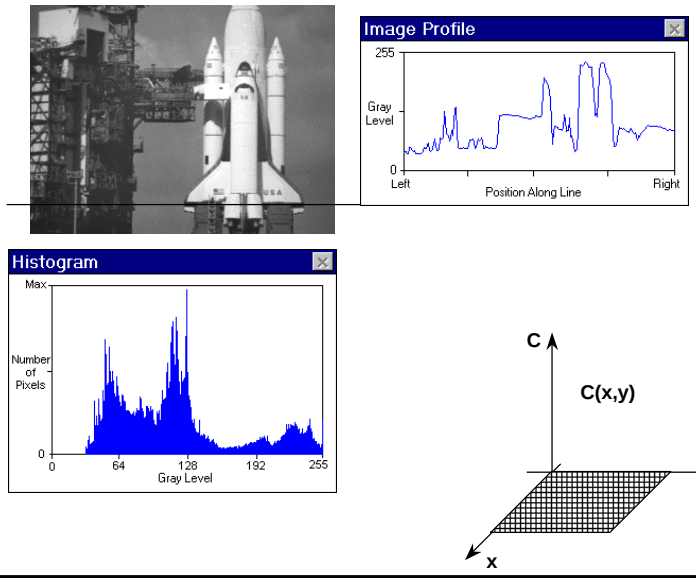
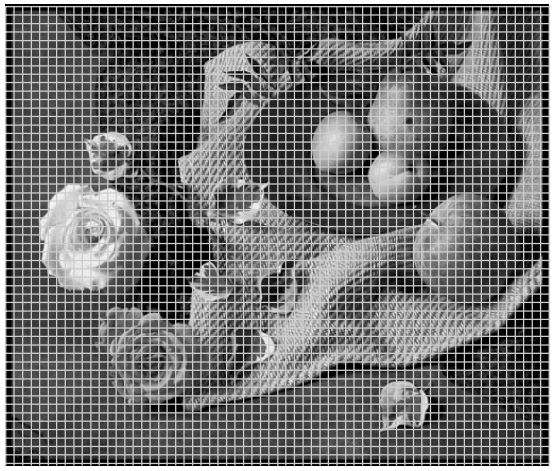
Imagens

Características Básicas de Imagens



Digitalização de Imagens

Discretização espacial (amostragem)



Digitalização de Imagens

Imagem de tons (ou cores) contínuas

amostragem →

64x54

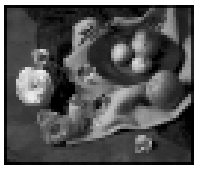


Imagem amostrada

quantização ↓

64x54 - 16 cores

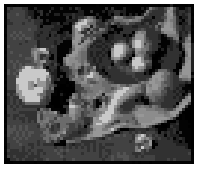


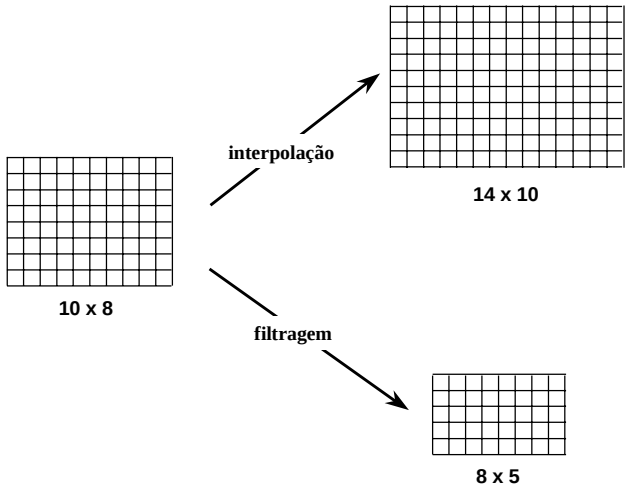
Imagem amostrada e quantizada

← *codificação*

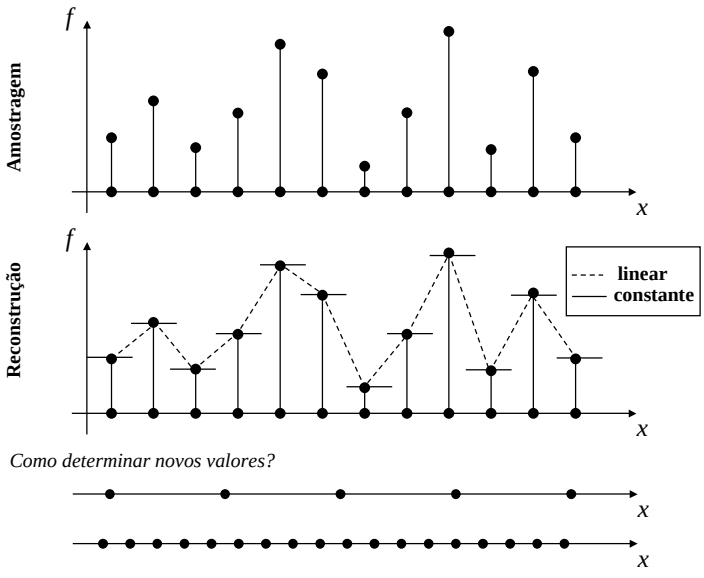
55	55	55	55	55	55	55
55	20	22	23	45	55	55
55	55	10	09	11	55	55
55	55	43	42	70	55	55
55	55	28	76	22	55	55
55	55	55	55	55	55	55

Resolução Espacial

Escala como uma re-amostragem de uma função

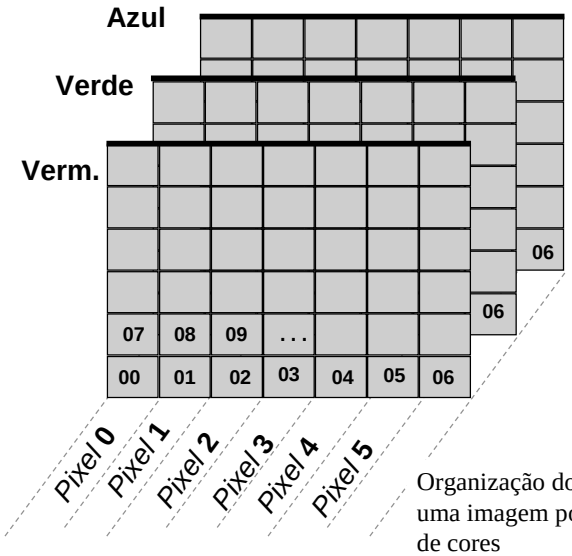


Analogia ao Sinal 1D



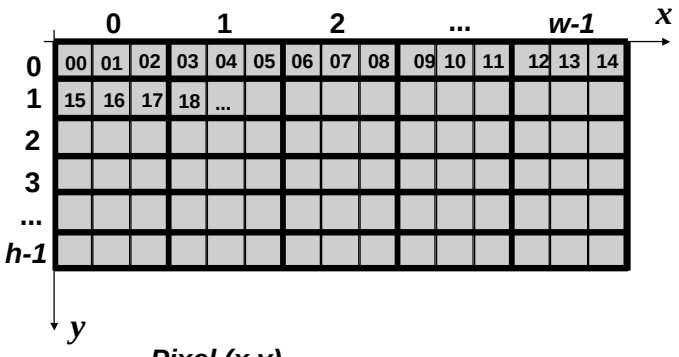
Armazenamento de Imagens

Plano de Cores



Organização dos *pixels* de uma imagem por planos de cores

Organização de *pixels* num *array* no formato PPM (o mais simples)



Pixel (x,y)

```
offset=3*(w*y+x);
red   = rgb_vector[offset];
green = rgb_vector[offset+1];
blue  = rgb_vector[offset+2];
```

Formato PPM

- **File_signature** "P6".
- **White_space** (blanks, TABs, CRs, LFs).
- **Width**, *w*, (ASCII decimal characters).
- **White_space** (blanks, TABs, CRs, LFs).
- **Height**, *h*, (ASCII decimal characters).
- **White_space** (blanks, TABs, CRs, LFs).
- **Max_color**, *max*, (ASCII decimal characters).
- **White_space** (blanks, TABs, CRs, LFs).
- **Pixels,** (*3*w*h* bytes rgb components of pixels)

- Comments from # to the end of line
- lines ≤ 70 characters

Formato PPM

exemplo

```
P6
# Created by Paint Shop Pro
358 539
255
=?:?A<AC>CE@EFAFGBGHCGHCGHB . . .
```

Gravação em PPM

```
int ppm_write(int w, int h, unsigned char *rgb,
                                    char *file_name)
{
  FILE *fp;

  fp = fopen(file_name, "wb");
  if (fp == NULL)
    return 0;

  if (fprintf(fp, "P6\n%d %d\n255\n", w, h) <= 0)
  {
    fclose(fp);
    return 0;
  }

  if (fwrite(rgb, 3*w*h, 1, fp) != 1)
  {
    fclose(fp);
    return 0;
  }

  fclose(fp);
  return 1;
}
```

Leitura em PPM

```
int ppm_read(int *p_w, int *p_h, unsigned char **p_rgb,
                                    char *file_name)
{
  FILE *fp;   char line[80];   int rgb_size;   int max;

  fp = fopen(file_name, "rb");
  if (fp == NULL) {
      printf(”Error reading %s",file_name); return 0;}

  fgets(line,80,fp);
  if(strcmp(line,"P6\n")) {
    printf(”Wrong signature\n"); return 0; }

  while (fscanf( fp, " %d ", p_w ) != 1)
          fgets(line, 80, fp);

  while (fscanf( fp, " %d ", p_h ) != 1)
          fgets(line, 80, fp);

  while (fscanf( fp, " %d", &max ) != 1)
          fgets(line, 80, fp);
  fgetc(fp);
  rgb_size=3*(*p_w)*(*p_h);
  (*p_rgb) = (unsigned char *) calloc(rgb_size, 1);
  if ((*p_rgb) != NULL)
    fread( (*p_rgb), rgb_size, 1, fp );

  fclose(fp);
  return 1;
}
```

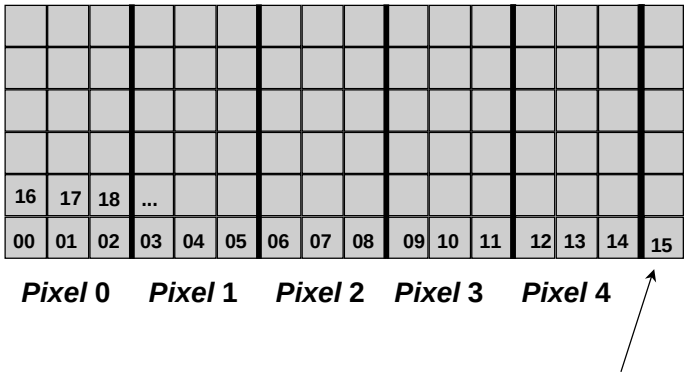
Programa Simples

```
void main(void){
  int w, h;                // dimensões da imagem
  unsigned char *rgb;      // bytes de rgb
  unsigned char r,g,b,grey; // componentes de cor
  int x,y;   long int k;

  if (ppm_read(&w,&h,&rgb,"test_in.ppm")==0) return;

  for (y = 0; y < h; y++)  {
    for (x = 0; x < w; x++)    {
        k = 3*(y*w+x);
        r = rgb[k];
        g = rgb[k+1];
        b = rgb[k+2];
        grey = (unsigned char)(0.3*r+0.6*g+0.1*b);
        rgb[k] = grey;
        rgb[k+1] = grey;
        rgb[k+2] = grey;
    }
  }
 ppm_write(w, h, rgb, "test_out.ppm");
 free(rgb);
}
```

Arquivo BMP



Organização dos *pixels* de uma imagem RGB no arquivo BMP

Microsoft Windows Bitmap - BMP

Características Principais

- Mono, 4-*bit*, 8-*bit*, 24-*bit*
- Tipo de compressão: RLE / não comprimido
- Tamanho máximo: 64K x 64K *pixels*
- Seções (versão 3):

Header
Info. Header
Palette
Bitmap Data

BMP - Header

```
typedef struct _Win3xBitmapHeader
  {
  WORD    Type;          /* Image file type 4D42h ("BM")*/
  DWORD  FileSize;       /* File size (bytes)         */
  WORD    Reserved1;     /* Reserved (always 0)     */
  WORD    Reserved2;     /* Reserved (always 0)     */
  DWORD  Offset;         /* Offset to bitmap data  in bytes     */
  } WIN3XHEAD;
```

BMP - Information Header

```
typedef struct _Win3xBitmapInfoHeader
   {
   DWORD  Size;               /*  Size of this Header  (40) */
   DWORD  Width;              /*  Image width (pixels)  */
   DWORD  Height;             /*  Image height (pixels)  */
   WORD   Planes;             /*  Number of Planes (always=1) */
   WORD   BitCount;           /*  Bits per pixel (1/4/8 or 24)*/
   DWORD  Compression;        /*  Compression (0/1/2)  */
   DWORD  SizeImage;          /* Size of bitmap (bytes) */
   DWORD  XPelsPerMeter;      /* Horz. resol.(pixels/m)          */
   DWORD  YPelsPerMeter;      /* Vert. resol.(pixels/m)  */
   DWORD  ClrUsed;            /* Num of colors in the image     */
   DWORD  ClrImportant;       /* Num of important colors   */
   } WIN3XINFOHEADER;
```

BMP - Palette

```
typedef struct _Win3xPalette
  {
  RGBQUAD Palette[ ];  /* 2, 16, or 256 elem.*/
  } WIN3XPALETTE;


typedef struct _Win3xRgbQuad
  {
  BYTE   Blue;   /* 8-bit blue component     */
  BYTE   Green;  /* 8-bit green component    */
  BYTE   Red;    /* 8-bit red component      */
  BYTE   Reserved;     /* Reserved (= 0)
      */
  } RGBQUAD;
```

BMP - Image Data

Notas

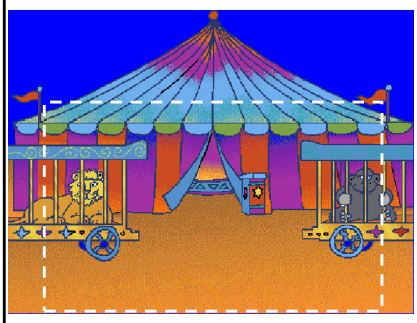
Cada *scan line* em um arquivo BMP é sempre um múltiplo de 4.

Imagens com1-, 4-, e 8-bits usam uma palheta de cores.

Imagens com 24-bits guardam a cor diretamente, na ordem azul, verde e vermelho.

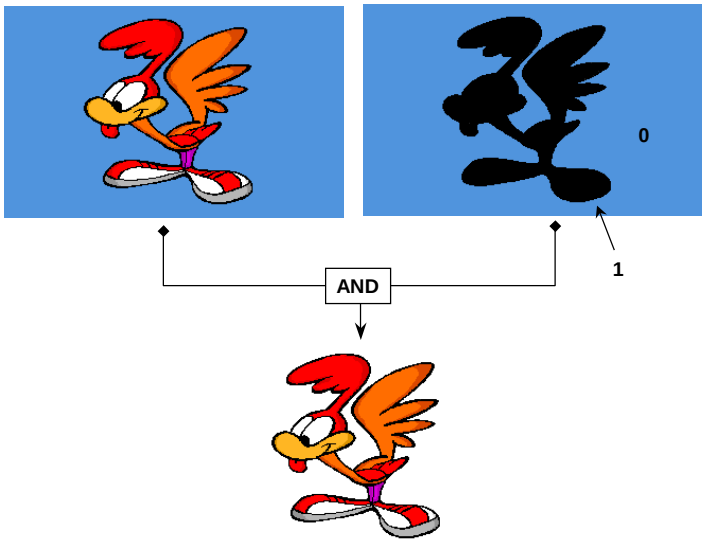
O armazenamento da imagem é sempre feito a partir do canto esquerdo inferior.

Composição de imagens com cor transparente



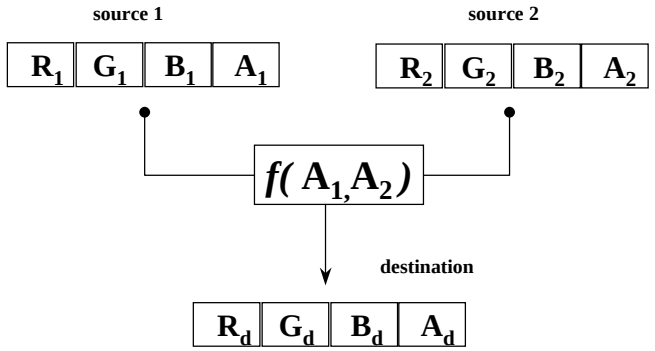
```
for (x_dst, y_dst ∈ Destination) {
  x_src = T_x^-1(x_dst, y_dst)
  y_src = T_y^-1(x_dst ,y_dst)
  cor = Source (x_src, y_src)
  if (cor != transparente)  Pixel (x_dst, y_dst, cor)
}
```

Composição de imagens com máscaras



Animação de *sprites*

Colocação de transparência no código da cor



por exemplo:

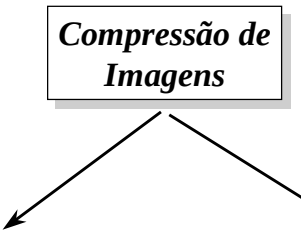
$R_d = A_1R_1+(1-A_1)R_2$

$G_d = A_1G_1+(1-A_1)G_2$

$B_d = A_1B_1+(1-A_1)B_2$

$A_d = A_1+(1-A_1)A_2$

Compressão de imagens



- Preserva exatamente o conteúdo da imagem
- Taxas de compressão 3 : 1

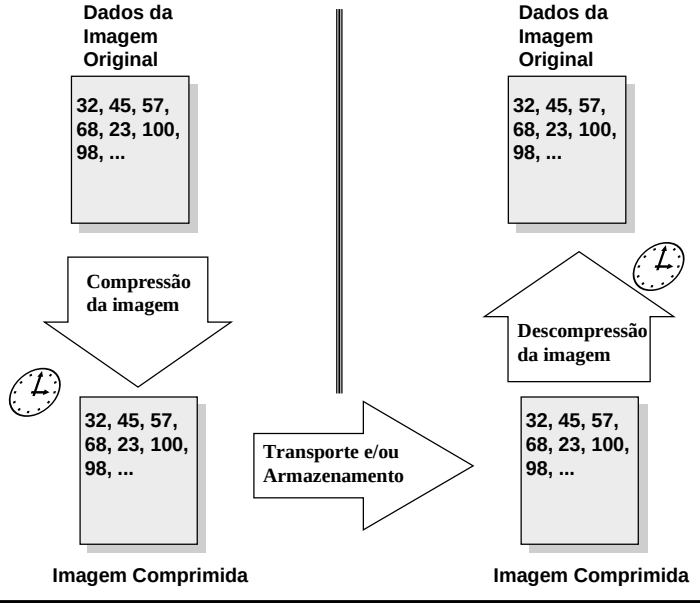
- Preserva de forma controlada o nível de qualidade da imagem
- Taxas de compressão que chegam a valores de mais de 100 : 1

Métodos de compressão

- ***Sem perdas***
  - » ***Run length encoding (RLE)*** *- repetição*
  - » ***Huffman coding*** *- histograma*
  - » ***Predictive coding*** *- diferenças*
  - » ***Block coding (LZW)*** *- dicionário*
- ***Com perdas***
  - » ***Truncation coding -*** *reduz a representação*
  - » ***Predictive coding -*** *descarta diferenças altas*
  - » ***Block coding -*** *dicionário aproximado*
  - » ***Transform coding -*** *descarta freqüências altas*

Métodos compostos: JPEG, MPEG

Processo de compressão e descompressão

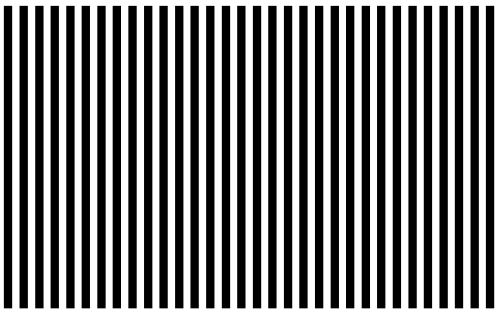


Fundamentos da Compressão de Imagens

A compressão de uma imagem é obtida quando se elimina a redundância de:

- **codificação**
- **entre *pixels***
- **psico-visual**

Redundância entre *pixels*



640 colunas x 480 linhas x 1 byte/pixel = 300 KBytes

480*(1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0) = ~18 Kbytes

onde 1 = 32 bytes de preto e 0 = 32 bytes de branco

Compressão - RLE

Objetivo

Reduzir a quantidade de dados redundantes.

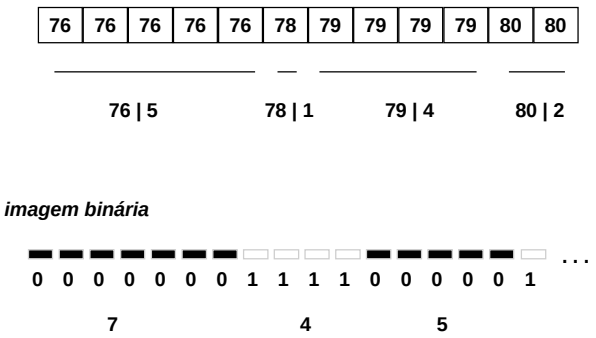
Exemplo

AAAAAAxxx ⟶ 6A3x

Caracterísiticas

Simples e rápido, porém a eficiência depende da imagem a ser comprimida.

Run-Length Encoding



Codificação de Huffman

s	p				
a2	0.4	0.4	0.4	0.4	0.6
a6	0.3	0.3	0.3	0.3	0.4
a1	0.1	0.1	0.2	0.3	
a4	0.1	0.1	0.1		
a3	0.06	0.1			
a5	0.04				

s	p				
a2	0.4 1	0.4 1	0.4 1	0.4 1	0.6 0
a6	0.3 00	0.3 00	0.3 00	0.3 00	0.4 1
a1	0.1 011	0.1 011	0.2 010	0.3 01	
a4	0.1 0100	0.1 0100	0.1 011		
a3	0.06 01010	0.1 0101			
a5	0.04 01011				

Redundância de Codificação

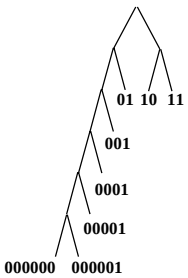
r	p(r)	Code 1	l(r)	l(r)p(r)	Code 2	l(r)	l(r)p(r)
0	0.19	000	3	0.57	11	2	0.38
1/7	0.25	001	3	0.75	01	2	0.50
2/7	0.21	010	3	0.63	10	2	0.42
3/7	0.16	011	3	0.48	001	3	0.48
4/7	0.08	100	3	0.24	0001	4	0.32
5/7	0.06	101	3	0.18	00001	5	0.30
6/7	0.03	110	3	0.09	000001	6	0.18
1	0.02	111	3	0.06	000000	6	0.12
	1.00		L_{avg}=	3.00		L_{avg}=	2.70

r_k = tons de cinza em uma imagem, k=0, 1, ..., τ−1

$p(r_k) = n_k / n$

onde n_k = número de pixels com tom r_k
n = número de pixels da imagem

$$L_{avg} = \sum_{k=0}^{\tau-1} l(r_k) p(r_k)$$



Resultado da Teoria da Informação

$$l_{opt}(r_k) = \log_2\left(\frac{1}{p(r_k)}\right) \qquad \textit{número de bits}$$

r	*p(r)*	*Code 1*	*l(r)*	*l(r)p(r)*	*Code 2*	*l(r)*	*l(r)p(r)*	*log(1/p)*	*log(1/p)*p*
0	0.19	000	3	0.57	11	2	0.38	2.4	0.46
1/7	0.25	001	3	0.75	01	2	0.50	2.0	0.50
2/7	0.21	010	3	0.63	10	2	0.42	2.3	0.47
3/7	0.16	011	3	0.48	001	3	0.48	2.6	0.42
4/7	0.08	100	3	0.24	0001	4	0.32	3.6	0.29
5/7	0.06	101	3	0.18	00001	5	0.30	4.1	0.24
6/7	0.03	110	3	0.09	000001	6	0.18	5.1	0.15
1	0.02	111	3	0.06	000000	6	0.12	5.6	0.11
∑=1.00			L_{avg} =	3.00		L_{avg} =	2.70	L_{opt} =	2.65