APENDICE A

FUNÇÕES GRÁFICAS

Este apendice apresenta um resumo das funcoes gr§¯cas do Turbo C. Sao apresentados alguns exemplos simples das funcoes mais utilizadas. Primeiro sao tratadas funcoes do modo texto. Na secao 2 sao descritas algumas funcoes do modo gr§¯co.

a.1 - MODO TEXTO

As funcoes do modo texto nao necessitam de processadores gr§¯cos especiais. Para serem usados ¶ necess§rio apenas incluir o arquivo conio-h no in¶cio do programa.

a.1.1 - Windows

Num contexto geral WINDOW ¶ uma §rea retangular da tela que forma um contorno para outputs no v¶deo. Windows sao usadas em diversas interfaces gr§¯cas (no pr¶prio ambiente integrado do Turbo C, por exemplo).

Esta secao exempli¯ca algumas funcoes que permitem manipular texto nas window. Um ponto importante ¶ que dentro de uma windows, as coordenadas sao relativas as dimensoes dos limites da window.

O programa que segue de¯ne uma window e determina as cores do texto e do fundo, depois um texto ¶ sobre-escrito continuamente at¶ ser precionada a tecla \f". Quando uma windows est§ cheia e palavras forem escritas adicionalmente, o texto se mover§ para cima.

```
/* exemplo de window */
# include <conio-h>
void main (void)
fint esq = 10, dir = 70,
  sup = 15, inf = 20, y;
  char tecla;
  y = inf - sup;
  window (esq, sup, dir, inf) /* de¯ne window */
  textcolor (RED); /* cor do texto */
  textbackground (GREEN); /* cor fundo /*
  gotoxy (25, y); /* posiciona dentro da window /*
  do fcputs (\ESCOLHA SUA OPÇÃO ");
    tecla = getche ();g
  while (tecla != \f");
g
```

A função window() define a área da tela a ser ocupada pela window. Os seus parâmetros são inteiros dentro dos limites do modo texto a ser usado. Na resolução de 80 x 25 os limites vão da linha e coluna 1 até a linha 25 e coluna 80.

As funções que definem a cor de fundo, textbackground(), e a cor do texto, textcolor(), aceitam inteiros de 0 a 7 ou de 0 a 15 respectivamente. A cada um destes números corresponderá uma cor, se seu monitor for colorido.

No arquivo conio.h os nomes RED e GREEN são definitos, bem como os outros nomes mostrados na tabela que segue. Assim, como parâmetro destas funções, é possível usar-se também os nomes da tabela.

Número	Nome	Cor
0	BLACK	preta
1	BLUE	azul
2	GREEN	verde
3	CYAN	turquesa
4	RED	vermelho
5	MAGENTA	rosa-choque
6	BROWN	marron
7	LIGHT GRAY	cinza claro
8	DARK GRAY	cinza escuro
9	LIGHT BLUE	azul claro
10	LIGHT GREEN	verde claro
11	LIGHT CYAN	turquesa claro
12	LIGHT RED	vermelho caro
13	LIGHT MAGENTA	rosa
14	YELLOW	amarelo
15	WHITE	branco

Os eixos podem aparecer \piscando" se o número 128 for adicionado ao valor da cor ou se o nome BLINK for adicionado ao nome da cor; como por exemplo RED + BLINK.

A função gotoxy() move o cursor para qualquer localização dentro da window. As coordenadas a serem fornecidas são medidas relativamente a window e não a tela. Assim as coordenadas (30 , 3) correspondem a um ponto no meio da window do exemplo. A função gotoxy(30 , 3) moveria o cursor 30 colunas a partir da extremidade esquerda da window e a 3 linhas da extremidade superior da window.

As quatro funções de window vistas nesta seção não têm valor de retorno. São do tipo void. Os parâmetros de todas quatro são também do mesmo tipo: inteiros.

a.1.2 - Funções de Cópia, Armazenamento e Recuperação de Textos

A função movetext() copia uma área de texto em outra posição na tela. Esta função usa cordenadas da tela e não da window.

```
/* exemplo de movimento de window */
```

```
#  include < conio.h >
# de¯ne LEFT 26
# de¯ne TOP 7
# de¯ne RIGHT LEFT + 30
# de¯ne BOTTOM  TOP + 10
# de¯ne CORES 16
void main (void)
f
   int i, nL, nT;
   window (LEFT, TOP, RIGHT, BOTTOM );
   for (i =  0; i < 100 ; i + + )
   f
   textcolor (i %  CORES); /* muda a cor */
   cputs (\AVISO "); /* escreve */
   g
for (i =  0, nL =  1, nT =  1; i < 4; i+ + , nL =  nL +  20, nT =  nT +  8)
f delay (2000); /* pausa */
movetext (LEFT, TOP, RIGHT, BOTTOM , nL, nT); /* move a window */
g
g
```

O exemplo acima exempli¯ca o uso da fun»c~ao movetext ( ). Neste exemplo a window ¶e copiada sucessivamente, com o limite superior esquerdo (LEFT, TOP), ocupando as novas posi»c~oes (nL , nT). As c¶opias ser~ao realizadas mesmo que os textos se sobreponham.

O texto de uma ¶area da tela pode ser armazenado na mem¶oria e depois recuperado para alguma posi»c~ao qualquer da tela. Isto pode ser ¶util quando voc^e quizer abrir windows sobre textos existentes na tela e depois recuperar o texto anterior. Isto ¶e feito em v¶arios pontos no ambiente integrado do Turbo C. O exemplo que segue salva a tela existente antes do in¶icio do programa e depois a restaura quando o programa ¯naliza.

```
/* recupera»c~ao de tela */
#  include < conio.h >
int buf [80] [25] /* buffer de toda a tela */
void main (void)
f
   int x, y, i;
   gettext (1, 1, 80, 25, buf );
   /* armazena a tela */
   x =  wherex ( );
   y =  wherey ( ); /* obt¶em  a posi»c~ao original do cursor */
   clear ( ); /* limpa a tela */
   getche ( ); /* espera uma tecla ser precionada */
   puttext (1, 1, 80, 25, buf ); /* restaura a tela */
   gotoxy (x , y); /* restaura posi»c~ao do cursor */
g
```

A funcao gettext (esq, sup, dir, inf, pont) armazena a área definida pelo retangulo (esq, sup, dir, inf) no endereco de memória definido pelo último parâmetro: pont. Este buffer de memória deve ser suficente para armazenar a área desejada. Cada caracter é armazenado na memória de vídeo em 2 bytes: um para o código ASCII do caracter e o outro para os atributos.

O buffer necessita de dois bytes para cada caracter da tela, pois isto é definido uma área do tipo int. Foi usado um array de duas dimensoes, com largura e altura igual às da área a ser armazenada. Neste caso um array de uma dimensao de 2000 posicoes (80 x 25) também funcionaria perfeitamente.

A funcao clear( ) limpa a tela (clear screen). A localizacao do cursor é obtida pelas funcoes wherex ( ) e wherey ( ), que retornam a coluna e a linha em que o cursor se encontra respectivamente. Estas três funcoes nao aceitam parâmetros.

A funcao puttext ( ) é semelhante a gettext, ela move o texto da memória para a tela. O uso destas funcoes dá um \ar" profissional em qualquer programa.

a.1.3 - Outras Funcoes do Modo Texto

Uma linha nova pode ser inserida em uma window na posicao corrente do cursor através da funcao void insline (void).

Uma linha existente pode ser apagada usando a funcao void delline (void). A funcao clreol( ) (clear to end of line) deleta da posicao do cursor ao fim da linha.

Um texto pode mudar sua intensidade pelo uso das funcoes void highvideo (void) e void lowvideo (void). A intensidade anterior pode ser restabelecida pela funcao normvideo( ).

As funcoes para escrever em window sao: cprintf(string, argumentos), cputs(string) e putch(char). A primeira funciona como printf só que em windows. A segunda escreve uma string na window, como puts.

A funcao textmode ( ) muda o modo texto corrente. Esta funcao nao serve para mudancas entre o modo texto e gráfico, ela muda formatos apenas no modo texto. Ela pode ter como parâmetro os valores ou nomes da tabela abaixo:

Valor	Nome	funcao
-1	LASTMODE	restabele o modo anterior
0	BW40	preto e branco com 40 colunas
1	C40	cores com 40 colunas
2	BW80	preto e branco com 80 colunas
3	C80	cores com 80 colunas
7	MONO	monocromático, 80 colunas

É importante lembrar que nao é possível, no Turbo C, windows multiplas: apenas uma estará ativa a determinado momento. Assim as funcoes como cputs( ) estarao relacionadas sempre com a última window definida.

a.2 - MODO GRÁFICO

Você pode escolher para o seu programa diversos formatos de resolução e cores, a possibilidade de apresentar apenas textos ou representar também gráficos. No modo texto, grupos de pixeis correspondentes aos caracteres são acesso.

Determinar que o sistema use funções do modo gráfico é mais complexo que determinar que funções do modo texto sejam usadas. Uma das causas desta maior complexidade é que as funções do modo gráfico estão em arquivos bem maiores. Três tipos de arquivos devem ser considerados quando se estabelece que o sistema usará funções gráficas: o arquivo graphics-lib; os drivers gráficos e o arquivo graphics.h.

a.2.1 - O Arquivo graphics-lib

Este arquivo contém o código das funções. Para usar alguma função do modo gráfico, o arquivo biblioteca graphics-lib deve ser \linkado" com o programa. Os arquivos normais de biblioteca (por exemplo, o arquivo cs.lib no modelo \small" de memória) são \linkados" automaticamente no seu programa. Porém o arquivo graphics.lib não é incluido automáticamente nos procedimentos de linkedição, e pode ser incluido de duas maneiras.

A primeira maneira de incluir a biblioteca gráfica é a mais simples, porém a mais tediosa quando feita repetidamente. Nesta maneira cria-se um arquivo \project" (com terminação .prj) para cada programa. Este arquivo pode ser editado normalmente como um arquivo fonte usual. Suponha que seu programa esteja no ¯le program.c, o arquivo project será \salvado" com o nome program.prj e terá as linhas

```
program
graphics.lib
```

Quando você quizer fazer um \executável" do seu programa, você deve fornecer o nome do arquivo \project" na opção Project Name do menu Project. A desvantagem deste modo é que você deve criar um arquivo .prj para cada programa e especi¯car seu nome toda vez que iniciar o TC.

A segunda maneira é mais difícil de utilizar mas se torna invisível depois de estabelecida. Esta maneira combina permanentemente os arquivos bibliotecas usados no seu programa (o arquivo cs.lib por exemplo). Esta maneira torna o arquivo biblioteca maior. Neste modo graphics.lib deve ser adicionado a cada biblioteca que for usada. Por exemplo se voce usar os modelos de memória small, medium e large você deve adicioná-lo às bibliotecas cs.lib, cm.lib, cl.lib. Porém uma vez incorporada a parte gráfica a uma biblioteca você não precisa mais se preocupar com as funções gráficas que passarão a serem acessadas como qualquer outra função.

O utilitário TLIB do TC serve para fazer a combinação de bibliotecas. Para usá-lo copie este programa para o seu diretório nLIB, veri¯que se as bibliotecas que você vai usar estão também neste diretório (suponha que você quer combinar cs.lib e graphics.lib) então a linha abaixo combinará as bibliotecas.

```
tlib cs + graphics.lib
```

Ao fazer a combinação você notará que tamanho de cs.lib aumentará. O arquivo cs.lib original é chamado agora cs.bak.

a.2.2 - Os Arquivos de Drivers

Diferentes routinas de vídeo são necessários para diferentes tipos de placas gráficas. Placas CGA precisam de um driver específico, placas EGA de outro driver, placas VGA de um terceiro. Os códigos destes diferentes drivers não estão todos juntos (o que ficaria muito grande) mas separados em diferentes arquivos com a terminação .bgi.

A incorporação do arquivo .bgi ao seu programa pode ser feito de diversas maneiras. Na primeira delas você "diz" ao seu programa onde encontrar o driver apropriado através dos argumentos da função initgraph ( ). Quando a função initgraph ( ) for executada pelo seu programa, ela faz o sistema "alocar" memória para o driver, procura o arquivo do driver e o "carrega" na memória. Esta é a maneira mais fácil e funciona bem quando você escrever programas para você mesmo.

Se você planeja distribuir seu programa é melhor que ele ocupe um único arquivo e não esteja em módulo separados de programa e arquivos drivers. Existem dois métodos para fazer isto que também funcionam melhor quando seu programa for executado em computador onde o Turbo C não estiver instalado. Nestas formas os drivers (ou mesmo as fontes (.char) no caso de usar-se texto no modo gráfico, como se verá adiante) são "linkados" ao aplicativo no procedimento de "linkedição". Nestes processos se cria um arquivo único que não precisa de nada mais para se executado.

Em ambos os métodos os arquivos drivers e fontes devem ser convertidos em arquivos .obj. Vamos primeiro ver como isto é feito. O utilitário BGIOBJ do Turbo C faz esta conversão. Para usá-lo, por exemplo para converter o arquivo cga.bgi, digite a linha (supoem-se que os arquivos estejam no mesmo diretório):

```
C > bgiobj cga
```

para converter a fonte de caracteres góticos (goth.chr) para .obj digite:

```
C > bgiobj goth
```

desta maneira serão gerados os arquivos cga.obj e goth.obj. O utilitário BGIOBJ dirá qual será o "public name" do driver. Este nome é usado para "registrar" o driver. No exemplo anterior os "public names" são _CGA_driver e _gothic_font.

O primeiro método apenas liga em um módulo único todos os .obj formando um arquivo único. O segundo método "linka" os drivers e fontes com o arquivo arquivo biblioteca normalmente usado (o cs.lib por exemplo), e toda a vez que você "linkar" o arquivo biblioteca estará automaticamente "linkando" também o driver e a fonte.

Para combinar os arquivos .obj com o seu programa acrescente estes no seu project, por exemplo (se seu programa se chama prog1.c e os .obj estiverem num mesmo diretório)

proc1
cga.obj
goth.obj

Quando você carregar este project na opção Project Nome do menu project, e teclar F9 tornará todos um .EXE único.

Para combinar os .obj no arquivo biblioteca faz-se um procedimento similar ao da seção a.2.1. Os arquivos a serem usados já convertidos em .obj devem estar no mesmo diretório do utilitário TLIB e da biblioteca de memória a ser usada. Se voce, por exemplo, quizer ligar o driver CGA e a fonte GOTHIC na biblioteca de memória small deve usar o comando

```
tlib cs + cga + goth
```

Neste método os \public names" fornecidos pelo utilitário BGIOBJ devem ser usados no código fonte do seu programa. Como usá-los será descrito nas próximas seções.

a.2.3 - O Arquivo graphics.h

O arquivo cabeçalho graphics.h contém o protótipo das funções gráficas (exceto as do modo texto). Estes protótipos são necessários pois são do tipo far, devendo ser incluidos em todos os arquivos que usarem funções gráficas (#include < graphics.h>). Este arquivo contém também muitas constantes, em #definenes, que são muito utilizados nas funções gráficas. As seções que seguem apresentam algum exemplos das funções gráficas mais usuais.

a.2.4 - Inicialização

Antes de usar qualquer função do modo gráfico, o sistema gráfico deve ser inicializado pela função

```
void far initgraph (endDriver, endMode, endPath)
int far * endDriver /*endereço do número do driver */
int far * endMode /*endereço do número do modo */
int far * endPath /*endereço e path do driver */
```

Esta função \carrega" o driver gráfico especificado no primeiro parâmetro, e modifica o display para o modo especificado no segundo parâmetro. O exemplo a seguir desenha círculos e linhas:

```
/* retas_e_circulo */
# include < graphics.h >
void main (void)
f int driver, mode;
  int raio = 80, xc = 100, yc = 100;
  int x1 = 0, y1 = 0 , x2 = 150, y2 = 150;
```

```
driver = CGA; /* driver CGA */
mode = CGACO; /* palette 0 */
intgraph (& driver, & mode, "C:\tc\lib"); /* inicio modo grafico */
circle (xc, yc, raio); /* desenha circulo */
line (x1, y1, x2, y2) /* desenha uma linha */
line (x2, y2, x1, y2) /* desnha outra linha */
getche ( ); /* o desenho permanece até uma tecla ser precionada */
closegraph ( ); /* finaliza o modo grafico */
```

Os valores possíveis para o primeiro parâmetro são nomes, constantes, armazenados no arquivo graphics.h. Você pode usar os nomes ou os valores da tabela abaixo.

Valor	Nome	OBS
0	DETECT	O sistema detecta o melhor modo
1	CGA	
2	MCGA	
3	EGA	com 256k de memória
4	EGA64	com 64k de memória
5	EGAMONO	
6	IBM8514	
7	HERCMMONO	
8	ATT400	
9	VGA	
10	PC3270	

Para cada um destes drivers há diversos modos de operação. A tabela a seguir dá exemplos de modos possíveis para os drivers CGA, EGA e VGA. Este modo é determinado pelo segundo parâmetro da função initgraph ( ).

DRIVER	VALOR	NOME C	RESOLUÇÃO	PÁGINAS	CORES
CGA	0	CGAC0	320 x 200	1	4 - palete 0
	1	CGAC1	"	1	4 - palete 1
	2	CGAC2	"	1	4 - palete 2
	3	CGAC3	"	1	4 - palete 3
	4	CGAHI	640 x 200	1	2
EGA	0	EGALO	640 x 200	2	16
	1	EGAHI	640 x 350	2	16
VGA	0	VGALO	640 x 200	2	16
	1	VGAMED	640 x 350	2	16
	2	VGAHI	640 x 480	1	16

O terceiro argumento de initgraph ( ) é o path (localização nos diretórios do sistema) para o driver gráfico. Neste exemplo assumiu-se que o driver se encontra no subdiretório lib do diretório tc.

Se o processo escolhido para incorporação dos arquivos, ôle, não for o primeiro descrito na seção a.2.2, mas sim algum dos outros que possibilitem a obtenção de um único arquivo

é necessário algumas modificações no programa anterior. Estas modificações são para informar ao sistema que o arquivo driver já está linkeditado e são feitas pelas funções registerbgidriver ( ) (e registerbgifont ( ) no caso das fontes também estarem em módulos obj). O argumento desta função é o nome fornecido pelo utilitário BGIOBJ quando os arquivos .obj foram criados, sem o travessão inicial. Esta função deve aparecer no programa antes da chamada à função initgraph ( ). O exemplo anterior, neste caso, torna-se

```
/* retas_e_circulo */
# include <graphics.h>
void main (void)
   f int driver = CGA, mode = CGAC0;
   int raio = 80, xc = 100, yc = 100;
   int x1 = 0, y1 = 0, x2 = 150, y2 = 150;
   registerbgidriver (CGA_driver); /*registra o driver */
   initgraph (& driver, & mode, " ");
   cicle (xc, yc, raio);
   line (x1, y1, x2, y1)
   line (x2, y2, x1, y2);
   getche ( );
   closegraph ( ); g
```

Observe que neste caso o último parâmetro é desnecessário pois o driver não precisará ser localizado.

a.2.5 - Funções de desenho

As funções de desenho como line ( ), que desenha linha entre duas localizações, e circle ( ), que desenha um círculo de raio e centro dados, têm suas coordenadas expressas em pixeis. Seus parâmetros são portanto inteiros, variando de x = 0 e y = 0, no alto a esquerda do vídeo, até a resolução do modo gráfico escolhido em colunas (x) e linhas (y).

Outras funções são por exemplo: rectangle (left, top, right, botton) que desenha um retângulo de extremidades fornecidas, e ellipse (xc, yc, angulo-inicial, angulo-final, raiox, raioy) que desenha uma elipse com ponto central, xc e yc e com ângulo inicial e final medidos em graus a partir da extremidade de direita e na horizontal (posição do relógio às 3 horas) no sentido contrário aos ponteiros do relógio.

Estes desenhos são feitos na cor corrente. A cor corrente é determinada pela função setcolor (color), mas depende do modo gráfico usado. Por exemplo o número zero, nos modos CGAC0, CGAC1, CGAC2 e CGAC3, corresponde a cor preta, mas as outras 3 cores possíveis, em cada um destes modos, não tem o mesmo número. A tabela a seguir mostra isto.

nº	CGAC0	CGAC1	CGAC2	CGAC3
0	preto	preto	preto	preto
1	verde claro	azul claro	verde	azul
2	vermelho claro	rosa	vermelho	p¶rpura
3	amarelo	branco	marron	cinza

Nos modos EGA e VGA, o n¶mero de cores ¶ maior e estao de¯nidos por constantes no arquivo graphics.h. Estas constantes sao: BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW e WHITE.

a.2.6 - Viewports

Viewports proporcionam uma maneira de restringir a ¶rea da tela usada para desenho. Elas se parecem com as windows do modo texto. Pode-se deixar apenas uma parte vis¶vel de uma imagem que ocupa a ¶rea inteira usando-se viewport. A funcao viewport, no entanto, nao ir¶ comprimir a imagem, as partes que nao couberem na viewport ¯carao invis¶veis.

A seguir demostra-se o uso de uma viewport para restringir a visibilidade de uma parte de um gr¶¯co do tipo \pizza".

```
/* gera um gr¶¯co tipo pizza */
# include <graphics.h>
# de¯ne N 6 /* numero de dados */
int dados[N] = {11,19,44,32,15,7}; /* dados */
void main(void)
{ int driver = CGA, mode = CGAC0;
int j, raio = 50, xc = 100, yc = 100;
int left = top = 0, right = botton = 100; /* limites viewport */
int clip = 1; /* liga o clipping da viewport */
° oat soma, angi, angf, angrel;
initgraph(&driver, &mode, "C:ntcnlib");
setviewport(left, top, right, botton, clip); /* estabelece a viewport */
for (j = 0, soma = 0; j < N; j++) soma += dados[j];
/* soma os valores dos dados */
angf = 0; /* angulo ¯nal na posicao do rel¶gio as 3 horas */
for (j = 0; j < N; j++) /* desenha as fatias */
{ angi = angf; /* inicia no ¯m da fatia anterior */
angrel = 360 * (dados[j] / soma); /* calcula o angulo */
angf = angi + angrel; /* ¯m da fatia atual */
set¯llstyle(j % N, 3); /* estabelece estilo de preenchimento e cor amarelo */
pieslice(xc, yc, angi,angj, raio); /* desenha uma fatia */
}
getche();
closegraph();
```

A viewport é criada pela função setviewport ( ). O último parâmetro indica, se igual a zero, que a imagem exterior a viewport não será desenhada (clip).

A função set_llstyle ( ) estabelece o padrão e a cor para preenchimento de uma área. Os padrões possíveis podem ser um dos nomes ou valores definidos em graphics.h.

NOME	VALOR	RESULTADO
EMPTY_FILL	0	mantém o fundo
SOLID_FILL	1	preenche com uma cor
LINE_FILL	2	linhas horizontais
LTSLASH_FILL	3	linhas a 45º finas
SLASH_FILL	4	linhas a 45º grossas
BKSLASH_FILL	5	linhas a - 45º grossas
LTBKSLASH_FILL	6	LINHAS a - 45º finas
HATCH_FILL	7	hachuras
XHATCH_FILL	8	hachuras em \X"
INTERLEAVE_FILL	9	linhas entrelaçadas
WIDE_DOT_FILL	10	pontos espaçados
CLOSE_DOT_FILL	11	pontos próximos
USER_FILL	12	padrão definido pelo usuário

O último valor deve ser criado pelo uso anterior da função set_llpattern ( ).

A função clearview port ( ) apaga a imagem da view port, sem alterar o resto da tela. A função getview pettings ( ) obtém as dimensões e o status de clip da viewport.

a.2.7 - Problemas devido a pixeis não quadrados

Em alguns modos gráficos os pixeis não são quadrados (ou circulos), iguais na direção x e y, e sim retangulares (ou elipticos). A razão entre a altura e a largura é chamada aspect ratio. Pixeis quadrados têm aspect ratio igual a 1. Na criação de imagens, pixeis não quadrados criam certos problemas de distorção nas figuras. Este problema é automaticamente compensado em algumas função do TC como arc( ) e circle ( ). Para fazer esta compensação em seus desenhos use a função getaspectratio ( ) que \retorna" dois números a partir dos quais o aspect ratio pode ser corrigido. O elemento Y é sempre 10 000, dividindo a parte X por Y tem-se o aspect ratio. O protótipo desta função é:

```
void far getaspectratio (endX , endY )
int far * endX  /* endereço da parte X  da razão */
int far * endY  /* endereço da parte Y  da eazão de lados */
```

a.2.7 - Pixeis

Pode-se plotar cada pixel individualmente usando a função putpixel (x , y, cor), que tem como parâmetros as coordenadas x e y do pixel e a cor em que será desenhado. A função cor = getpixel (x , y) pode ser usada para encontrar a cor do pixel na posição x, y.

A imagem de uma área retangular da tela pode ser armazenada na memória. Armazenar uma imagem significa armazenar a cor de cada pixel em determinada ordem. Uma imagem armazenada na memória pode ser reproduzida novamente na tela, na mesma localização ou outra posição.

Um uso importante do armazenamento de imagens é em animação, pois neste caso imagens devem ser desenhadas com rapidez para simular o movimento. Se a imagem a ser \animada" for complexa seu tempo de criação pode ser muito maior que o adequado para simular uma animação. Se a imagem for desenhada, armazenada e apenas restaurada pela animação seu tempo é constante e independente da complexidade do desenho.

A função getimage(left, top, right, botton, endBuf) armazena a imagem da região retangular definida por (left, top) e (right , bottom) no endereço da memória especificado pelo último parâmetro. É necessário antes dispor-se de um endereço para armazenamento da área de memória utilizada pela imagem. O valor de retorno da função imagesize(left, top, right, bottom) é a quantidade de bytes necessários para armazenar a imagem. Este valor de retorno pode ser \passado" para um pedido de alocação dinâmica e o ponteiro resultante então usado como argumento de getimage( )

Uma imagem armazenada pode ser reconstituida pela função putimage(left, top, endBuf, putop). Para esta função é necessário fornecer o canto esquerdo superior da área onde a imagem será reconstituida, o endereço do buffer de memória da imagem e um valor que proporciona diferentes efeitos de acordo com a tabela abaixo.

VALOR	NOME	RESULTADOS
0	COPY_PUT	substitui a imagem anterior da tela pela armazenada
1	XOR_PUT	faz a operação de ou - exclusivo entre as imagens
2	OR_PUT	faz a operação de or entre as imagens
3	AND_PUT	faz a operação and entre as imagens
4	NOT_PUT	copia a imagem inversa da armazenada

O exemplo a seguir cria uma bola e \anima" o seu movimento imitando uma bola quicando em uma caixa.

```
/* bola quicando */
#  include < graphics.lib >
#  define CGAC0_VERDE 1
#  define RAIO  8
void main (void)
f int driver = CGA, mode = CGAC0;
int x, y, dx, dy, xant, yant; /* coordenadas da bola */
void * bolabuf; /* ponteiro para o buffer */
unsigned size; /* tamanho do buffer */
initgraph (& driver, & mode, \C :ntcnlib");
rectangle (0, 0, 319, 199); /* desenha a caixa */
setcolor (CGAC0_VERDE);
x = y = Raio + 10;
cicle (x, y, RAIO ); /* desenha o circulo */
```

```
°ood¯ll(x,y,CGAC0_VERDE);/* preenche seu interior */
size = imagesize(x-RAIO,y-RAIO,y+RAIO);
/* obt¶m o tamanho em bytes da imagem /*
bolabu® = malloc(size) /* obt¶m mem ória para a imagem
getimage(x-RAIO,y-RAIO,x+RAIO,y+RAIO,bolabu®);/* armazena */
dx = 2;dy = 1;/* velocidade da bola *
while(!kbhit( )) /* para com qualquer tecla */
f
putimage(x-RAIO;y-RAIO,bolabu®,COPY_PUT);
xant = x;yant = y;/* armazena coordenadas da tela */
x+ = dx;y+ = dy;/* move posi»ao da bola */
x+ = dx;y+ = dy;/* move posi»ao da bola */
if(x<=RAIO+2::x>=319-RAIO-2)dx = -dx;
/* re°ete nas paredes verticais */
if(y<=RAIO+1::y>=199-RAIO-1)dy = -dy;
/* re°ete nas paredes horizontais */
putimage(xant-RAIO,yant-RAIO,bolabu®,XOR_PUT);
/* apaga a imagem anterior */
gclosegraph( );
g
```

a.2.8 - Textos e Gr¯cos

No modo gr¯co, textos s»ao escritos com fun»c»oes especiais que permitem maior °exibilidade. O estilo do texto a ser escrito ¶ especi¯cado pela fun»c»ao settextstyle(fonte, dire»c»ao, tamanho).

O argumento fonte pode ser algum dos valores ou nomes abaixo

VALOR	NOME	ARQUIVO	RESULTADO	TIPO
0	DEFAULT_FONT	j¶ incorporado	8 x 8	bit_map
1	TIPLEX_FONT	TRIP.CHR	estilo times-roman	stroke
2	SMALL_FONT	LITT.CHR	letras pequenas	stroke
3	SANSSERIF_FONT	SANS.CHR	estilo sans_serif	stroke
4	GOTHIC_FONT	GOTH_CHR	estilo g¶tico	stroke

O argumento dire»c»ao pode ter dois valores, 0 ou 1, ou dois nomes, HORIZ_DIR ou VERT_DIR. Os primeiros 0 ou HORIZ_DIR, indicam que o texto ser¶ escrito da esquerda para a direita. O segundo, 1 ou VERT_DIR que o texto ser¶ escrito de baixo para cima (como se rotacionada de 90^{0}).

O argumento tamanho ¶ um n¶umero inteiro que indica quantas vezes maior ser»ao os caracteres que o menor tamanho de cada fonte.

A fun»c»ao outtext(endstring) escreve o texto na tela na posi»c»ao corrente (CP). Assim para usar esta fun»c»ao ¶ necess¶rio determinar a posi»c»ao corrente, CP, o que pode ser feito pela

função moveto (x, y). A função outtext ( ) escreve o texto usando a fonte, a orientação e o tamanho especi¯cado pela função settextstyle ( ). A função outtextxy ( ) ¶ semelhante a outtext ( ) s¶ que determina a posição xy do texto.

A função settextjusfy (horiz, vert) determina como o texto ser¶ escrito em relação a CP. Os valores ou nomes poss¶veis destes parâmetros s~ao:

	VALOR	NOME	RESULTADO
horiz	0	LEFT_TEXT	a CP ¯ca a esquerda do texto
	1	CENTR_TEXT	a CP ¯ca horizontalmente no centro
	2	RIGHT_TEXT	a CP ¯ca a direita do texto
vert	0	BOTTOM_TEXT	a CP ¯ca abaixo do texto
	1	CENTER_TEXT	a CP ¯ca verticalmente no centro do texto
	2	TOP_TEXT	a CP ¯ca acima do texto

A proporção horizontal e vertical dos caracteres a serem escritos por fontes strokes pode ser modi¯cada pela função setsercharsize (num x, denx, num y, deny). Os parâmetros desta função s~ao os numeradores e denominadores de uma fração que ser¶ multiplicado ao tamanho do caracter, na direção x e y. Quando esta função ¶ usada o parâmetro \tamanho" na função settextstyle ( ) deve receber o valor 0 ou o nome USER_CHAR_SIZE. O exemplo abaixo gera um texto em diversas proporções.

```
/* varia proporções */
# include < graphics >
void main (void)
int driver = DETECT; /* obtem a placa do sistema */
int mode, numx, denx, numy, deny;
initgraph (& driver, & mode, \C:ntcnlib");
settextstyle (TRIPEX_FONT, VERT_DIR, USER_CHAR_SIZE);
outtext (\tamanho normal");
moveto (0, 60);
setusercharsize (1, 1, 3, 2); /* altura 3/2 */
outtext (\tres meios mais alto");
setusercharsize (1, 2, 1, 1); /* largura 1/2 */
outtext (0, 120, \metade da largura");
getche ( );
closegraph ( ); g
```