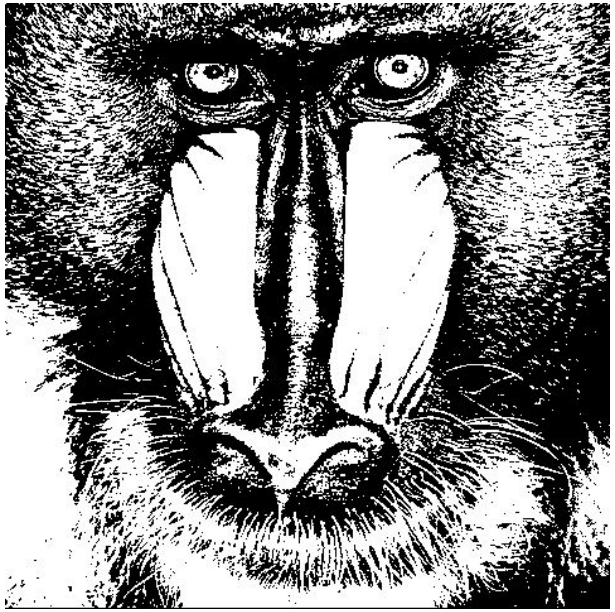
256 цветов

Fixed Thresholding

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



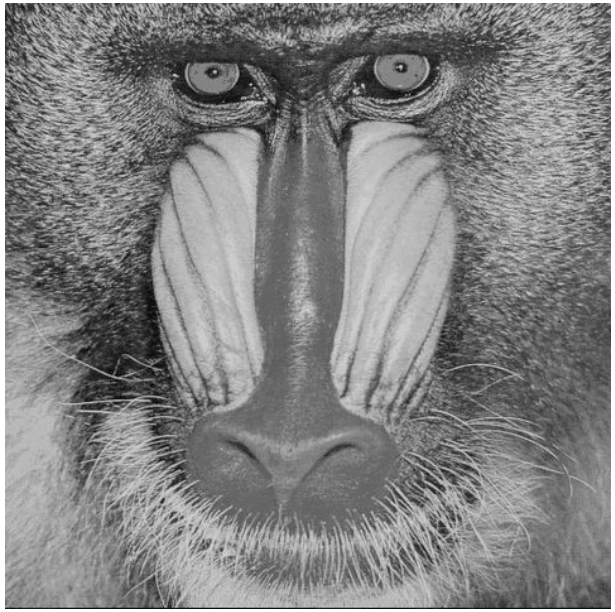
оригинал



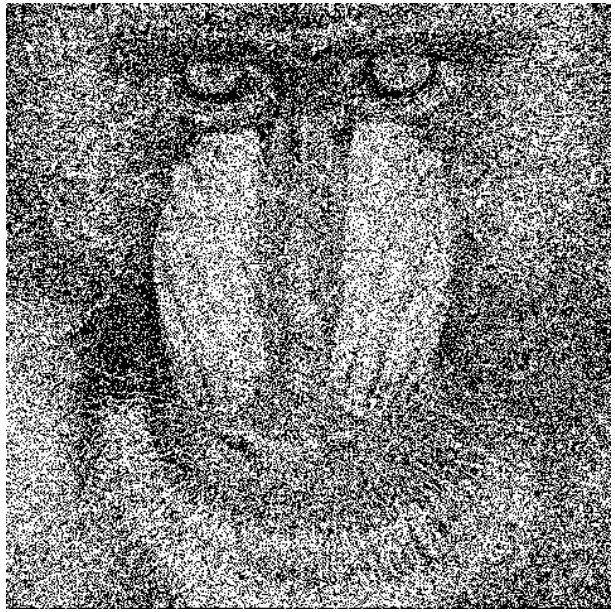
порог = 128

Random Thresholding

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

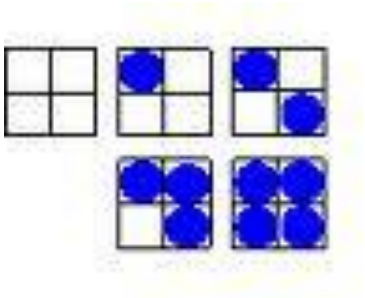


оригинал



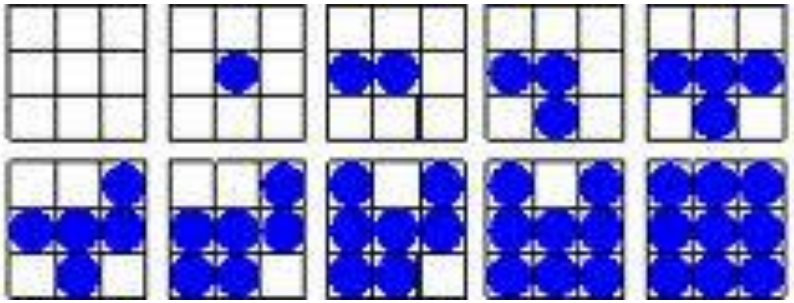
«случайный» порог

Patterning



5 уровней

(2x2)

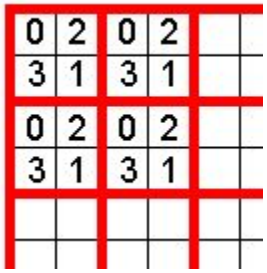


10 уровней

(3x3)

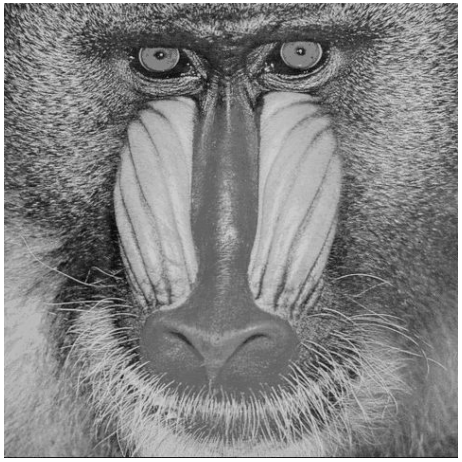
Ordered Dither

0 2
3 1

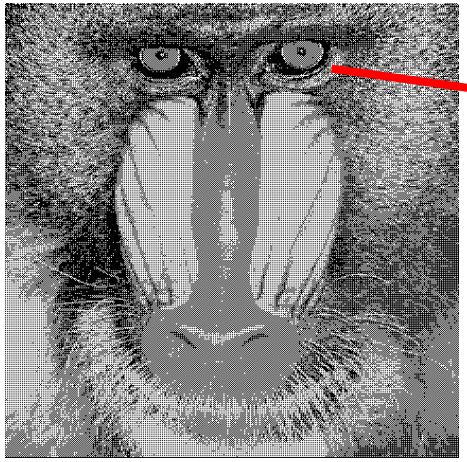


экран заполняется матрицами

```
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

увеличенный фрагмент

Метод Байера получения матриц смешивания

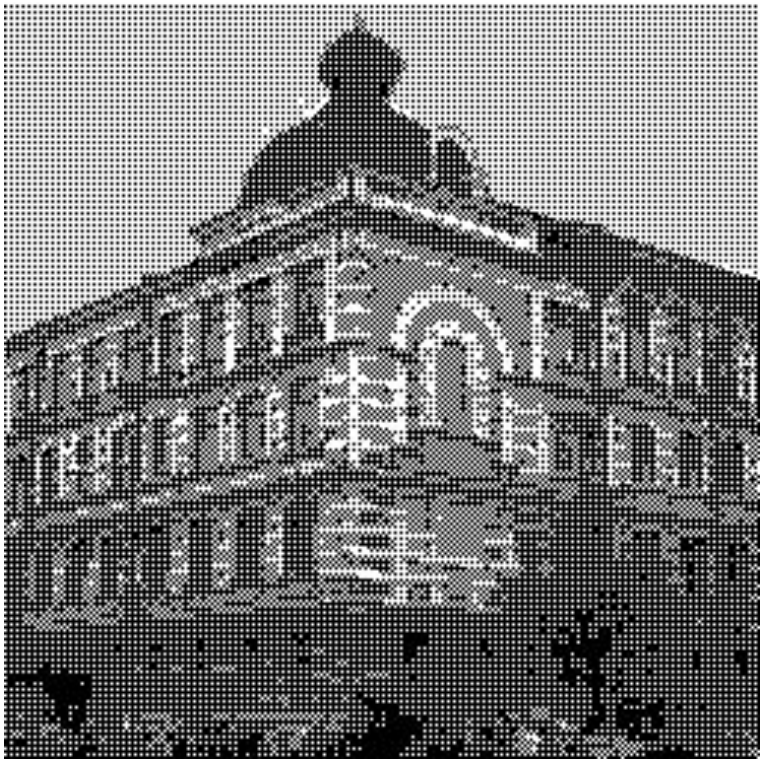
```
int M2[2][2] =
{
  {0, 2},
  {3, 1}
};
```

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

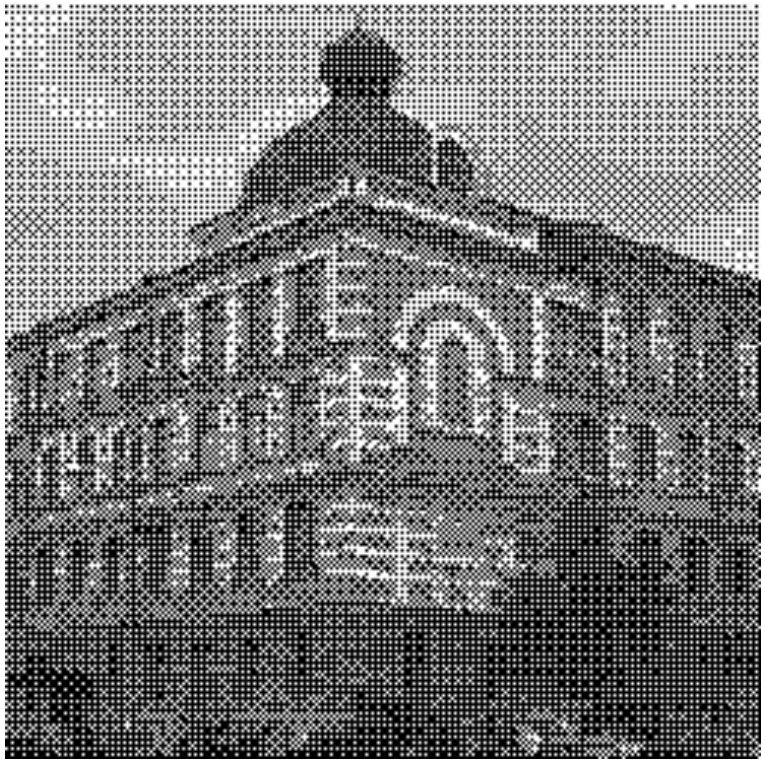
```
-----------------------------------
for (i = 0; i < 4; i++)
  for (j = 0; j < 4; j++)
  {
    M2n[0 + i][0 + j] = Mn[i][j] * 4 + 0;
    M2n[0 + i][4 + j] = Mn[i][j] * 4 + 2;
    M2n[4 + i][0 + j] = Mn[i][j] * 4 + 3;
    M2n[4 + i][4 + j] = Mn[i][j] * 4 + 1;
  }
```

Метод формирования полутоновых изображений путем пороговой обработки ис
ходного изображения. Выбор порогов осуществляется с помощью матрицы со с
лучайным набором порогов. Ее использование
позволяет создать эффект “черепичной крыши”, что улучшает визуальное воспр
оизведение двухуровневых изображений.

Примеры матриц Байера

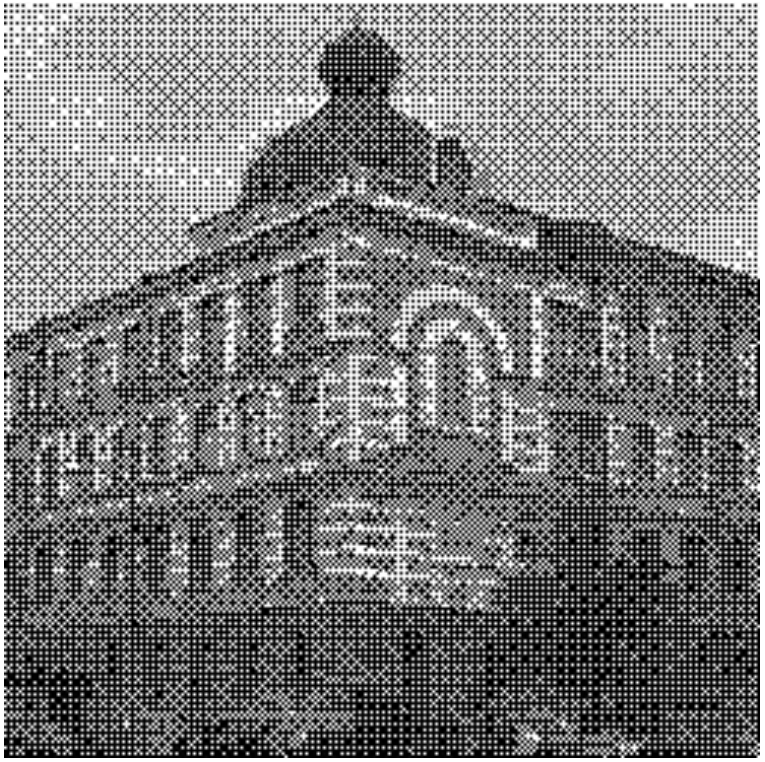


2x2

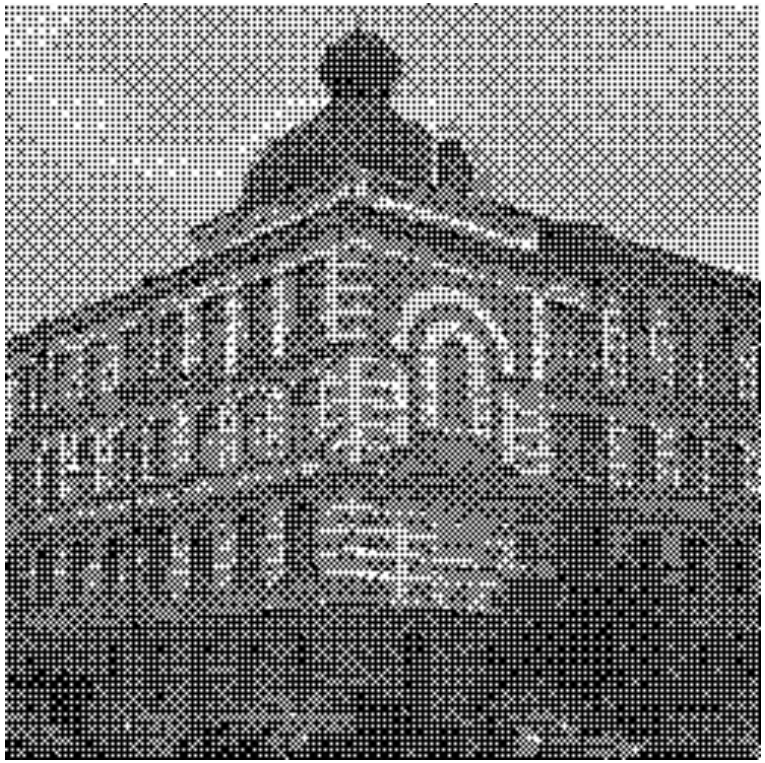


4x4

Примеры матриц Байера

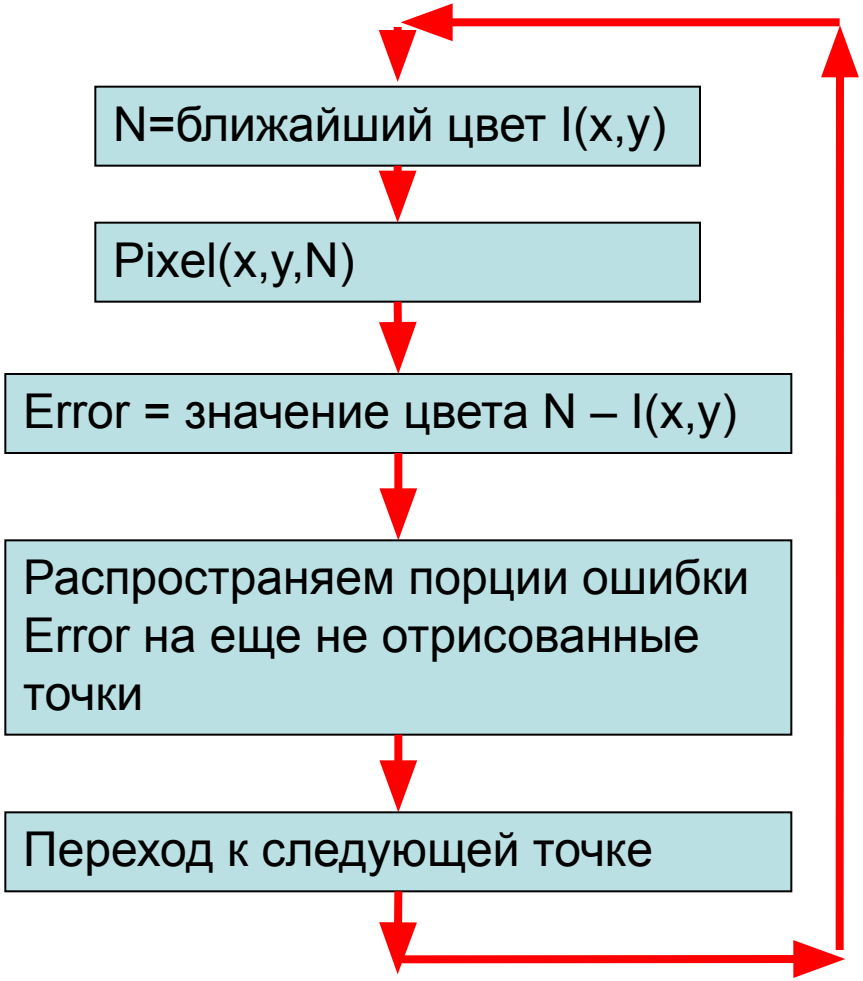


8x8



16x16

Error Diffusion: алгоритм Флойда-Стейнберга



	*	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

		*	8	4
2	4	8	4	2

1/32

Burkes

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:

$$ColorNo = B + SizeB * (G + SizeG * R)$$

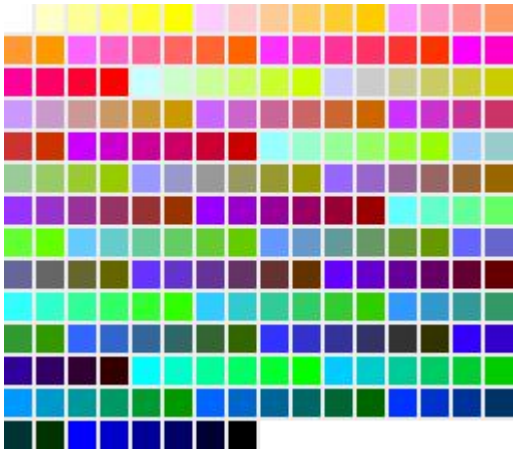
8

27

64

125

216



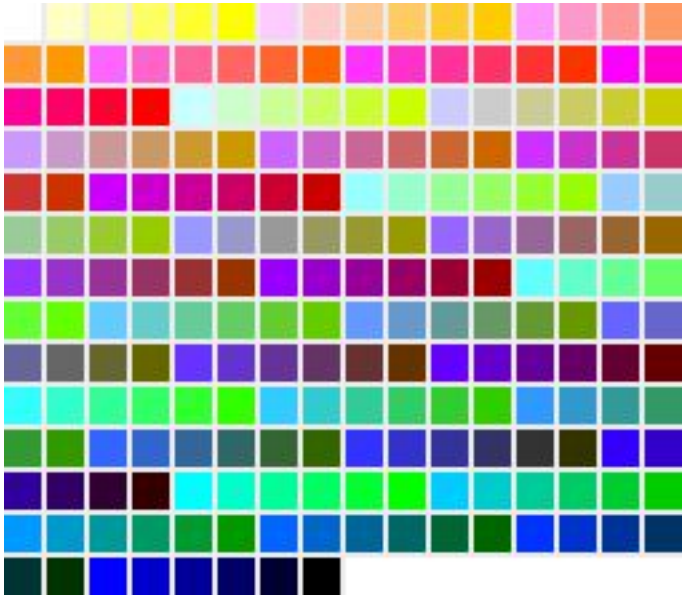
Подбор цвета: OS palette

Используются в индексированных графических режимах

MS Windows

Mac OS

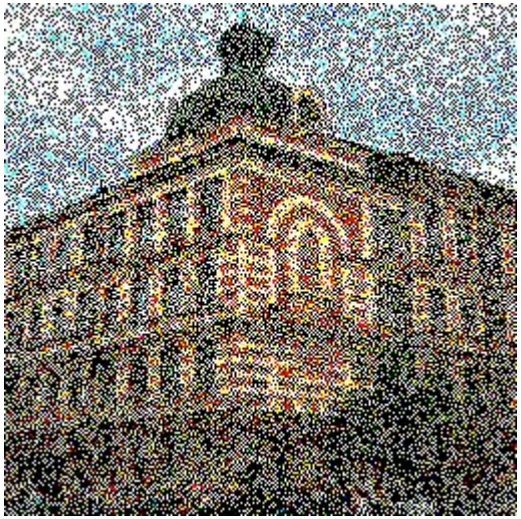
Подбор цвета: WEB & safe palette



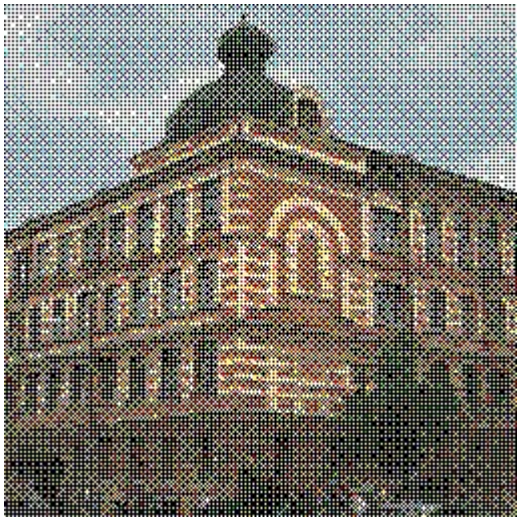
WEB палитра

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

Пример разных методов



random threshold

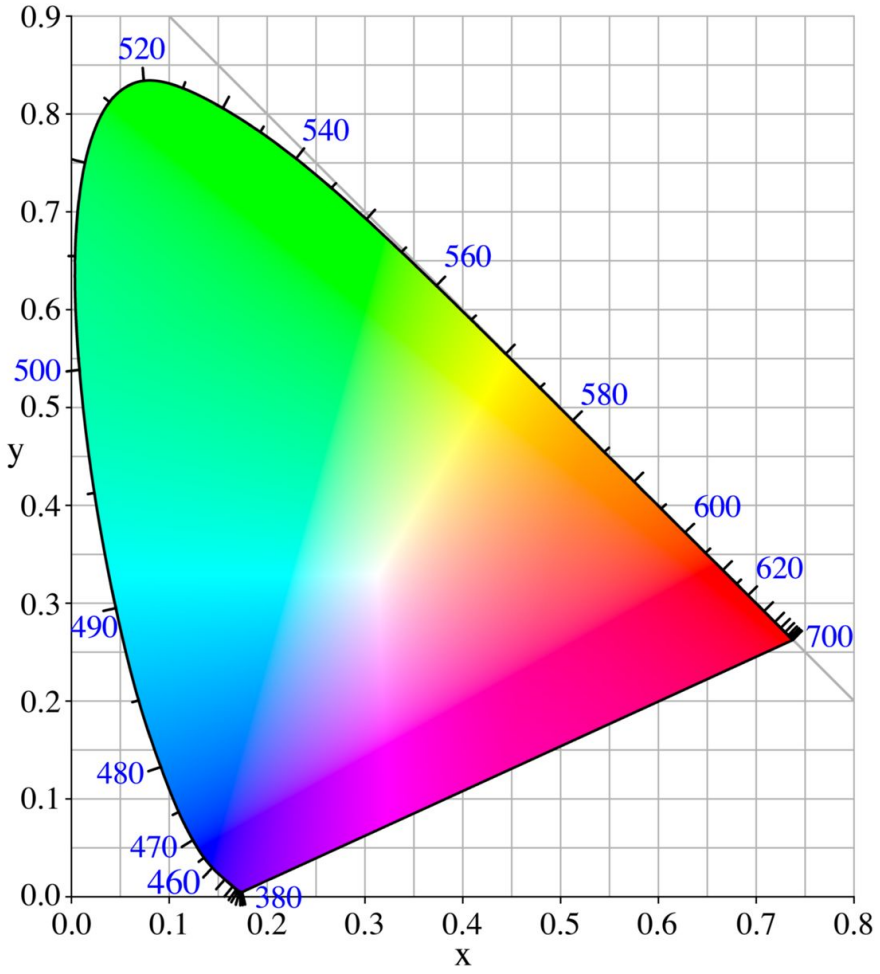


ordered dither

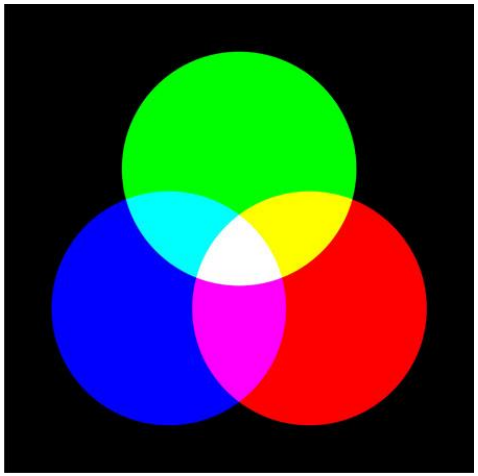
error diffusion

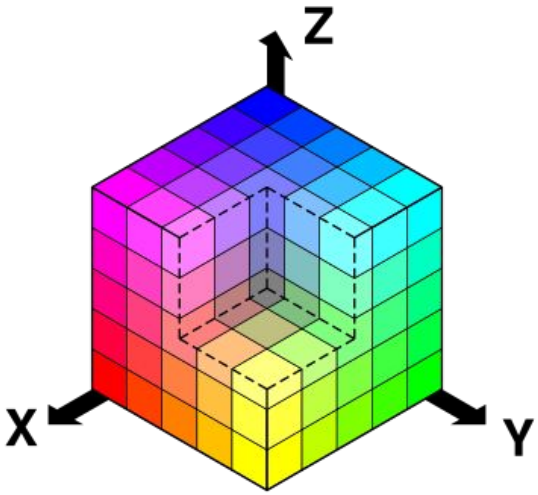
График МКО

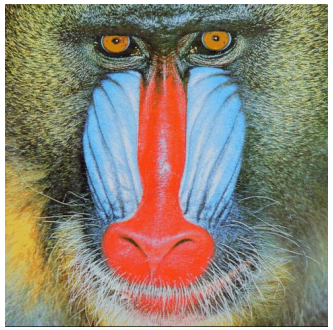
Международная Комиссия по Освещенности (Commission internationale de l'éclairage - CIE)

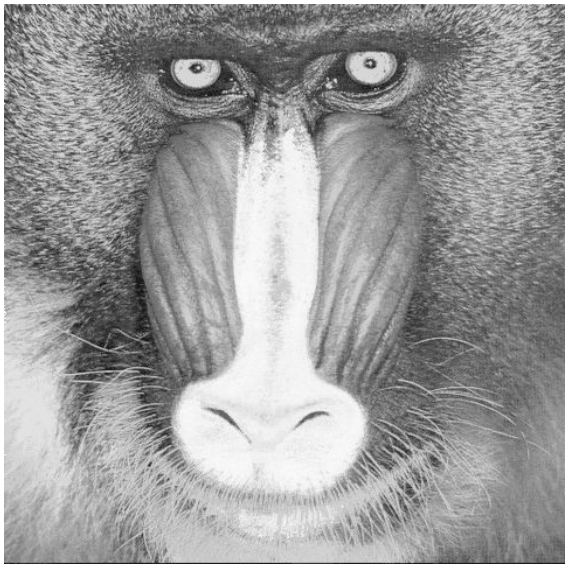


RGB

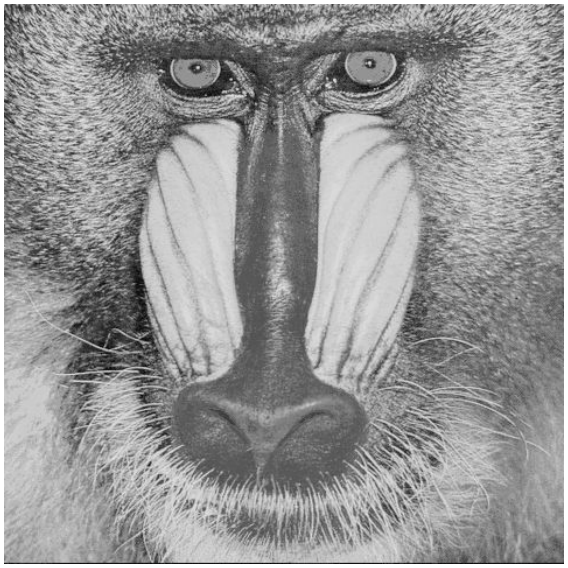




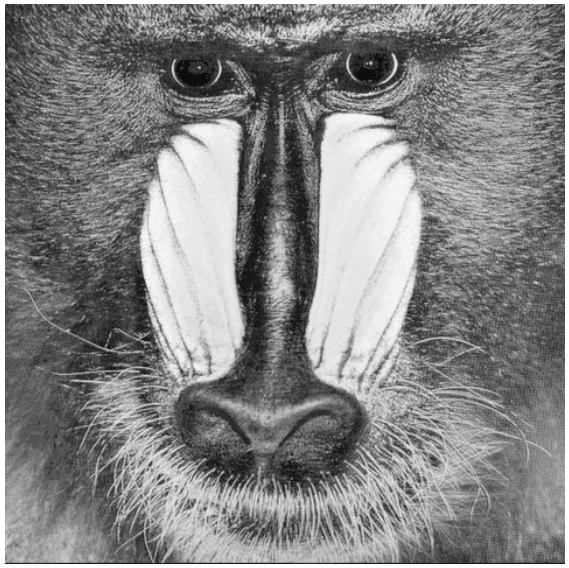




red

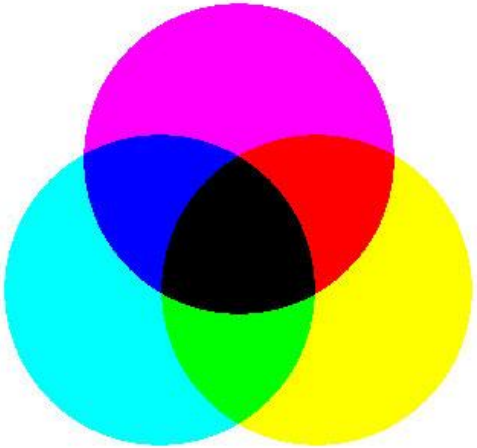


green

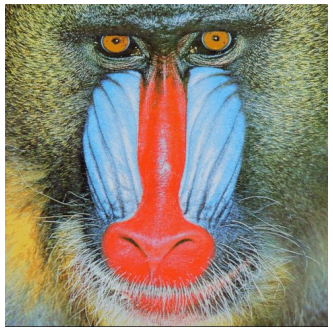


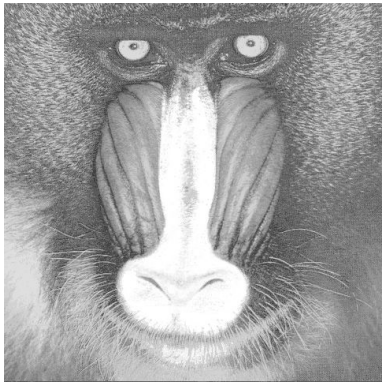
blue

CMYK

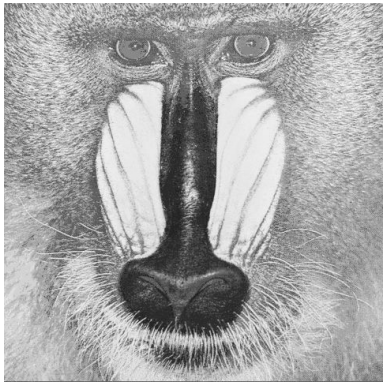


```
RGB2CMYK:
  C = 1.0 - R;
  M = 1.0 - G;
  Y = 1.0 - B;
  K = min(C, M, Y);
  C = C - K;
  M = M - K;
  Y = Y - K;

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);
```

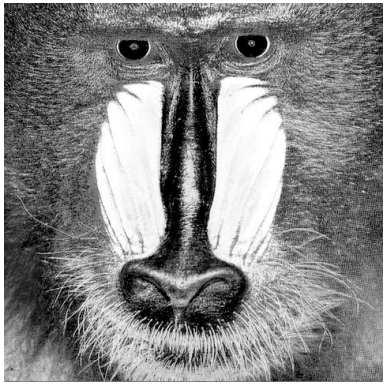




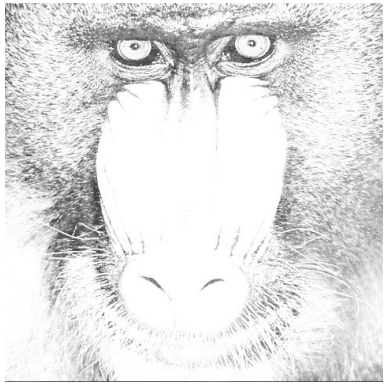
cyan



magenta

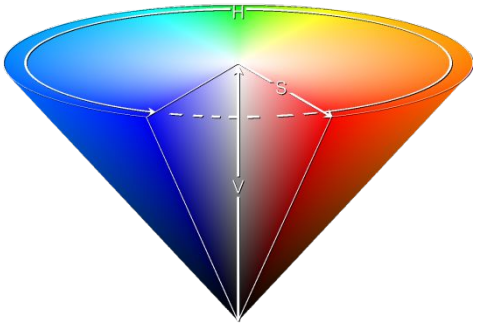


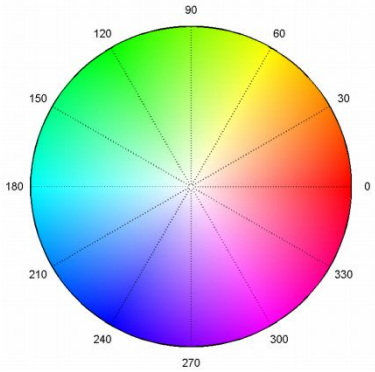
yellow

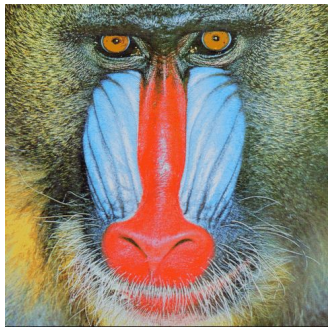


black

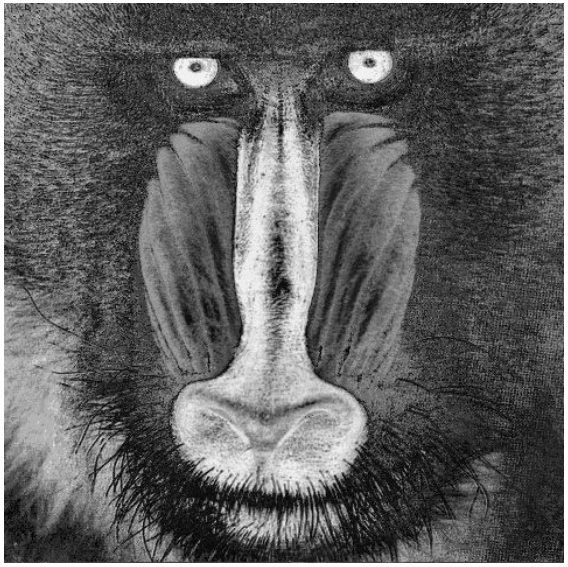
HSV (тон, насыщенность, яркость)



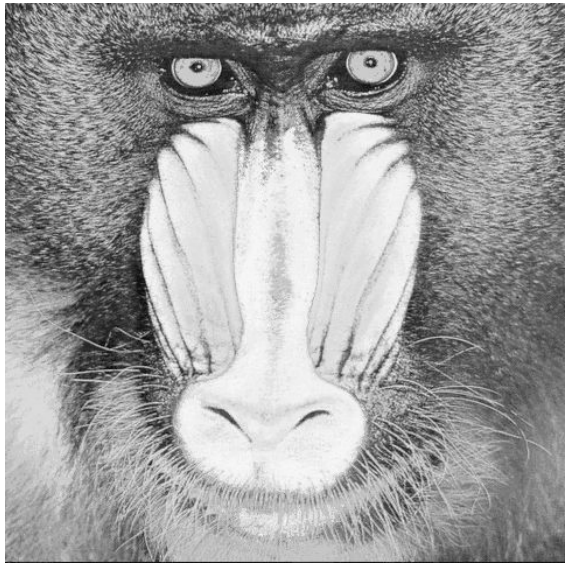




hue



saturation



value

HSV conversion

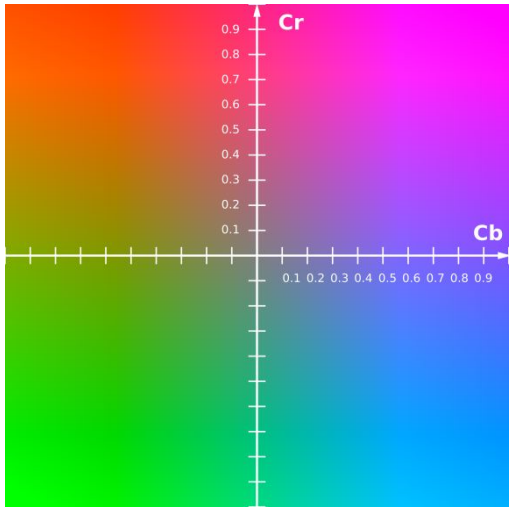
```
HSV2RGB:
  if (S == 0)
    return <V, V, V>;
  else
  {
    H = H / 60.0;
    n = (int)H;
    frac = H - n;
    c1 = V * (1.0 - S);
    c2 = V * (1.0 - S * frac);
    c3 = V * (1.0 - S * (1.0 - frac));
    if (n == 0)
      return <V, c3, c1>;
    if (n == 1)
      return <c2, V, c1>;
    if (n ==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

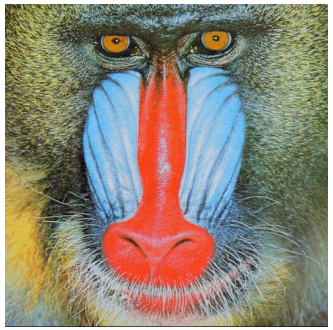
```
RGB2HSV
  maxc = max(R, G, B);
  minc = min(R, G, B);
  delta = maxc - minc;

  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;
  }
```

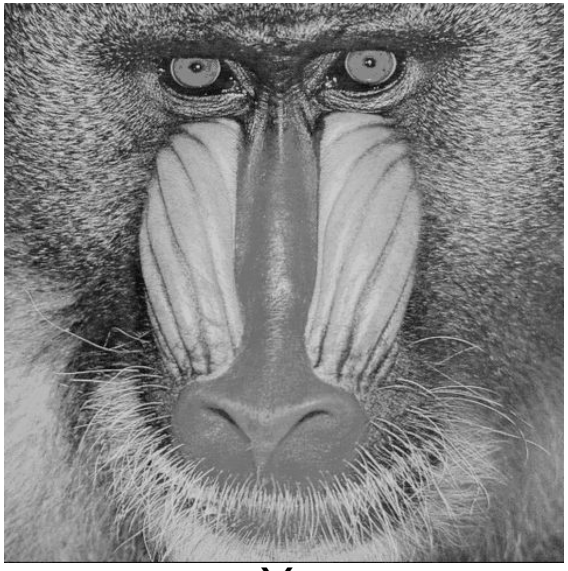
YCrCb

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

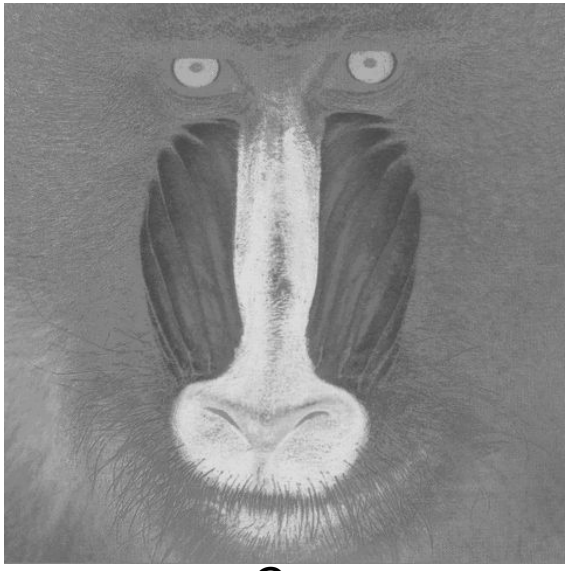
$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{vmatrix} 1 & 0 & 1.402 \\ 1 & -0.34414 & -0.71414 \\ 1 & 1.772 & 0 \end{vmatrix} * \left(\begin{vmatrix} Y \\ Cb \\ Cr \end{vmatrix} - \begin{vmatrix} 0 \\ 128 \\ 128 \end{vmatrix} \right)$$



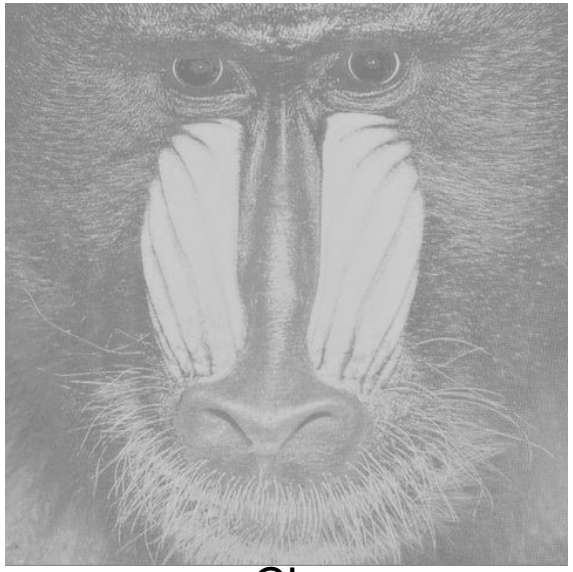




Y

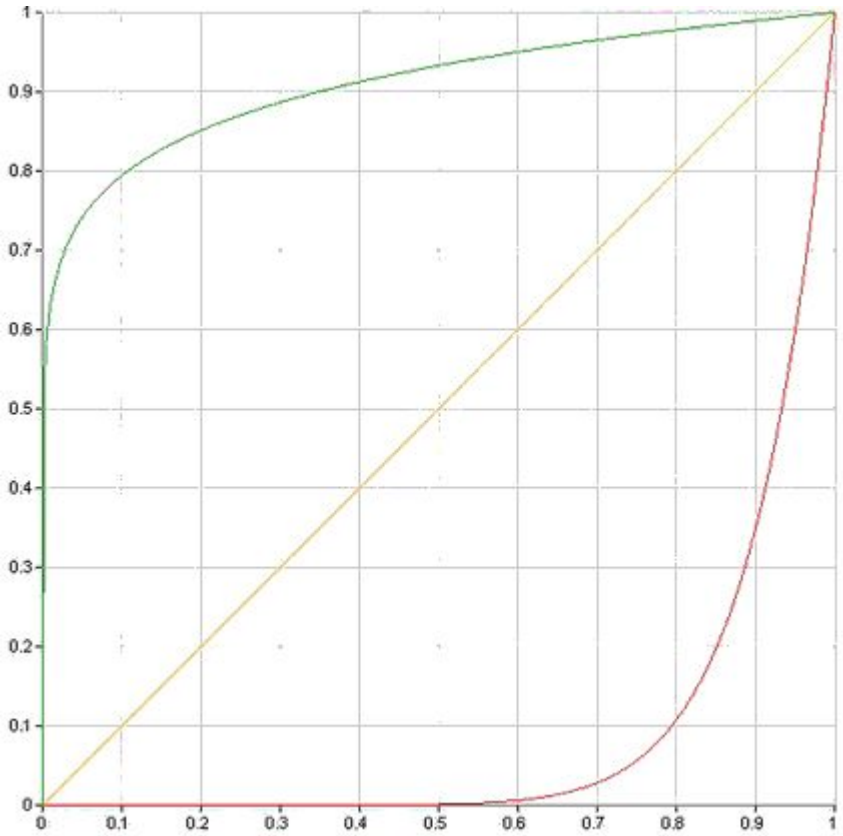


Cr



Cb

Коррекция цвета

- LUT: `Color = LUT[Color];`
- Гамма коррекция, контрастность, яркость

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