

# **Notas de Aula de Algoritmos e Programação de Computadores**

FLÁVIO KEIDI MIYAZAWA

*com a colaboração de*

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

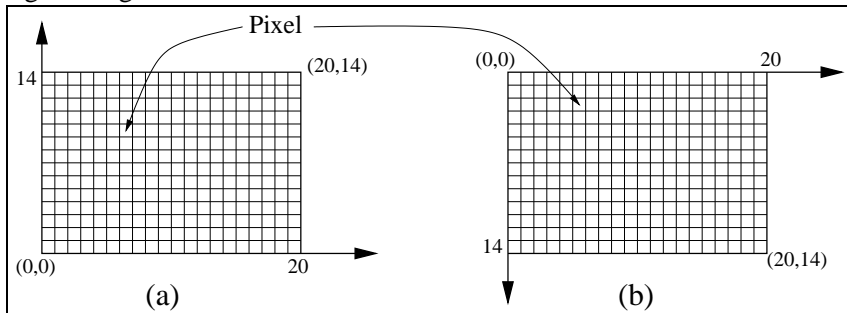
Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação  
UNICAMP  
Caixa Postal 6176  
13083-970 Campinas-SP  
{fkm,tomasz}@ic.unicamp.br

## 14 Figuras e Gráficos

Muitas vezes queremos visualizar objetos que estão sendo representados no modelo computacional, como figuras, curvas, retas, polígonos, etc. Embora muitos objetos possam ter características tridimensionais, na maioria das vezes precisamos traduzi-los de maneira que possam ser visualizados de maneira bidimensional, uma vez que o vídeo é um dispositivo de apresentação bidimensional. Além disso, um vídeo de computador apresenta os objetos como uma matriz de pontos forçando a apresentar os objetos de maneira discretizada. Assim, vamos considerar que o menor elemento usado para apresentar um gráfico/figura é um ponto (*pixel*). Deste modo, podemos considerar o desenho de uma reta no vídeo como sendo um conjunto de pontos, onde cada ponto da matriz tem coordenadas específicas nos dois eixos da matriz. Naturalmente, os objetos que iremos apresentar não precisam necessariamente estar definidos através de pontos, podendo ter uma representação interna contínua, como é o caso de funções de retas e curvas. Embora, no momento de sua apresentação na tela, esta deva ser discretizada para pontos, uma vez que o vídeo é uma matriz de pontos. Duas maneiras comuns de representação de uma tela gráfica com matriz de pontos por coordenadas são apresentadas nas duas figuras seguintes.



Para simplificar nossos programas, iremos usar a representação (a).

No processo de apresentação de objetos gráficos no vídeo, os pontos (representados por bits) são lidos de uma memória (memória de vídeo) e são apresentados no vídeo, em torno de 30 vezes por segundo.

Uma figura pode ser apresentada em um dispositivo gráfico usando cores, em geral usando uma quantidade limitada de memória para armazenar cada ponto de uma figura (e conseqüentemente usando um número limitado de cores). Esta memória dependerá da quantidade de cores possíveis que cada ponto pode assumir. Se consideramos uma figura em preto e branco, cada ponto pode ser representado por apenas um bit. Para representar 256 cores (ou tonalidades diferentes), precisaremos de um byte para representar cada bit. Neste caso, uma figura usando  $1024 \times 1024$  pontos, onde cada ponto assume 256 cores, usa uma memória de vídeo de pelo menos 1 Megabyte.

Uma maneira muito comum de se representar pontos coloridos é usando o formato RGB (*Red-Green-Blue*), através de três valores que indicam a intensidade das cores Vermelha, Verde e Azul (lembrando que uma determinada cor pode ser obtida misturando quantidades específicas destas três cores).

### 14.1 Usando o formato PPM – Portable PixMap

Para podermos trabalhar com gráficos de maneira independente de computador e sistema operacional, vamos usar um formato que pode ser visualizado em diversos ambientes computacionais. Vamos usar o formato PPM (Portable PixMap Format). Este formato representa uma matriz de pontos em um arquivo, usando o formato RGB, onde cada ponto colorido é apresentado por três bytes seguidos, representando a intensidade das cores vermelha, verde e azul. Cada byte apresenta um valor entre 0 e 255. Portanto, cada pixel pode ser uma de  $2^{24}$  cores (naturalmente, sua visualização dependerá da qualidade do vídeo usado). Uma vez que temos esta matriz de pontos, podemos formar uma figura mudando as cores dos elementos desta matriz, ponto a ponto.

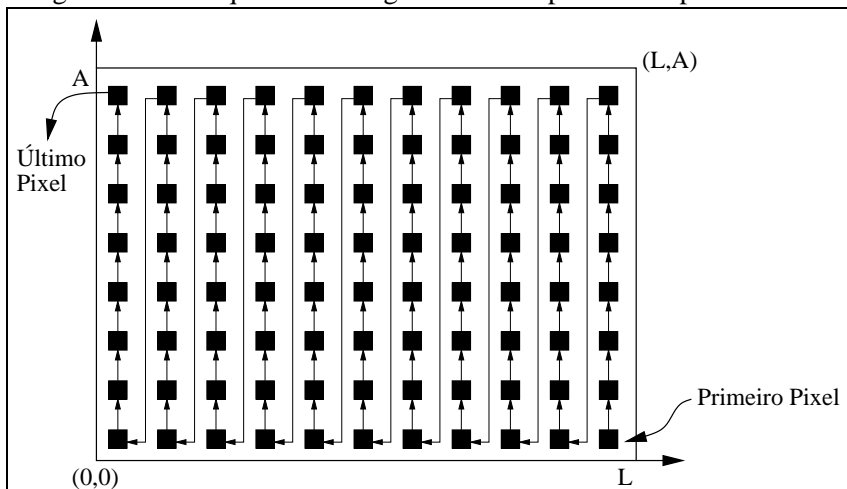
Existem diversos programas para visualização de figuras armazenadas em arquivos de diversos formatos gráficos, alguns destes livres ou a preços reduzidos (*Freeware e Shareware*) que podem visualizar figuras no formato PPM. Para o ambiente Linux, podemos usar o programa *GQview*, que pode ser obtido na página <http://gqview.netpedia.net/>. No ambiente Unix/X-Windows, podemos usar o program *xv*, disponível para a maioria das plataformas. Para o ambiente Windows podemos usar o programa *Irfan View32*, disponível livremente na rede <http://sunsite.auc.dk/tucows/g>

Por simplicidade, usaremos um tipo restrito para as figuras do formato PPM. Neste formato usamos um cabeçalho apresentando 4 valores descritos em formato texto (*P6 Largura Altura 255*), separados por um espaço em branco ou

em linhas diferentes. P6 são os caracteres ASCII 'P' e '6', e devem ser os primeiros dois bytes do arquivo. Em seguida devem vir a largura e altura da figura, *Largura* e *Altura*, descritos em números ASCII. Por fim, devem vir o três caracteres ASCII do número 255, indicando que a maior componente de uma cor (intensidade de Vermelho, Verde ou Azul) é 255. Caracteres neste cabeçalho que estejam em uma linha após o símbolo '#' não são considerados (comentários). Logo em seguida, como último byte deste cabeçalho, temos o byte de valor 10 (NewLine) e em seguida devem vir a seqüência de bytes que representam os pixels da figura. A cada três bytes (usando o sistema RGB) representamos um pixel (o primeiro byte para a intensidade de vermelho, o segundo para a intensidade de verde e o terceiro para a intensidade de azul), ao todo temos  $Largura \times Altura$  pixels. Um exemplo de um cabeçalho para uma figura com 200 pixels de largura e 300 pixels de altura é apresentado no quadro seguinte:

```
P6 200 300 255
```

A seqüência de pixels gravados no arquivo PPM segue a ordem apresentada pelas setas na seguinte figura:



Para trabalhar com a matriz de pixels, vamos usar um tipo chamado *TipoTela*, onde cada pixel é do tipo *TipoCor*, em RGB, declarados no seguinte quadro.

```

const
  MAXPONTOS = 200;
type
  TipoCor = record
    red,green,blue    : Byte;
  end;
  TipoTela = array[0..MAXPONTOS-1,0..MAXPONTOS-1] of TipoCor;

```

Com estas declarações podemos gerar uma figura, usando uma matriz de  $MAXPONTOS \times MAXPONTOS$  pixels. Além disso, podemos definir algumas cores para facilitar a atribuição de cores. Para definir a cor vermelha, podemos atribuir, em RGB, a intensidade máxima para vermelho (Red) e intensidade zero para as outras duas cores (Green e Blue). Podemos fazer o mesmo processo para definir as cores verde e azul. A cor branca é definida quanto temos a máxima intensidade (valor 255) para as componentes vermelha, verde e azul, e a cor preta é definida pela ausência de cores (valor zero). O quadro da página 138 mostra a definição destas cores e rotinas para iniciar a tela com cor branca e gravar a tela em um arquivo no formato PPM.

Obs.: Além do formato PPM, existem diversos outros formatos gráficos, sendo que alguns incorporam métodos de compressão de dados, como é o caso dos formatos *GIF* e *JPG*.

```

const
  MAXPONTOS = 200;
type
  TipoCor      = record
    Red,Green,Blue : Byte;
  end;
  TipoTela     = array[0..MAXPONTOS-1,0..MAXPONTOS-1] of TipoCor;
  TipoNomeArq  = string[100];
  TipoArquivoBinario = file of byte;
var Branco,Vermelho,Verde,Azul,Preto : TipoCor;
procedure InicializaAlgumasCores; { Cores: Branco, Preto, Vermelho, Verde, Azul }
begin { Definicao das cores usadas no formato RGB (Red-Green-Blue) }
  Branco.Red:=255; Branco.Green:=255; Branco.Blue:=255;
  Preto.Red:=0; Preto.Green:=0; Preto.Blue:=0;
  Vermelho.Red:=255; Vermelho.Green:=0; Vermelho.Blue:=0;
  Verde.Red:=0; Verde.Green:=255; Verde.Blue:=0;
  Azul.Red:=0; Azul.Green:=0; Azul.Blue:=255;
end;
procedure InicializaTela(var tela: tipotela); { Coloca todos os pixels como Branco }
var i,j : integer;
begin { Obs.: O ponto (0,0) está no canto inferior esquerdo }
  for i:=0 to MAXPONTOS-1 do
    for j:=0 to MAXPONTOS-1 do
      tela[i,j] := Branco;
end; { inicializatela }
procedure DesenhaPonto(var tela : tipotela; x,y:integer; Cor:TipoCor);
begin
  tela[x,y] := Cor; { Atribui o pixel (x,y) com Cor }
end;
procedure EscreveCabecalhoPPM(var arq : TipoArquivoBinario;largura,altura:integer);
var i : integer; str1,str2,cabecalho : string[100]; b:byte;
begin
  str(largura,str1); str(altura,str2); { Pegar a largura e altura como strings }
  cabecalho := ' P6 ' + str1 + ' ' + str2 + ' 255 ' ;
  for i:=1 to length(cabecalho) do begin
    b:=ord(cabecalho[i]);
    write(Arq,b);
  end;
  b:=10; write(Arq,b); { Caracter para Pular linha }
end;
procedure GravaTela(var t:TipoTela; nomearq:tiponomearq); { Grava a tela em arquivo }
var Arq: TipoArquivoBinario; lin,col : integer;
begin
  assign(Arq,NomeArq); rewrite(Arq);
  EscreveCabecalhoPPM(Arq,MAXPONTOS,MAXPONTOS);
  for col:= MAXPONTOS-1 downto 0 do
    for lin:= 0 to MAXPONTOS-1 do begin
      write(Arq,t[lin,col].Red); { Grava um pixel definido em RGB }
      write(Arq,t[lin,col].Green);
      write(Arq,t[lin,col].Blue);
    end;
  close(Arq);
end; { GravaTela }

```

## 14.2 Retas e Círculos

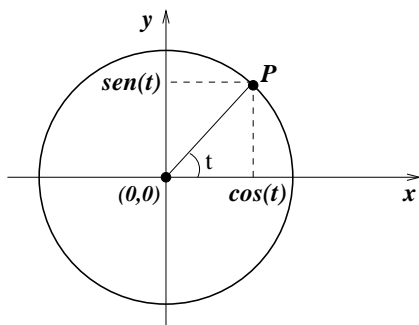
Uma vez que sabemos como imprimir um ponto em uma figura e gravá-la em um arquivo no formato PPM, vamos construir duas primitivas gráficas para construir desenhos mais complexos, as rotinas para desenhar um segmento de reta e um círculo. Para estas duas operações, vamos construir rotinas paramétricas, que definem os pontos do objeto através de um parâmetro.

Podemos definir os pontos de um segmento de reta  $R$ , ligando dois pontos  $P_1 = (x_1, y_1)$  e  $P_2 = (x_2, y_2)$ , usando um parâmetro  $t$  no intervalo  $[0, 1]$  como  $R = \{(x(t), y(t)) : t \in [0, 1]\}$ , onde

$$x(t) = x_1 + (x_2 - x_1) \cdot t, \quad y(t) = y_1 + (y_2 - y_1) \cdot t.$$

Note que quando  $t = 0$  obtemos o ponto  $P_1$  e quando  $t = 1$  obtemos o ponto  $P_2$ . Para desenhar o segmento de reta  $R$ , podemos “plotar” alguns pontos de  $R$ , cuja quantidade dependerá da dimensão da matriz de pontos, escolhendo valores de  $t$  distribuídos no intervalo  $[0, 1]$ .

Para desenhar um círculo  $C$ , com centro  $(c_x, c_y)$  e raio  $r$ , também vamos usar um parâmetro  $t$  que refletirá um ângulo no intervalo  $[0, 2\pi)$ . Primeiramente, vamos verificar a seguinte representação para um círculo com centro em  $(0, 0)$  e raio 1.



Note que as coordenadas do ponto pertencente ao círculo, definido pelo ângulo  $t$  é o ponto  $P = (\cos(t), \sin(t))$ . Para definir um círculo de raio maior, basta multiplicar as coordenadas dos pontos pelo valor de  $r$ , e para que o círculo seja definido com centro  $(c_x, c_y)$ , adicionamos os valores de  $c_x$  e  $c_y$  nas coordenadas correspondentes. Desta forma, obtemos a seguinte representação paramétrica para os pontos do círculo:  $C = \{(x(t), y(t)) : t \in [0, 2\pi)\}$  onde

$$x(t) = c_x + r \cdot \cos(t), \quad y(t) = c_y + r \cdot \sin(t).$$

Agora podemos definir uma figura ou objeto gráfico, mais elaborado, como um conjunto de objetos primitivos (retas e círculos). Vamos definir este objeto mais elaborado como o tipo *TipoFigura* da seguinte maneira:

```

const MAXOBJETOS = 200; {Quantidade máxima de objetos que uma figura pode ter}
type
  TipoCor   = record
    red,green,blue : Byte;
  end;
  TipoForma = (FormaCirculo,FormaReta);
  TipoObjeto = record
    Cor           : TipoCor;
    case Forma     : TipoForma of
      FormaCirculo : (centrox,centroy:integer; Raio:Real);
      FormaReta    : (p1x,p1y,p2x,p2y:integer);
    end;
  TipoFigura   = record
    nobjetos : 0..MAXOBJETOS; {quantidade de objetos, armazenados em vetor}
    objeto   : array[1..MAXOBJETOS] of TipoObjeto;
  end;

```

A seguir, apresentamos um programa, contendo os procedimentos para desenhar um segmento de reta e um círculo, para gravar uma figura de um desenho de um rosto construído com círculos e retas e gravado em um arquivo no formato PPM.

```

program ProgFigura;
const
  MAXPONTOS = 200;
  MAXOBJETOS = 200;
type
  TipoCor = record
    red,green,blue : Byte;
  end;
  TipoTela = array[0..MAXPONTOS-1,0..MAXPONTOS-1] of TipoCor;
  TipoForma = (FormaCirculo,FormaReta);
  TipoObjeto = record
    Cor : TipoCor;
    case Forma : TipoForma of
      FormaCirculo : (centrox,centroy:integer;Raio:Real);
      FormaReta : (p1x,p1y,p2x,p2y:integer);
    end;
  TipoFigura = record
    nobjetos : 0..MAXOBJETOS;
    objeto : array[1..MAXOBJETOS] of TipoObjeto;
  end;
  TipoNomeArq = string[100];
  TipoArquivoBinario = file of byte;
var Branco,Vermelho,Verde,Azul,Preto : TipoCor;
{ Incluir as seguintes rotinas }
{procedure InicializaTela(var tela : tipotela);}
{procedure DesenhaPonto(var tela : tipotela; x,y:integer; cor:TipoCor);}
{procedure GravaTela(var t : TipoTela; nomearq: tiponomearq);}
procedure DesenhaReta(var tela : tipotela; x1,y1,x2,y2:integer; cor:TipoCor);
var t,delta,maxit,x,y : real; {representacao parametrica}
begin
  delta := 1/MAXPONTOS;
  x:=x1; y:=y1; t:=0;
  while t<=1 do begin {0 <= t <= 1}
    x:=x1+(x2-x1)*t;
    y:=y1+(y2-y1)*t;
    DesenhaPonto(tela,round(x),round(y),cor);
    t:=t+delta;
  end;
end;
procedure DesenhaCirculo(var tela : tipotela; cx,cy:integer;raio:real; cor:TipoCor);
var t,delta,maxit,x,y,DoisPi : real; {representacao parametrica}
begin
  DoisPi := PI * 2;
  delta := 1/MAXPONTOS;
  t:=0;
  while t<=DoisPi do begin { 0 <= t <= DoisPi} {Usando Coordenadas Polares}
    x:=cx + raio * cos(t);
    Y:=cy + raio * sin(t);
    DesenhaPonto(tela,round(x),round(y),cor);
    t:=t+delta;
  end;
end;

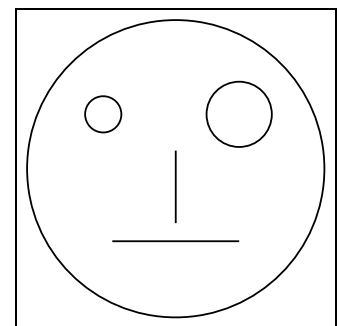
```

```

procedure DesenhaObjeto(var tela : TipoTela; obj : TipoObjeto);
begin
  case Obj.Forma of
    FormaCirculo : DesenhaCirculo(tela, obj.centrox, obj.centroy, obj.raio, obj.cor);
    FormaReta    : DesenhaReta(tela, obj.p1x, obj.p1y, obj.p2x, obj.p2y, obj.cor);
  end; { case }
end;
procedure DesenhaFigura(var t : TipoTela; var Fig:TipoFigura);
var i:integer;
begin
  for i:=1 to Fig.nobjetos do DesenhaObjeto(t,Fig.objeto[i]);
end;
procedure EscreveString(var arq : TipoArquivoBinario;str:string);
var i : integer;
begin
  for i:=1 to length(str) do write(Arq,ord(str[i]));
end;
var t      : tipotela;
    face : TipoFigura;
begin
  {Definicao de algumas cores usando o formato RGB (Red-Green-Blue) }
  Branco.Red:=255;  Branco.Green:=255; Branco.Blue:=255;
  Preto.Red:=0;    Preto.Green:=0;   Preto.Blue:=0;
  Vermelho.Red:=255; Vermelho.Green:=0; Vermelho.Blue:=0;
  Verde.Red:=0;    Verde.Green:=255;  Verde.Blue:=0;
  Azul.Red:=0;     Azul.Green:=0;     Azul.Blue:=255;
  inicializatela(t);
  {cabeca}
  face.objeto[1].Forma := FormaCirculo; face.objeto[1].cor := Preto;
  face.objeto[1].centrox := 100;   face.objeto[1].centroy := 100;
  face.objeto[1].raio := 50;
  {olho esq}
  face.objeto[2].Forma := FormaCirculo; face.objeto[2].cor := Azul;
  face.objeto[2].centrox := 80;   face.objeto[2].centroy := 120;
  face.objeto[2].raio := 6;
  {olho dir}
  face.objeto[3].Forma := FormaCirculo; face.objeto[3].cor := Verde;
  face.objeto[3].centrox := 120;   face.objeto[3].centroy := 120;
  face.objeto[3].raio := 10;
  {nariz}
  face.objeto[4].Forma := FormaReta;   face.objeto[4].cor := Preto;
  face.objeto[4].p1x := 100;   face.objeto[4].p1y := 110;
  face.objeto[4].p2x := 100;   face.objeto[4].p2y := 90;
  {boca}
  face.objeto[5].Forma := FormaReta;   face.objeto[5].cor := Vermelho;
  face.objeto[5].p1x := 80;   face.objeto[5].p1y := 80;
  face.objeto[5].p2x := 120;   face.objeto[5].p2y := 80;

  face.nobjetos :=5; {face foi definido com 5 objetos graficos}
  DesenhaFigura(t,face);
  GravaTela(t,'saida.ppm');
end.

```



### 14.3 Exercícios

1. O formato PGM (Portable Grayscale pixMap) é muito parecido com o formato PPM. Neste formato podemos representar figuras monocromáticas, com intensidade nos pixels. I.e., todos os pixels tem apenas uma cor, mas podem ter intensidade que pode ir de 0 a 255. A seguir, listamos as diferenças/semelhanças nos formatos destes dois arquivos:
  - (a) O cabeçalho é descrito da mesma maneira que o cabeçalho de um arquivo PPM, mas em vez de 'P6', você deve colocar 'P5'.
  - (b) A seqüência de pixels segue a mesma ordem que no formato PPM.
  - (c) Cada pixel usa apenas 1 byte (em vez de 3). O valor deste byte indica a intensidade do pixel. O valor 255 para o branco e 0 para preto. Os valores entre 0 e 255 indicarão uma intensidade intermediária.

Faça um programa que desenha a figura apresentada na página 141, mas em formato PGM.

2. Faça outras primitivas gráficas, além do segmento de reta e o círculo: Retângulo, Retângulo Preenchido (todo preenchido com uma cor), Círculo Preenchido, Poligono fechado de  $n$  pontos.
3. Faça um programa contendo uma rotina que *plota* uma função  $f(x)$  em um arquivo de formato PPM, para valores de  $x$  em um intervalo  $[x_1, x_2]$ . A rotina deve ter o seguinte cabeçalho:

**procedure** PlotaFuncao(f: TipoFuncaoReal;  $x_1, x_2, y_1, y_2$ :real; NomeArquivo:TipoString);

onde TipoFuncaoReal foi definido como:

**type** TipoFuncao = function(x:real):real;

Com isso, iremos passar a função a plotar como parâmetro. Para plotar os pontos  $(x, f(x))$ , discretize os valores de  $x$  no intervalo  $[x_1, x_2]$ , imprima os pontos  $f(x)$  que caem no intervalo  $[y_1, y_2]$ . Além disso, imprima as retas dos eixos  $x = 0$  e  $y = 0$ , caso estas estejam nos intervalos de impressão.

Faça uma imagem de tamanho  $L \times C$  pontos (com digamos  $L = 300$  e  $C = 300$ ). Naturalmente você terá que fazer uma reparametrização, de maneira que o ponto  $(0, 0)$  na matriz de pontos seja equivalente a posição  $(x_1, y_1)$  e o ponto  $(L, C)$  na matriz de pontos seja equivalente a posição  $(x_2, y_2)$ .

4. Faça uma rotina que tenha como parâmetros, um vetor de valores reais positivos com  $n$  elementos. Faça um programa que gere gráficos de barras, histograma e tipo torta, de acordo com a escolha do usuário.
5. A figura de uma letra pode ser armazenada em uma matriz de tamanho  $8 \times 8$ . Gere matrizes para algumas letras, atribuindo os valores dos pontos através de um programa (vc pode usar as rotinas de reta e círculo para isso). Uma vez que você gerou estas letras, faça uma rotina que tem como parâmetro uma posição (bidimensional) e uma letra. A rotina deve *imprimir* a matriz que forma a letra na tela de pontos. Isto permitirá que possamos escrever na figura.
6. Faça uma rotina que tem como parâmetro os nomes de dois arquivos e um valor real positivo,  $\alpha$ . A rotina lê o arquivo do primeiro nome, que é uma figura no formato PPM e gera outro arquivo que é a mesma figura, mas com tamanho reparametrizado de um fator  $\alpha$ .