UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

CONSTRUÇÃO DE UMA BIBLIOTECA DE APOIO GRÁFICO AOS PROGRAMAS APLICATIVOS NA ENGENHARIA

DISSERTAÇÃO SUBMETIDA À UNIVERSIDADE FEDERAL DE SANTA CATARINA
PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA MECÂNICA

José Maria Bezerra Silva

FLORIANÓPOLIS, DEZEMBRO DE 1988

CONSTRUÇÃO DE UMA BIBLIOTECA DE APOIO GRÁFICO AOS PROGRAMAS APLICATIVOS NA ENGENHARIA

JOSÉ MARIA BEZERRA SILVA

ESTA DISSERTAÇÃO FOI JULGADA ADEQUADA PARA OBTENÇÃO DO TÍTULO DE

"MESTRE EM ENGENHARIA"

ESPECIALIDADE ENGENHARIA MECÂNICA, ÁREA DE CONCENTRAÇÃO PROJETO, APROVADA EM SUA FORMA FINAL PELO CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA.

PROF. EDISON DA ROSA, M.Sc.
ORIENTADOR

PROF. ARNO BLASS, Ph.D.
COORDENADOR

BANCA EXAMINADORA:

PROF. NELSON BACK, Ph.D.

PROF. EDISON DA ROSA, M.Sc.

PROF. ABELARDO ALVES DE QUEIROZ, Ph.D.

À Simoni e Maria José.

À minha Mãe.

AGRADECIMENTOS

- Ao Professor Arno Blass, na figura de Coordenador do Curso de Pós-Graduação em Engenharia Mecânica, pela oportunidade;

- Em especial ao Professor Edison da Rosa, pela orientação técnica e pela amizade;

- Aos Professores e Amigos do Departamento de Engenharia Mecânica da UFPE, pelo apoio e incentivo à conclusão deste trabalho;

- Ao Aluno José Rizzo Hahn Filho, pelo auxílio na elaboração do manual de utilização do "zmSAG";

- Ao Aluno Marco Túlio Rodrigues, pela estafante bateria de testes feitos no "zmSAG" com exemplos de programas;

- À minha esposa; Rose Simoni e à Mafalda Cristina Vieira, pelos excelentes trabalhos de processamento de texto e revisão ortográfica;

- A Inácio Martins de Lima Filho, pela excelente qualidade dos desenhos;

- À Comissão Nacional de Energia Nuclear (CNEN), pelo apoio financeiro.

SUMÁRIO

RESUMO

Este trabalho consiste na obtenção de um núcleo gráfico formado por várias funções que irão apoiar o "Engenheiro Programador" no desenvolvimento de programas que necessitem de algum tipo de saída gráfica tais, como geração de malhas para elementos finitos, simulação de processos industriais, análise de interferência em mecanismos articulados e tantos outros que tenham interação gráfica com o usuário final.

O conjunto de funções recebeu o nome de zmSAG (Sistema de Apoio Gráfico - o prefixo "zm" faz referência ao nome do autor) e foi desenvolvido com base na norma GKS, padronizada pela ANSI e adotada pela ABNT desde 1986. Tendo sido totalmente implementado em linguagem "C", este pode ser acessado pelos principais compiladores desta linguagem disponíveis para o sistema operacioal MS-DOS.

ABSTRACT

This work concetrates on the determination of a graphical kernel system to support the "software engineer" on the development of computational programs with graphic output.

Particular examples of such computational programs are: network genertion in finite element method, general system simulation, analysis of linkage mechanism, etc..

The "C" language together with the GKS standard (based on ANSI and adapted by ABNT) are used in the development process of the graphical kernel system.

The set of functions is designated by zmSAG (Sistema de Apoio Gráfico), where "zm" refers to the author's initials, and its access is made by a "C" compiler under MS-DOS system.

CAPÍTULO 1

Introdução e Histórico

1.1 Introdução

Não resta a menor dúvida de que estamos passando por uma transformação radical no que diz respeito ao projeto e fabricação de todos os "produtos industriais", desde automóveis, calçados e até mesmo a mais simples peça de um brinquedo. Dúvida também não se tem de que esta revolução se deve à introdução do computador nestes processos e, em particular às facilidades cada vez maiores de comunicação e iteração entre o homem e este. Estas "facilidades" têm permitido que o trabalho conjunto entre a máquina e o homem se torne cada vez mais rápido e eficiente, e isto se torna possível a partir da utilização de sistemas "user friendly", que introduzem um novo conceito na utilização de computadores, permitindo que o homem se comunique com a máquina da forma mais facilitada e simples possível.

Estes princípios também têm permitido que os sistemas de CAD (*Computer Aided Design*) sejam cada vez mais difundidos nos projetos de aplicação industrial, trazendo como principais vantagens a iteratividade e facilidade de concepção de novos projetos, a possibilidade de simular o comportamento do sistema antes da construção do protótipo - modificando, caso seja necessário, seus parâmetros -, a geração de planos com detalhes sobre todas as vistas e secções de uma peça, além da possibilidade de conexão com um sistema de fabricação, gerenciada pelo computador, para a confecção automática do produto.

Dada a complexidade destes sistemas, a sua construção exige, de uma forma geral, o empenho de profissionais não só da área de informática mas, principalmente, de engenheiros e técnicos especialistas no assunto. Um dos aspectos que tornam um sistema CAD "user friendly" é a facilidade de iteração através de imagens gráficas, estas imagens são geradas a partir de bibliotecas gráficas padrões que, devido a especificidade do assunto, são geralmente construídas por profissionais da área técnica e com conhecimento detalhado acerca do "hardware".

A aceitação e utilização, pelo engenheiro, destes conceitos e técnicas, o tem levado a procurar introduzí-los em seus novos projetos. Desafortunadamente,

mesmo nos "mainframes", as ferramentas disponíveis ao desenvolvimento de programas científicos não prevêem ainda uma interface do tipo "user friendly" a nível de desenvolvimento nem a nível de programa final, principalmente no que diz respeito ao processamento das imagens gráficas, isto tem levado cada vez mais os próprios engenheiros a procurarem construir bibliotecas contendo rotinas de apoio dos mais diversos tipos, porém específicas a um determinado problema de engenharia. Muitas destas bibliotecas, desde que seguindo certas normas, poderiam ser extendidas a uma utilização mais geral, em específico pode ser citado o caso de subrotinas gráficas onde já existem várias normas tais como GKS, PHIGS, etc. e em cada "hardware" uma maneira diferente de se implementar. A proposta básica do "zmSAG" (Sistema de Apoio Gráfico) se firma na implementação de um conjunto de subrotinas gráficas baseadas na utilização da norma GKS e implantação em "hardware" do tipo PC-XT ou supermicros à base do sistema operacional UNIX, com o propósito de dar suporte aos engenheiros e projetistas que destes equipamentos se utilizem a partir das linguagens *C*, *Pascal* e *Fortran*.

1.2 Suporte Gráfico

As imagens gráficas são bem exploradas como efetivos meios de comunicação. Alguns estudos [14] indicam que uma simples mensagem que se utilize de meios gráficos é conscienciosamente reconhecida mais rapidamente que utilizando-se apenas de texto. A "publicidade informatizada" nos últimos anos tem explorado este meio de comunicação muito mais que qualquer outro segmento. Para isto, tem-se desenvolvido um "hardware" cada vez mais sofisticado e direcionado ao processamento gráfico de imagens. Contudo, o desenvolvimento do "software" não se faz tão rápido quanto o do "hardware", existindo na prática uma defazagem enorme deste para os modernos equipamentos de informática hoje existentes, e isto se faz notório quando se trata de equipamentos e ferramental destinados ao processamento de imagens gráficas.

Quando se trata de aplicativos destinados a auxiliar nos projetos de engenharia, os pesquisadores e projetistas brasileiros dispõem apenas de uns poucos "softwares" científicos que se utilizam de recursos gráficos, sendo estes na sua quase totalidade produtos importados que, via de regra, se constituem "caixas pretas" tendo os seguintes inconvenientes:

. não se adequam totalmente às necessidades da emprêsa;

. total inexistência de manutenção e suporte de apoio ao usuário;

. incompatibilidade de interfaceamento com os periféricos nacionais;

. custo elevado.

Em parte, o motivo desta "escassez" se deve às várias dificuldades encontradas no desenvolvimento de "softwares" gráficos no Brasil, entre as quais podem-se destacar duas como sendo as maiores em termos de projeto e desenvolvimento.

Em primeiro lugar, verifica-se o simples fato de concretamente ainda não existirem metodologias bem definidas para produção de "softwares" de características científicas, como é o caso dos aplicativos gráficos. Isto leva cada projetista a desenvolver uma metodologia própria de construção, incompatibilizando-se com os demais.

Em segundo lugar, vem um problema que é comum a todos os projetos que envolvem tecnologia de ponta, que consiste basicamente em se alterar uma tendência anteriormente seguida por uma determinada comunidade. No caso, a tendência anterior era a linguagem FORTRAN, muito utilizada por cientistas do mundo inteiro e com grande quantidade de rotinas nela escritas. Procurando-se mudar para algo mais novo [21], a linguagem "C" apresenta-se como uma excelente opção, por se tratar de uma linguagem estruturada, ter um alto nível de portabilidade, apresentar uma alocação dinâmica de memória e permitir que se trabalhe ao nível da máquina em alguns casos, o que torna mais fácil a programação de diversos periféricos, o que é comum em computação gráfica.

O "zmSAG" (Sistema de Apoio Gráfico) é uma experiência rara no país (no meio acadêmico) para a produção de sistemas gráficos. Tem uma tecnologia totalmente aberta, incluindo os fontes, acessíveis a todas as pessoas (do meio acadêmico) que se interessarem por CAD e Computação Gráfica. O objetivo desta experiência é o de permitir que usuários, programadores e projetistas desenvolvam sistemas gráficos, difundindo, assim, a cultura da *computação gráfica* no país.

1.3 Histórico

Durante muito tempo [6] a comunicação entre o homem e o computador foi realizada através de sequências intermináveis de números e letras, só possíveis de serem manipuladas por especialistas. Faltava ao computador a capacidade de apresentar e receber imagens.

O desenvolvimento tecnológico que permitiu o advento dos compactos processadores lógicos (microprocessadores), dos computadores pessoais e de chips de memória de baixo custo industrial, também permitiu [14] que o processamento de imagens gráficas começasse a formar parte integrante dos sistemas computacionais; convertendo dados numéricos em "mapas" gráficos, combinando textos e desenhos, trocando tipos e estilos, e atc.. Ainda assim, durante anos a utilização da computação gráfica, como efetivo meio de produção, esteve limitada aos sistemas sofisticados e, devido ao alto custo destes sistemas, em termos de requisição de memória e capacidade de processamento, limitava-se apenas ao controle de tráfego aéreo ou de tarefas nas quais as imagens gráficas incrementavam significativamente a performance em razão do custo.

Este panorama começou a se modificar significativamente quando, em 1962, Ivan E. Sutherland apresentou a sua tese de doutoramento no MIT: *Sketchpad: a man-machine graphical communication system*, o que lhe valeu o título de "pai da moderna Computação Gráfica". Na década de 70, a tecnologia de fabricação dos displays para uso gráfico e o desenvolvimento significativo da microeletrônica e programas de processamento gráfico, tornaram a Computação Gráfica um recurso viável economicamente para uma variedade muito grande de aplicações nas diversas áreas, como por exemplo, medicina, administração, engenharia e arte entre outras.

Porém, o grande salto tecnológico nesta área, foi dado mesmo com o surgimento da *Computação Gráfica Interativa* [7], que permite ao usuário interferir diretamente no desenvolvimento de um desenho e visualizar o resultado imediatamente. A própria imagem passou a ser o objeto principal de aplicativos voltados para a documentação, criação de logotipos, desenvolvimento de projetos, simulação, e outros. A área de informática que maior proveito vem tirando da computação gráfica interativa é a de CAD/CAM (*Computer Aided Design/Computer Aided Manufacturing*), que tem ligação direta com os projetos de engenharia de um

modo geral.

No Brasil [17], os primeiros trabalhos de Computação Gráfica, ainda em caráter exploratório e acadêmico, foram desenvolvidos no final da década de 60, tendo as primeiras iniciativas em termos de empresa ocorrido na primeira metade da década de 70. Nestes casos, geralmente o que se pretendia era a obtenção de representação gráfica de programas de cálculo em engenharia, nas suas diversas modalidades.

Em termos de associações voltadas para o assunto, ainda não foi criada oficialmente nenhuma, embora existam alguns trabalhos desenvolvidos por intermédio da SUCESU, através da qual foram organizados vários simpósios específicos sobre o assunto. Destacando-se, em 1981, uma conferência internacional sobre CAD/CAM, em que a Computação Gráfica foi abordada como tecnologia necessária para esse tipo de aplicação.

No final de 1985, a ABNT formou uma comissão de estudos de "Padronização em Computação Gráfica e CAD/CAM" [17], com a missão de preocupar-se com a Computação Gráfica como ferramenta de apoio para o desenvolvimento de aplicativos, estudando itens como GKS, CGM, CGI, PHIGS etc.

Atualmente, esta comissão tem o nome de "Comissão Técnica de Computação Gráfica", e trabalha na tradução da norma GKS para o português, a fim de tornar o GKS padrão brasileiro na área.

CAPÍTULO 2

Padrões Gráficos e o GKS

2.1 Introdução

O processamento das imagens gráficas, com a utilização do computador, envolve, basicamente, o núcleo de processamento (CPU), os dispositivos periféricos e um programa de aplicação que fará o controle sobre todo o sistema. Os primeiros sistemas dedicados à Computação Gráfica, por não adotarem ainda uma padronização, tinham geração de imagens totalmente dependente do conhecimento que cada usuário/programador tivesse de todos os periféricos de entrada e saída gráfica ligados aos mesmos.

Com o crescimento da demanda de aplicações gráficas, exigiu-se uma solução com a qual os usuários ficassem livres da necessidade de conhecer em detalhes cada dispositivo. Esta idéia proporcionou o surgimento de programas especiais, chamados "pacotes gráficos", que forneciam uma interface, a nível de usuário, de forma a permitir uma interação mais amigável com os vários dispositivos periféricos.

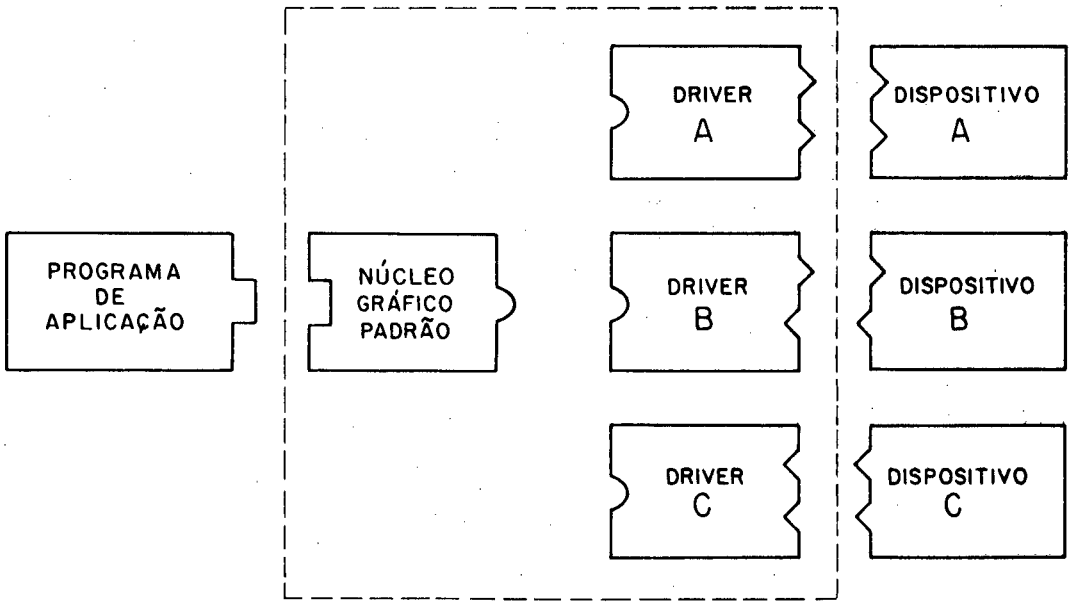


FIGURA 2.1 - Independência do "pacote gráfico" em relação aos dispositivos periféricos.

Na decáda de 70 surgiram comercialmente os primeiros "pacotes gráficos", chamados pelos fabricantes de *turn-key systems*, que em geral, se destinavam a auxiliar nos projetos de engenharia.

Este conceito evoluiu a tal ponto, que nos dias de hoje o "pacote gráfico" tornou-se independente dos tipos de dispositivos e até da linguagem de programação, oferecendo um conjunto de funções e primitivas padronizadas a nível de usuário (figura 2.1).

2.2 Busca de Padronização

Este esforço de padronização envolveu um longo período de tempo e grupos de trabalho. Em 1969 a ACM (Association for Computing Machinery) criou [17,19] o SIGGRAPH (Special Interest Group on Computer Graphics), que por sua vez criou, em 1974, o GSPC (Graphics Standards Planning Committee). Também houve esforços de vários outros comitês de normalização, destacando-se a ANSI, DIN, ISO como os mais importantes. Dos resultados alcançados por esses grupos, destacam-se dois deles:

- O GSPC, com o desenvolvimento de duas versões da especificação do sistema CORE, uma vez em 1977 e outra em 1979, formando a última delas um núcleo gráfico com funções tridimensionais.
- O grupo do Instituto Alemão de Padronização (DIN), com a produção de várias versões do GKS (Graphical Kernel System), sistema bidimensional com um conjunto de funções mais restrito que o CORE.

Após vários estudos em cima das propostas GKS e CORE para uma padronização internacional, a ISO decidiu, em 1979, concentrar esforços no GKS e, ao final de 1982, a versão 7.2 do GKS, depois de ter passado por todos os grupos técnicos da ISO, foi transformada em *Draft International Standard* (DIS 7942).

Em junho de 1985, a ANSI reconhece o GKS com o padrão norte-americano em Computação Gráfica.

Finalmente os marcos mais importantes, nesta busca pela padronização, estão resumidos no quadro 2.1 a seguir:

DATA	MARCO
1974	formado o ACM-SIGGRAPH/GSPC
1975	fundado o DIN NI/UA 5.9
1976	workshop SEILLAC I, na França
1977	formado o ISO TC97/SC5/WG2 GSPC/CORE publicado GKS 1.0 publicado
1979	GSPC/CORE revisado formado o comitê X3H3 da ANSI governo canadense aprova o TELIDON início de trabalhos sobre VDI e VDM
Out./1980	GKS 6.2 torna-se item de trabalho na ISO
1981	NAPLPS anunciado pela AT&T
Out./1982	GKS 7.0 torna-se proposta de draft
1982	NAPLPS recebe apoio industrial
Maio/1982	GKS 7.0 torna-se draft de padrão internacional (ISO DIS 7942)
Jun./1982	GKS 7.0 é revisado para versão 7.2
Jun./1985	GKS é aprovado padrão americano pela ANSI
Jul./1985	GKS é aprovado padrão internacional pela ISO
1986	CGM é aprovado como padrão americano pela ANSI e internacional pela ISO

QUADRO 2.1 - Marcos importantes na busca da padronização

Após vencida esta etapa de padronização, o principal objetivo das entidades internacionais concentra-se, agora na definição de procedimentos e certificação, para verificar se as implementações comerciais do GKS estão realmente seguindo as definições normalizadas.

2.3 Padrões Importantes

Embora seja o GKS aceito internacionalmente como interface gráfica padrão para programas aplicativos, existem outros padrões em Computação Gráfica e CAD já adotados [17], ou que se encontram atualmente em fase de proposta de estudos.

São eles:

- GKS - Graphical Kernel System

- Interface gráfica padrão para programas aplicativos:

- Adotada como norma americana pela ANSI (American National Standards Institute) em 1985;

- Adotada como norma internacional pela ISO (International Organization for Standards) em 1985.

- GKS-3D - Graphical Kernel System for Three Dimensions:

- Extensão tridimensional do GKS;

- Proposta de norma internacional em estudos.

- CGI - Computer Graphics Interface:

- Especificação de técnicas de interfaceamento para uso padronizado de dispositivos gráficos;

- Objetos de estudos no âmbito da ISO e ANSI.

- CGM - Computer Graphics Metafile:

- Especificação de um mecanismo de transferência e armazenamento de informações de descrição de figuras;

- Adotado como padrão pela ANSI e ISO.

- PHIGS - Programmer's Hierarchical Interactive Graphics System:

- Interface gráfica bi e tridimensional para programas aplicativos;

- Objeto de estudo em desenvolvimento na ISO e ANSI.

- IGES - Initial Graphics Exchange Standard:

- Especificação de formatos de arquivos e linguagem para transmissão e armazenamento de dados gráficos;

- Sua versão 1.0 faz parte da norma ANSI de 1981.

- NAPLS - North American Presentation - Level Protocol Syntax:

- Interface de transmissão de texto e gráficos para dispositivos de exibição ou armazenamento com inteligência local;

- Norma ANSI de 1983.

Todos estes padrões são de interface e podem ser divididos quanto ao tipo de interface envolvida no padrão:

- Interface de dados: padrões de interface de dados especificam uma forma de representação de dados gráficos para transmissão a um dispositivo gráfico, ou armazenamento em arquivo de dados. Nesta classe, tem-se os padrões IGES, CGM e NAPLS.
- Interface de sub-rotinas: padrões de interface de sub-rotinas definem o comportamento de um sistema ou conjunto de rotinas destinadas à manipulação de dados gráficos. Neste grupo encontram-se os padrões GKS, GKS-3D, CGI e PHIGS.

2.4 O Padrão GKS

De forma a simplificar as operações ao nível do usuário, a norma GKS estabelece um modelo funcional através da definição de um conjunto de funções (vide apêndice A), para aplicações gráficas e, do ponto de vista de acesso a periféricos, fornece uma visão uniforme com isolamento das características das ET's (Estações de Trabalho) através de uma descrição funcional. (A figura 2.2 ilustra bem o modelo GKS).

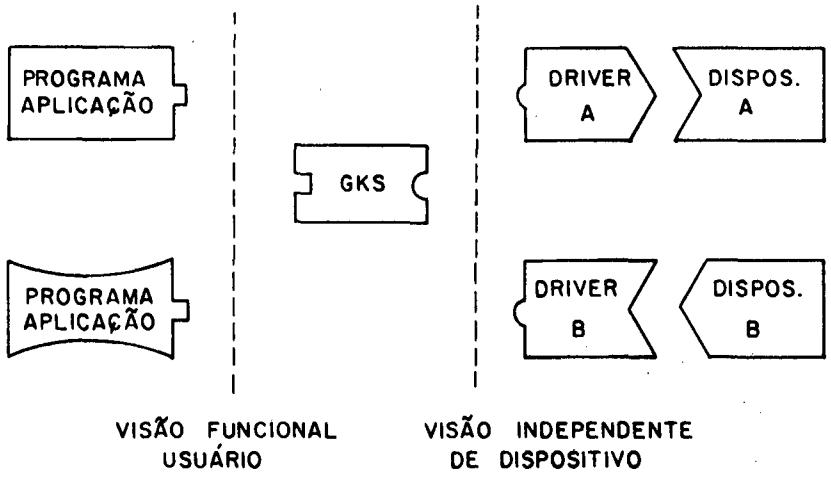


FIGURA 2.2 - Pacote gráfico GKS, visão funcional

Algumas das características do modelo funcional oferecido pelo GKS são descritas, como sub-itens a seguir.

2.4.1 Modelamento de Entrada de Dados

O GKS considera os dispositivos de entrada como unidades lógicas compostas por:

- medida
- disparo

- dependentes da unidade lógica

- valor inicial
- tipo de prompt/eco
- área de eco
- registro de dados

- dependentes do programa de aplicação

São definidos ainda três modos de aplicação:

- request (entrada por demanda)
- sample (entrada por amostragem)
- event (entrada por interrupção)

2.4.2 Estados de Operação

Os estados de operação do GKS estão inter-relacionados seqüencialmente (figura 2.3) e determinam parâmetros de uso pelas funções neles permitidas. Nesta hierarquia não é permitida a ativação de um estado sem que o seu antecedente esteja ativo.

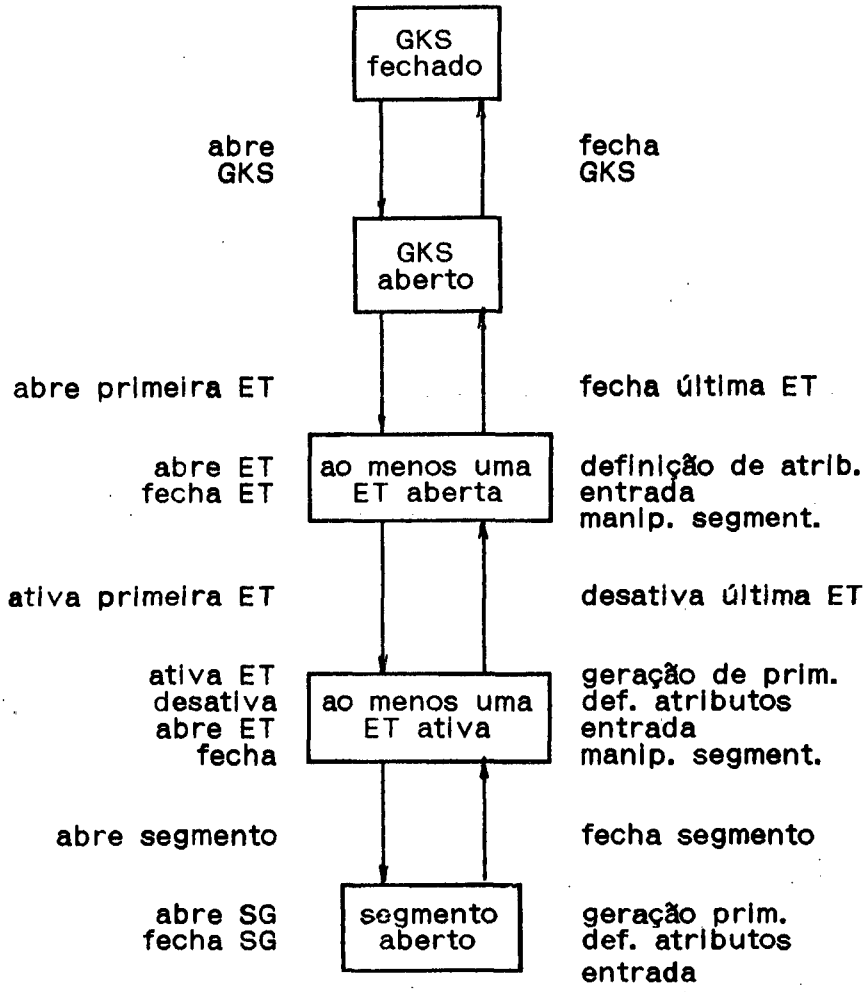


FIGURA 2.3 - Estados de operação do GKS

2.4.3 Sistemas de Coordenadas

São definidos três sistemas de coordenadas, os quais podem ser inter-relacionados através de matrizes de transfomação convenientes. Dentro do GKS existem funções específicas de construção e manipulação destas matrizes, são eles:

- coordenadas do mundo real (WC)
- coordenadas normalizadas (NDC)
- coordenadas do dispositivo (DC)

Quando utilizando o "mundo real", o usuário é quem define os seus limites de trabalho, não se preocupando com as limitações da máquina ou dos periféricos. As coordenadas normalizadas servem como um padrão de interfaceamento entre o

mundo real e os diversos periféricos, uma vez que os valores máximos para coordenadas destes dispositivos sao pré-definidos e limitados, neste contexto tem-se variações de zero a um em x e y para as coordenadas normalizadas. Finalmente o sistema de coordenadas do dispositivo representa as dimensões físicas reais que realmente se pode obter como meio final de observação. Note-se contudo, que o usuário, se assim o desejar, não prcisa fazer uso dos sistemas normalizado e do dispositivo, ficando estas transformações a encargo do GKS.

2.4.4 Estações de Trabalho

Dentro do GKS a estação de trabalho tem um conceito abstrato, de forma a que o usuário possa enxergar qualquer dispositivo gráfico, através da mesma, de maneira uniforme.

Assim, uma ET é um dispositivo lógico de E/S com até uma superfície de visualização retangular e zero ou mais dispositivos lógicos de entrada.

São também consideradas ET's quaisquer arquivos lógicos do GKS, isto é, arquivos que contenham dados gráficos ou numéricos/alfanuméricos.

2.4.5 Segmentos

Segmentos são conceituados como sendo um agrupamento de primitivas de saída sob um nome. As seguintes operações podem ser aplicadas sobre segmentos:

- transformação
- visibilidade
- cintilamento
- priorização
- detectabilidade
- deleção
- renomeação
- inserção

Segmentos podem ser armazenados em GKS (WISS - Workstation Independent Segment Storage) e nas ET's (WDSS - Workstation Dependent Segment Storage).

2.4.6 Metafile

Metafiles são arquivos de entrada/saída com informações gráficas, que o GKS provê ao usuário. Metafiles de saída armazenam funções GKS e itens do usuário. Metafiles de entrada terão funções que deverão ser interpretadas pelo GKS e itens do usuário a serem interpretados pelo programa de aplicação.

2.5 Funções de Implementação

A descrição vista nos sub-itens acima é implementada, no GKS, através de um conjunto de funções que, de forma genérica, podem ser agrupadas como a seguir:

- funções de controle
- funções de saída
- funções de definição de atributos
- funções de transformação
- funções de segmento
- funções de entrada
- funções de metafile
- funções de pergunta (inquire)
- funções de utilidade
- funções de tratamento de erro

Todas as funções, contidas em cada grupo, estão descritas suscintamente no apêndice A.

CAPÍTULO 3

Visualização e Recorte

3.1 Sistemas de Coordenadas

É mais cômodo ao usuário elaborar o sistema de coordenadas no qual vai trabalhar, ou seja, definir os valores mínimos e máximos (X_{min} e X_{max}) sem se preocupar com a apresentação final no dispositivo gráfico de saída. Este sistema, definido pelo usuário, será chamado de sistema do usuário ou coordenadas globais [1,17] (WC - World Coordinates). Através de uma transformação de coordenadas, denominada "transformação de normalização", cada ponto deste sistema é levado a um outro sistema *virtual*, o qual será chamado "sistema normalizado" (NDC - Normalized Device Coordinate). Este sistema tem a finalidade de simplificar as operações de "janelamento", uma vez que o usuário deixará de se preocupar com as dimensões dos diversos dispositivos de saída existentes na estação de trabalho em uso.

Um outro sistema, com o qual também se trabalha é o utilizado pelo dispositivo gráfico de saída [1,17] (DC - Device Coordinate). Neste sistema tem-se restrições quanto ao tamanho do intervalo de cada eixo coordenado e fica-se restrito a uma determinada área de trabalho. A transformação que leva um ponto do NDC à este sistema é chamada *transformação de estação de trabalho*.

A imagem final, vista no dispositivo de saída, será constituída a partir de dois mapeamentos descritos acima. Em cada mapeamento, a porção de desenho que se vai mapear é chamada *janela*, e a porção mapeada "viewport". A figura 3.1 ilustra o processo de forma completa; Uma proção de *janela* no WC (coordenadas globais - sistema em que o usuário trabalha) é mapeada no NDC (sistema normalizado), para só então, ser levada ao dispositivo físico (DC). A figura ilustra também, apenas a título de curiosidade, a transformação do "WC" para o "DC". Ressalve-se que isto não é possível nas implementações usuais.

Apesar de existirem, na realidade, dois mapeamentos distintos, no restante deste capítulo, para efeito de simplificação, não será mencionado um ou outro quando da abordagem dos itens *mapeamento*, *pré-mapeamento* e *recorte*, apenas entenda-se que a teoria aqui exposta aplica-se aos dois casos.

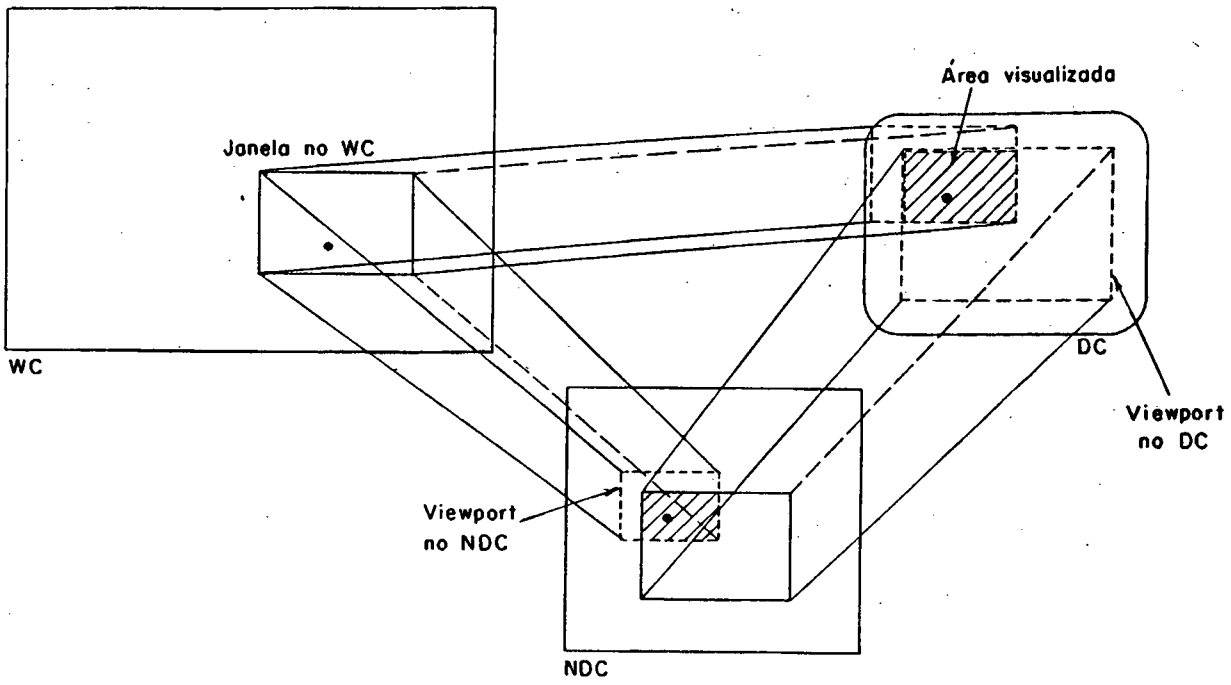


FIGURA 3.1 - Janelas e "viewportes" nos sistemas (WC), (NDC) e (DC).

3.1.1 Pré mapeamento

Uma vez definida a janela de desenho e sua respectiva viewport (janela na tela), o mapeamento caracteriza-se por uma simples transformação linear da janela na "viewport", note-se, entretanto que se os dois retângulos não forem semelhantes, a figura ficará distorcida na tela.

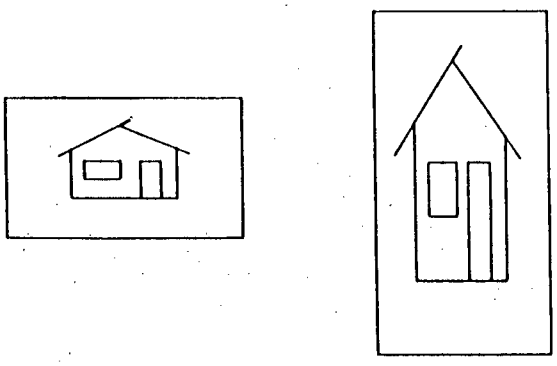


FIGURA 3.2 - Exemplo de distorção quando os dois retângulos não são semelhantes.

Isto pode ser evitado forçando-se a semelhança através da alteração nas dimensões de um dos retângulos. Sendo a "viewport" normalmente predefinida, esta mudança deverá ser feita na janela de desenho.

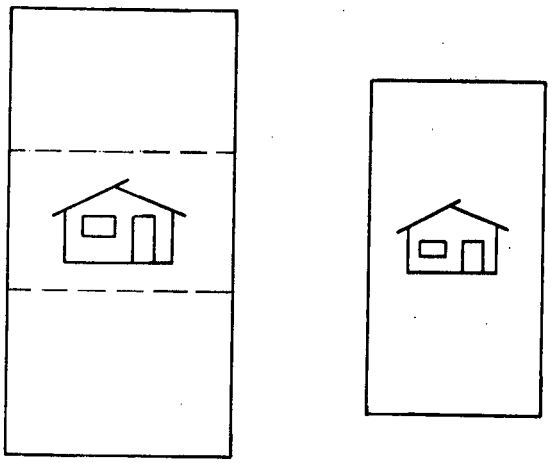


FIGURA 3.3 - Semelhança forçada pela alteração na altura da janela de desenho.

Este procedimento será chamado de *Pré-mapeamento* e sua rotina de *pre-map*. Com "jn_x", "jn_y", "vp_x" e "vp_y" representando a base e a altura da janela e "viewport" respectivamente, esta rotina irá ajustar um dos dois tipos de casos possíveis:

$$\frac{jn_x}{jn_y} < \frac{vp_x}{vp_y} \tag{3.1}$$

ou:

$$\frac{jn_x}{jn_y} > \frac{vp_x}{vp_y} \tag{3.2}$$

Supondo-se o ajustamento para o segundo caso (veja a figura 3.4 abaixo). O problema consiste em se encontrar um novo valor jn_y' para o qual a relação

$$\frac{jn_x}{jn_y'} = \frac{vp_x}{vp_y} \tag{3.3}$$

seja satisfeita. De onde se conclui que:

$$jn_y' = jn_x \cdot \frac{vp_y}{vp_x} \tag{3.4}$$

e os novos valores para y'_{min} e y'_{max} passam a ser:

$$y'_{min} = \frac{y_{min} + y_{max}}{2} - \frac{jn_y'}{2} \tag{3.5}$$

$$y'_{max} = \frac{y_{min} + y_{max}}{2} + \frac{jn_y'}{2} \tag{3.6}$$

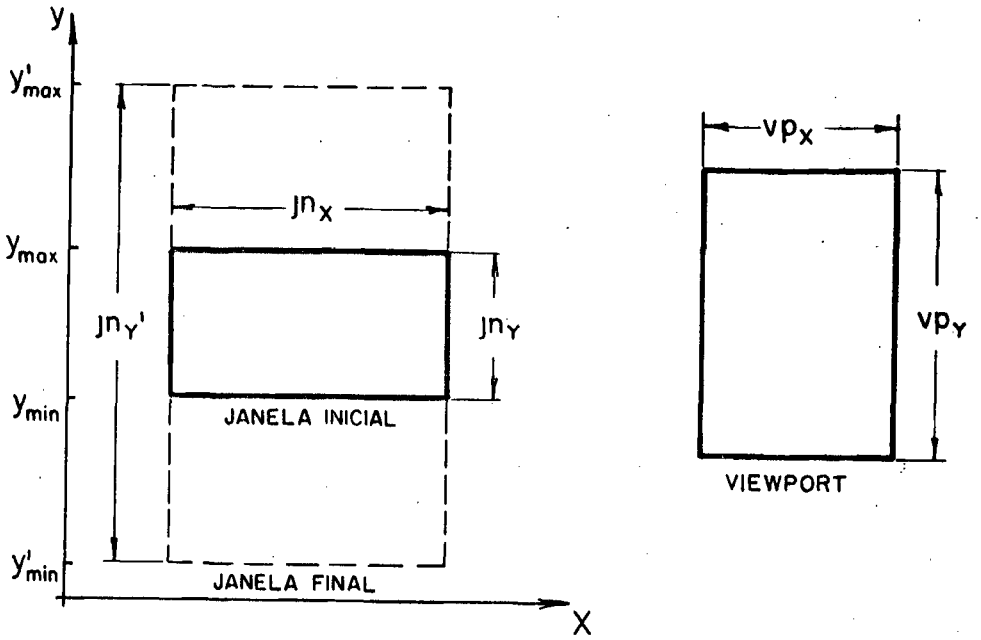


FIGURA 3.4 - Obtenção das novas coordenadas y'_{min} e y'_{max} da janela de desenho.

Note que as figuras aparecerão no dispositivo de saida numa escala proporcional ao desenho na vertical e horizontal. Um problema que poderá surgir será a sobra de espaços acima e abaixo (figura 3.3) ou dos lados, conforme seja o caso. Isto pode ser evitado pelo programador, procurando definir a "viewport" o mais semelhante possível com a janela do desenho.

3.1.2 Mapeamento

Após pré-mapeada a janela na "viewport", deverá ser chamada uma rotina de mapeamento (*set_map*) que determinará a matriz de transformação, a qual deverá mapear um ponto $P(x_d,y_d)$ da *janela* no seu correspondente $P'(x_v,y_v)$ na "viewport" (figura 3.5).

por geometria obtém-se:

$$x_v = xv_{min} + sx\,(x_d - xd_{min}) \tag{3.7}$$

$$y_v = yv_{min} + sy\,(y_d - yd_{min}) \tag{3.8}$$

sendo:

$$sx = \frac{xv_{max} - xv_{min}}{xd_{max} - xd_{min}} \tag{3.9}$$

e:

$$sy = \frac{yv_{max} - yv_{min}}{yd_{max} - yd_{min}} \tag{3.10}$$

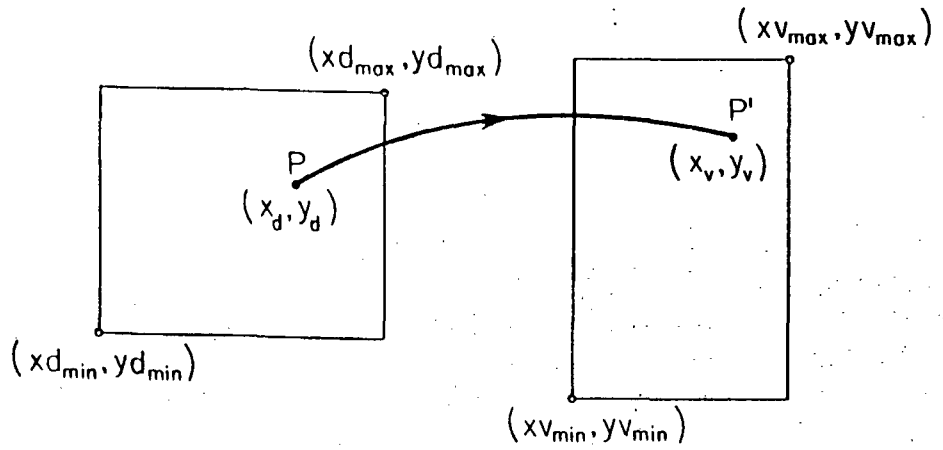


FIGURA 3.5 - Transformação da *janela* na "viewport".

3.2 Recorte

Para se apresentar um desenho ou parte dele no dispositivo de saída, o sistema deve tomar a precaução de verificar as porções visíveis a serem enviadas ao dispositivo, e as partes invisíveis que devem ser descartadas. Este procedimento é denominado *recorte*.

O processo de recorte, aqui, é efetuado em função de uma área retangular [4,7,17]. Toda a parte do desenho situada no interior do retângulo pode ser enviada ao dispositivo de saída; a parte que recair fora desta área retangular será descartada.

O processo de recorte é feito com base na área definida pela janela de desenho. Assim, todo o recorte é efetuado no sistema de coordenadas do usuário, e os dispositivos de saída recebem os valores corretos, prontos para serem visualizados.

3.2.1 Recorte de pontos

O recorte de pontos fora de uma janela especificada é simples e fácil de implementar; basta a aplicação de operações do tipo "min-max" para as coordenadas do ponto como a seguir:

```
se ((x <= janela.x_min) ou
    (x >= janela.x_max) ou
    (y <= janela.y_min) ou
    (y >= janela.y_max))
    entao
        descarte.
fim se
```

3.2.2 Primitivas com "bounding boxes"

Quando se define, para as primitivas, um retângulo circunscrito [21] "bounding box", este retângulo é comparado com a janela para se determinar se o mesmo está totalmente dentro (caso em que não há necessidade de recorte), totalmente fora, ou intercepta a janela (caso em que a primitiva é parcialmente mostrada na tela). A figura 3.6 fornece alguns exemplos. Note que apenas no caso

da elípse se faria necessário o acionamento de uma subrotina de recorte.

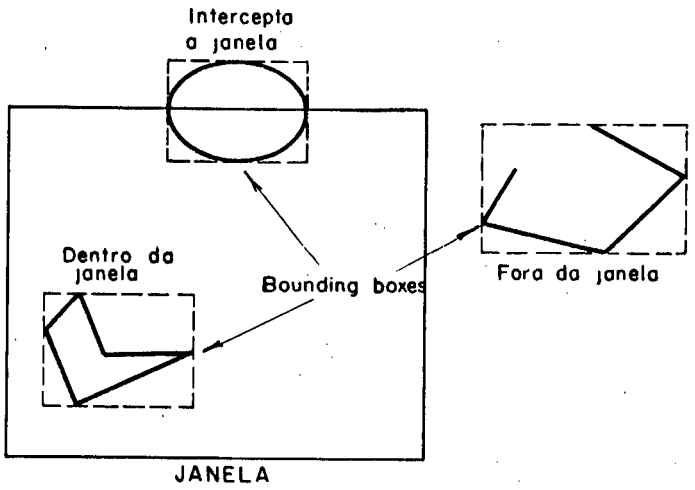


FIGURA 3.6 - Recorte de primitivas com "bounding boxes".

3.2.3 Recorte de linhas

O recorte de linhas é mais complexo do que o de pontos ou primitivas com "bounding boxes". Este último quando houver necessidade de ser recortado, utiliza-se do recorte de linhas para seccionar cada sub-elemento que contenha algum segmento de reta.

Um algoritmo muito simples e engenhoso foi desenvolvido por Dan Cohen e Ivan Sutherland [1,4] para o caso de janelas retangulares. Nele, o plano de traçado é dividido em nove zonas, tendo cada uma um código binário associado, de significado evidente, figura 3.7

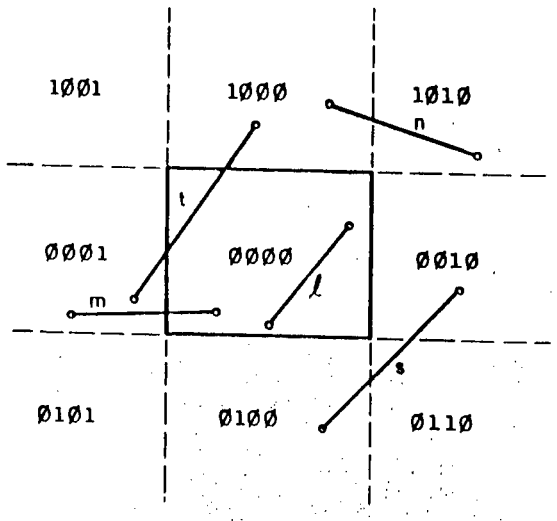


FIGURA 3.7 - Regiões definidas sobre a janela.

Esta divisão facilita a obtenção da posição relativa de um segmento de reta, verificando-se os "códigos de recorte" de cada uma de suas extremidades. O código de recorte utiliza-se de quatro bits, com o seguinte significado:

ligado		significa que
bit 1	-	O ponto situa-se à esquerda da borda esquerda
bit 2	-	O ponto está à direita da borda direita
bit 3	-	O ponto está abaixo da borda inferior da janela
bit 4	-	O ponto está acima da borda superior

Exemplificando, um ponto que se situa abaixo e à direita da janela teria um código de recorte 0110. A figura 3.7 mostra o código para cada região.

A posição relativa de um segmento de reta, com extremidades *i* e *j*, é verificada da seguinte maneira:

1. Segmento de reta todo dentro da janela. Código da extrem. *i* = código da extrem. *j* = 0

2. Segmento de reta com um ponto dentro e outro fora da janela. Um código será igual à zero e o outro diferente de zero.

3. Segmento de reta com ambas as extremidades fora da janela

3a. Segmento de reta totalmente fora. O resultado de um "and", lógico bit a bit, do código da extrem. *i* com o código da extrem. *j* é diferente de zero.

$$(\text{cód. da ext. } i) \text{ and } (\text{cód. da ext. } j) \neq 0$$

3b. Nada se pode concluir. Quando nenhuma das condições anteriores for verificada, o segmento poderá ou não, ter uma parte dentro da janela (segmentos *s* e *t* da figura_ 3.7). Neste caso têm-se:

(cód. da ext. *i*) and (cód. da ext. *j*) = 0

sendo o código de cada extremidade diferente de zero.

Na implementação deste algorítmo o teste a um determinado segmento, depois de identificados os códigos de recorte de cada um dos extremos, segue os critérios acima enunciados. Assim:

- Se a condição 1 é verificada, o segmento é imediatamente trocado;
- Se a condição 1 não se verifica, testa-se a condição 3a. Se esta for verdadeira o segmento é rejeitado.
- Se nenhuma das condições anteriores for verificada, testa-se a 2a. Se esta for verdadeira, é determinada a intersecção do segmento com a borda da janela e traçada a parte interior à janela.
- Não sendo verdadeiros nenhum dos casos anteriores, estamos no caso 3b. Será necessário então, verificar se o segmento intercepta ou não a janela, e em caso afirmativo, determinar as respectivas intersecções e traçar a parte interior.

Note que este algoritmo segue um processo iterativo, e em cada passo da iteração analisam-se os valores dos códigos para detectar o momento de parada do processo. Por exemplo, no caso mostrado na figura 3.8; em uma primeira iteração o algoritmo recorta o segmento AB no ponto C. Analisando os valores dos códigos associados a estes pontos, o procedimento conclui que o segmento AC deve ser descartado. Na próxima iteração, o segmento de reta CB é novamente recortado no ponto D, e pela análise dos códigos conclui-se que, CD deve ser descartado, restando o segmento DB como resultado do processamento de corte.

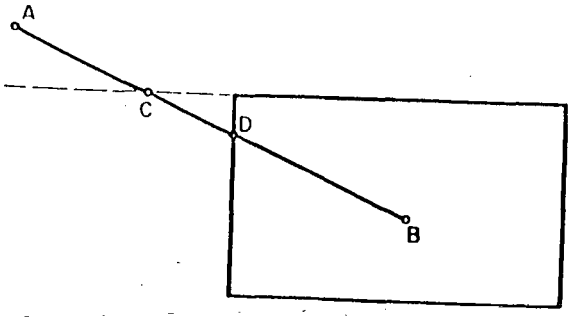


FIGURA 3.8 - Exemplo de recorte do segmento de reta AB, resultando no segmento de reta DB.

Na determinação das intersecções dos segmentos com o contorno da janela têm-se quatro casos distintos que é função do código da extremidade que se encontra fora. Supondo o caso do segmento AB da figura 3.8, aqui tem-se o quarto bit do código da extremidade "A" ligado. O cálculo do ponto de intersecção (x,y) com a janela, (figura 3.9) é fácil de ser efetuado através de geometria elementar.

tem-se:

$$\frac{a}{c} = \frac{b}{d} \tag{3.11}$$

o que equivale a:

$$\frac{x_B - x_A}{x_C - x_A} = \frac{-(y_B - y_A)}{-(y_C - y_A)} \tag{3.12}$$

Notando-se que $y_C = y_{max}$ (coord. y da borda superior) pode-se concluir:

$$x_C = x_A + (x_B - x_A) \cdot \frac{y_{max} - y_A}{y_B - y_A} \tag{3.13}$$

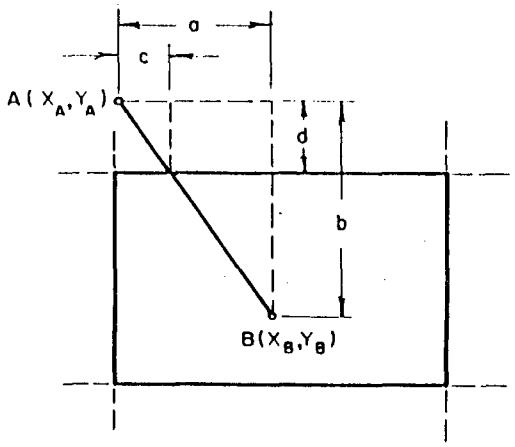


FIGURA 3.9 - Processo geométrico para a obtenção das coordenadas x_C,y_C.

CAPÍTULO 4

Estruturação dos Dados

4.1 Introdução

Um dos fatores mais importantes em um programa computacional é, sem dúvida, a estrutura de dados a ele associada [24]. Para uma aplicação específica como a computação gráfica, os dados devem ser organizados de forma a se conseguir uma performance o mais otimizada possível no que diz respeito à velocidade de manipulação destes dados pelo computador. A este respeito, muitos pacotes gráficos (como recomenda a norma GKS) oferecem recursos de *segmentação gráfica*. São recursos bastante úteis, com os quais o usuário pode criar um gabarito como conjunto de primitivas gráficas, e manipulá-lo como um todo (vide capítulo 6). Imaginemos, como um exemplo, que o sistema represente uma parte mecânica de um carro (um amortecedor). Esta peça, formada por retas e círculos pode ser encarada como um todo, e o sistema poderá manipulá-la inteira na tela de vídeo ou em qualquer outro dispositivo gráfico.

O sistema gráfico pode armazenar na estrutura de dados do aplicativo todas as informações geométricas, ou até outras mais, como (no caso de um amortecedor) características mecânicas, propriedades dinâmicas, etc.

A maneira simples de se implementar uma estrutura segmentada em nível processual é separando por partes cada elemento gráfico a ser definido; assim, ao se descrever um elemento gráfico envia-se uma série de comandos a um dispositivo gráfico qualquer, a fim de formar a imagem deste, se o sistema capta estes comandos, alocando-os de maneira logica e seqüenciada em "segmentos", é possível se tratar este "grupo de comandos" como um elemento isolado (segmento). Assim um segmento pode ser manipulado e visualizado como se fosse uma entidade isolada e podem também referir-se a outros segmentos em uma descrição hierárquica de imagens.

4.2 O Sistema Segmentado

O sistema proposto sustenta um modelo hierárquico de dados gráficos (segmentados), baseado em primitivos e em atributos gráficos [1,3,4]. Possibilitando assim, uma flexibilidade muito grande para se construir uma potente biblioteca gráfica, a partir das quais se construirão as aplicações.

A implementação de um sistema deste porte se torna extremamente simples se for programada em linguagem "C", pois a biblioteca gráfica tem todas as características inerentes à esta linguagem, incluindo um eficiente e compacto código e um alto grau de portabilidade. Assim, construindo-se o núcleo principal do sistema gráfico em "C", o desenvolvimento de aplicativos, que utilize este núcleo, poderá ser implementado produtiva e significativamente na própria linguagem "C" ou em outras linguagens que admitam mixação.

4.2.1 Criando Segmentos

Segmentos são construídos através de uma seqüência lógica em memória RAM ou em arquivos de disco. Primeiro cria-se o segmento, então, a informação (comandos gráficos) é adicionada ao segmento. Finalmente, o segmento é fechado. A rotina que fecha o segmento retorna alguma forma de indicação para o mesmo, podendo este ser então, utilizado como se fosse uma entidade isolada. O esboço do código para esta operação pode ser:

```
Cria_Segmento (nome);
    Atributo (...);
    Primitiva (...);
    ...
    Primitiva (...);
    Referencie (...);
Fecha_Segmento();
```

Um exemplo simples de segmentação é a descrição de um bloco de uma cidade, consistindo de quatro casas idênticas (figura 4.1). Cada casa tem três janelas, uma porta, o telhado e a parte que forma a armação.

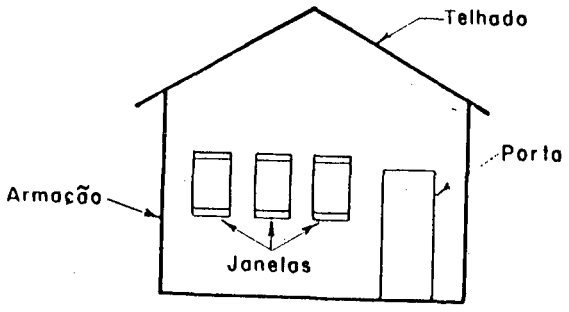


FIGURA 4.1 - Exemplo de entidade gráfica contida num segmento

Primeiramente são criados segmentos para a janela e a porta:

```
Cria_Segmento ("janela");
    Atributo (AZUL);
    Retangulo (10,10,60,80);
    Retangulo (10,00,60,90);
Fecha_Segmento();

Cria_Segmento ("porta");
    Atributo (ROXO);
    Retangulo (00,00,50,100);
Fecha_Segmento();
```

Em seguida, é criado o segmento "casa", que referencia os segmentos "porta" e "janela".

```
Cria_Segmento ("casa");
    Atributo (AMARELO);
    Retangulo (100,100,600,500);

    Inic_Poliline;
        vertce (50,480);
        vertce (350, 600);
        vertce (650,480);
    Fim_Poliline();

    Referencie ("janela", matran);
    Referencie ("janela", matran);
    Referencie ("janela", matran);
    Referencie ("porta" , matran);
Fecha_Segmento();
```

O atributo refere-se sempre aos primitivos gráficos que o seguem, podendo haver mais de uma declaração de atributos em um segmento.

Observe o formato do comando que referencia outro segmento; o nome do segmento e a matriz de transformação (matran - vide capítulo 6) que será utilizada na obtenção da posição relativa onde este segmento se situará dentro do segmento que o referencia, são passados à função referenciadora. Estas simples características fazem da "referência de segmentos" um comando poderoso que permite a representação hierárquica dos dados gráficos.

O passo final é a criação do segmento "cidade" que referenciará o segmento casa.

```
Cria_Segmento (cidade);
    Referencie ("casa", matran);
    Referencie ("casa", matran);
    Referencie ("casa", matran);
    Referencie ("casa", matran);
 Fecha_Segmento();
```

É notável a simplicidade da estrutura hierárquica deste exemplo. O bloco "cidade" contém "casas" as quais, por sua vez, contém "janelas" e "portas". Se não fosse utilizdao um método hierárquico para descrever o bloco "cidade", seria necessária a codificação em separado de doze janelas, quatro portas e quatro armações.

4.2.2 Usando Segmentos

Após criado, o segmento nada mais é que um bloco de dados contendo informações gráficas a cerca de uma entidade qualquer. Existirão, portanto, procedimentos que terão a função de traduzir as informações gráficas de cada segmento para um dispositivo gráfico de saída específico (normalmente tela gráfica ou plotter). No caso de tela gráfica tem-se a função:

```
desenha_segmento (nome_do_segmento);
```

Note que, quando a linguagem de programação utilizada beneficia-se do uso de ponteiros, a rotina, que desenha o segmento, usa o ponteiro de segmento para localizá-lo e não uma lista de nomes para localizar este segmento, esta prática otimiza o máximo possível a velocidade de colocação do desenho na tela ou em outro dispositivo gráfico qualquer.

4.2.3 Salvando Segmentos

Na utilização de sistemas segmentados é natural, e imprescindível em grandes aplicações, a necessidade de salvar os segmentos em arquivos de disco. Os segmentos arquivados em disco podem ser referenciados a partir de segmentos comuns na memória principal (RAM) ou, também, a partir de outros segmentos contidos no mesmo arquivo, ou em outro arquivo qualquer do disco.

Com a opção de arquivos de segmentos, o aplicativo poderá usar segmentos além do que a memória comportaria. Se uma aplicação usar toda a memória principal e precisar de mais, o sistema gráfico determinará quais os segmentos que podem ser escritos em arquivos, e quais os que devem ser mantidos na memória.

4.3 Primitivas Gráficas

O grau mais baixo dos elementos gráficos no sistema de segmentação são as primitivas e os operadores a eles associados [1,17]. A seguir será dada uma breve descrição das primitivas mais usadas nos sistemas gráficos. Cada primitiva segue uma estrutura de dados que se adapta bem à linguagem "C". Existe também um aparato de funções que criam, desenham e inserem a primitiva em um segmento aberto.

4.3.1 O Ponto

A primitiva mais simples é o ponto. A estrutura em linguagem "C" para o ponto é:

```
struct pmt_ponto {
            int  x,y;
};

typedef struct pmt_ponto t_ponto;
```

e para criar um ponto, foi definida a função:

```
t_ponto  * AlocEsp_ponto (x,y);
```

a qual retorna um ponteiro para uma variável do tipo t_ponto, quando são passados os argumentos "x" e "y", coordenadas absolutas do ponto.

4.3.2 O Retângulo

Esta primitiva descreve o elemento gráfico usado para representar todas as regiões retangulares como, janelas, "viewports", etc. Sua estrutura é:

```
struct pmt_retangulo {
                int  x_min;
                int  y_min;
                int  x_max;
                int  y_max;
};

typedef pmt_retangulo  t_retng;
```

Como anteriormente, existirá uma função que criará um retângulo, retornando um ponteiro para uma variável do tipo t_retng;

A figura a seguir mostra um esboço da primitiva retângulo.

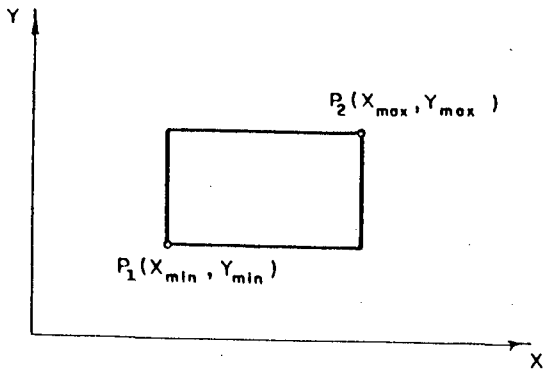


FIGURA 4.2 - Exemplo da primitiva retângulo

Uma das principais utilizações da primitiva retângulo é na descrição do "bounding box" de um segmento ou de uma outra primitiva qualquer. O "bounding box" é definido como sendo a menor área retangular que circunscreve a primitiva ou segmento, como na figura 4.3 a seguir.

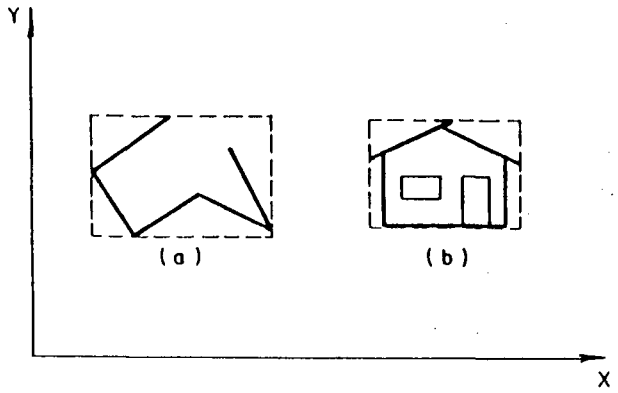


FIGURA 4.3 - "Bounding box" para uma primitiva (a) e para um segmento (b).

"Bounding boxes" são utilizados, neste caso, para determinar se haverá necessidade de recorte (Clipping) do elemento ou não, evitando que o sistema fique verificando para cada primitiva do elemento e perdendo tempo com isto.

4.3.3 A Elipse

A próxima primitiva a ser considerada tem a mesma estrutura de dados do retângulo, porém descreve elipses. Neste caso, o retângulo define apenas o "bounding box" da elipse, ou seja, os eixos maior e menor da elipse (que no caso é uma elipse). A função que irá interpretar esta primitiva decidirá se deve traçar uma elipse ou um círculo, em função das dimensões dos dois eixos. Como a estrutura de dados é a mesma do retângulo, em "C", só é preciso a declaração:

```
#define  t_elipse  t_retng
```

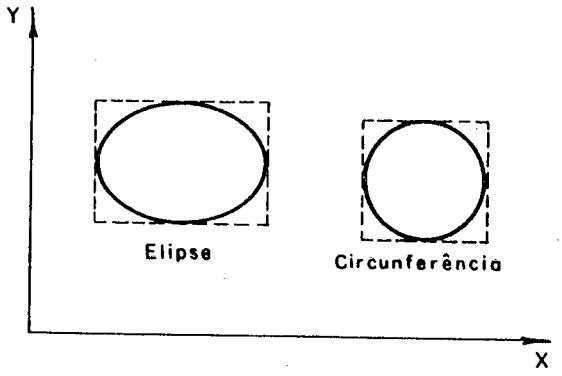


FIGURA 4.4 - Exemplo de primitiva do tipo elipse.

4.3.4 O Multiponto

O multiponto não se encaixa exatamente no contexto de "primitiva", sendo utilizado aqui apenas como auxiliar na construção das primitivas polilinha e polígono. A estrutura para o multiponto é:

```
struct  pmt_MultiPonto; {
                t_ponto *ponto;
                struct pmt_MultiPonto *next;
}

typedef struct pmt_MultiPonto  t_multpt;
```

4.3.5 A Polilinha

Esta primitiva tem o seu comprimento de armazenamento em memória variável, e por esta razão é implementada através de listas encadeadas para aumentar a eficiência e flexibilidade. A primitiva consiste de uma parte inicial seguida de uma lista de nós. A lista de nós é feita com o auxílio da primitiva multiponto.

```
struct pmt_PoliLinha; {
                int         num_pontos;
                t_retng     *bbox;
                t_multpt    *vertice;
};

typedef struct pmt_PoliLinha t_polilinha;
```

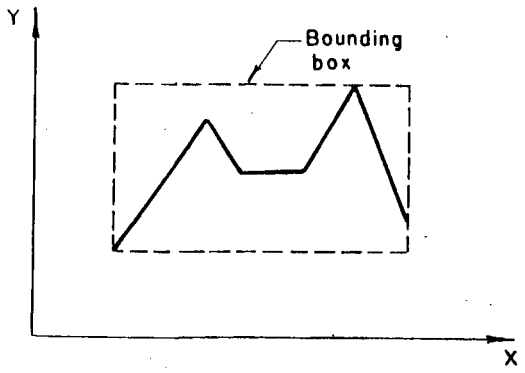


FIGURA 4.5 - Exemplo de primitiva do tipo polilinha

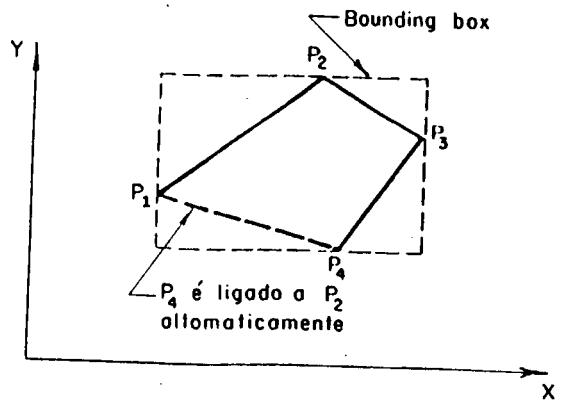


FIGURA 4.6 - Exemplo de primitiva do tipo Polígono

4.3.6 O Polígono

O polígono tem a mesma estrutura de uma polilinha, porém é definido separadamente, pois a função que o lê sempre ligará o último vértice ao primeiro.

Para que se possa criar uma primitiva gráfica do tipo polilinha ou polígono, deve-se chamar a rotina "Inic_linha", e então, adicionar os vários vértices através de sucessivas chamadas à função "vertce", finalmente chama-se "Fim_Poliline" que retorna um ponteiro para uma variável do tipo "t_polilinha" ou "t_polig". Por exemplo no caso da criação de uma Polilinha, ter-se-ia o seguinte código:

```
t_polilinha  *lin_queb;

Inic_linha();
    vertce(x1,y1);
    vertce(x2,y2);
    ...
    ...
    vertce(xN,yN);
lin_queb = end_line();
```

veja graficamente com fica a estrutura na figura a seguir.

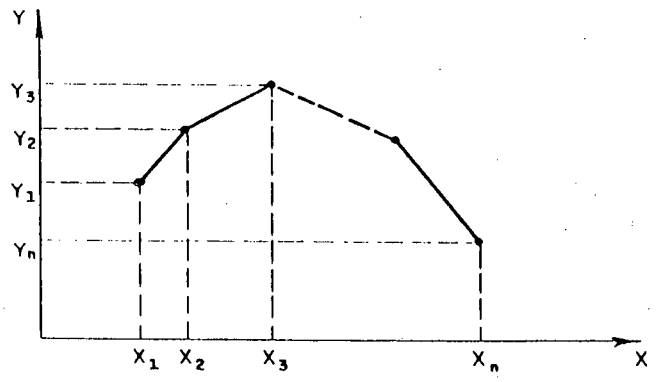


FIGURA 4.7 - Construção de uma primitiva do tipo polilinha

4.3.7 A Primitiva Texto

A Primitiva texto constitui-se de uma seqüência de caracteres (códigos ascii) mais alguns atributos que irão definir a sua forma de apresentação [8,17]. Um dos conceitos importantes para manipulação de texto é a justificação, com ela especifica-se o exato local de colocação do texto dentro de um "bounding box" previamente definido para o mesmo.

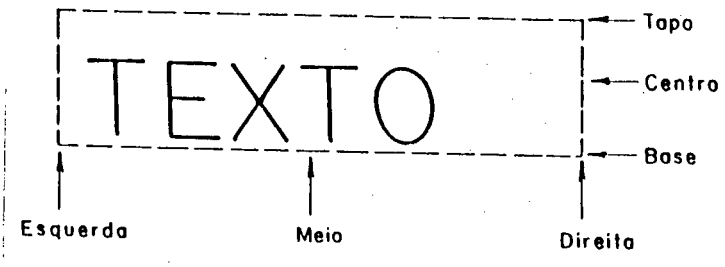


FIGURA 4.8 - Exemplo de texto justificado

A estrutura para a primitiva texto foi implementada em "C" da seguinte forma:

```
struct  pmt_texto {
            t_ponto   origem;
            t_retng   *bbox;
            char      just[2],
                      text[32];
};

typedef struct pmt_texto t_texto
```

4.4 Atributos

Atributos são operadores gráficos que modificam a maneira pela qual as primitivas gráficas podem ser apresentadas [1,17]. Assim, um segmento de reta pode aparecer com espessuras diferentes, tracejado ou pontilhado, um texto pode ser desenhado em diferentes tamanhos, espessuras, fontes, ou cores diferentes.

A colocação de atributos, no "zmSAG", segue o princípio de "finite-state-machine" (FSM) [21]. O sistema utiliza-se de um conjunto de parâmetros, conhecidos como "estado da máquina", que controlam uma determinada operação. Normalmente, uma FSM irá definir um novo estado de valores que farão a máquina operar de forma diferente com o mesmo tipo de dados. Os comandos de modificação da FSM (novos atributos) serão fornecidos pelo aplicativo (programa que utiliza o pacote gráfico).

4.4.1 Classes de Atributos

Os atributos são agrupados em classes distintas que, constituirão uma parte do FSM [21]. As classes de atributos utilizadas são:

- "bit map" draw mode

Indica como deve ser a forma de mudança dos bits no frame-buffer de dispositivos gráficos matriciais; quando este atributo for COPY, os bits novos serão copiados em cima dos antigos, quando for XOR deverá ser feita uma operação lógica, bit a bit, do tipo "xor" dos bits novos com os já existentes no frame-buffer.

- cor de desenho

Irá especificar em que cor deverá ser desenhada a primitiva gráfica na tela. Em casos de plotagem, poderá ser usado para definir a espessura do traço.

- preenchimento de áreas

Para esta classe temos as opções FILL e NOFILL que decidirão sobre o preenchimento ou não, de uma determinada área (polígono fechado), com determinada cor ou um padrão de textura especificado.

- definição de borda

Utilizado pela primitiva retângulo, este atributo decide pela colocação ou não de bordas em áreas retangulares. Este atributo pode assumir os estados FRAME e NOFRAME.

Existe ainda o "atributo" RESET que coloca os atributos de todas as classes em um padrão previamente definido. Normalmente é utilizado na iniciação do sistema, mas pode também ser utilizado pelo programador em qualquer parte do aplicativo.

4.5 Operações com Segmentos

Devido ao fato dos segmentos serem variáveis na extensão da estrutura de dados, o conceito de listas encadeadas (vide apêndice D) se faz apropriado na implementação de sistemas segmentados. Cada segmento consiste de uma parte principal, que designa parâmetros inerentes ao segmento, seguido por uma listagem de elementos chamados de "elos". Cada elo comporta um atributo ou uma primitiva. A figura 4.9 mostra (fazendo uso de simbologia de listas encadeadas) um exemplo

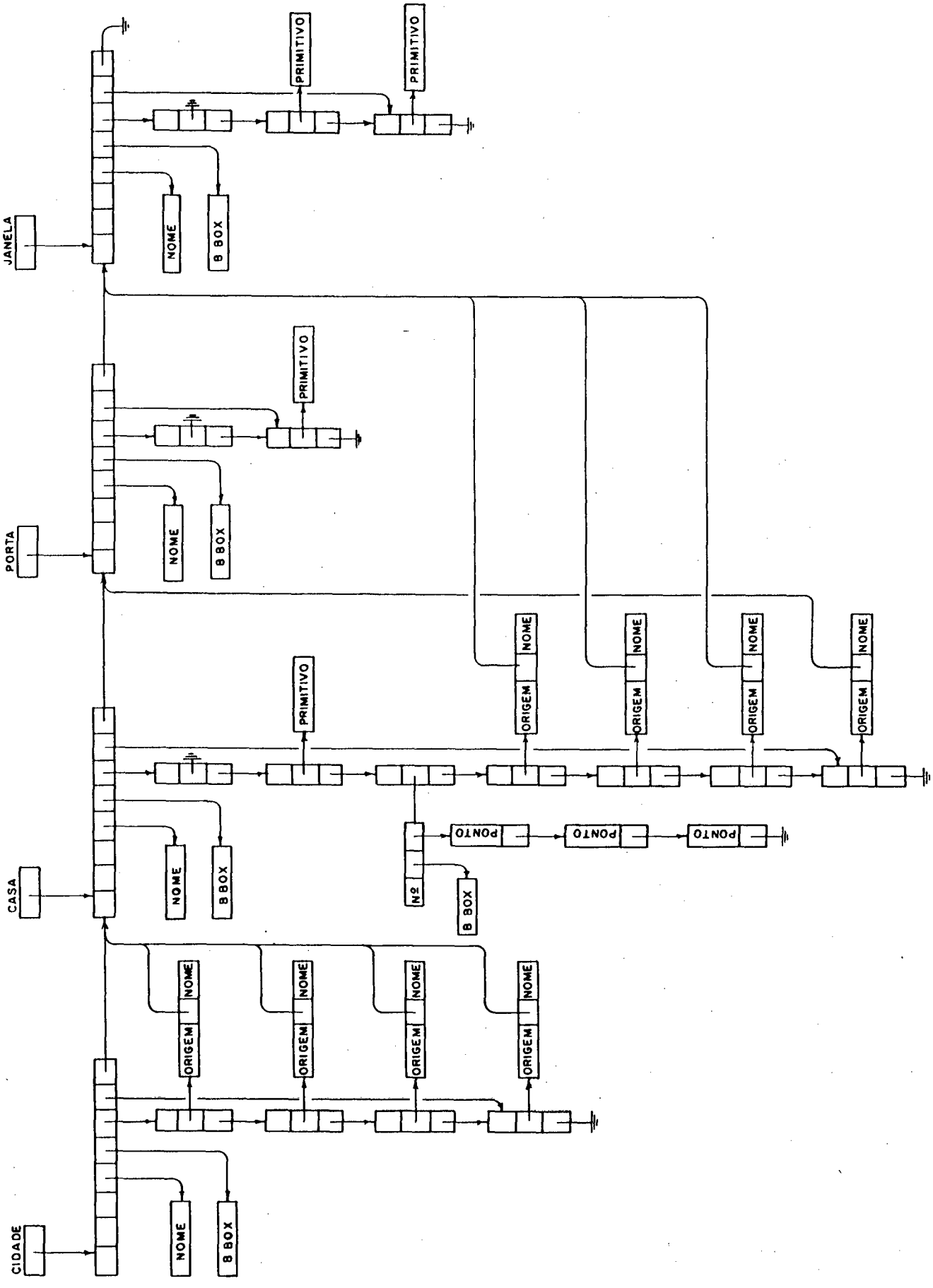


FIGURA 4.9 - Simbologia para a estruturação de dados do *segmento cidade*.

de como são dispostos os segmentos no sistema gráfico "zmSAG". A estrutura em "C" para definição de segmentos é:

```
struct  pmt_segmento {
            int         n_ref,
                        visible,
                        locked;
            char        *name;
            t_retng     *bbox;
            t_elo       *data,
                        *eol;
            float       *matran
            struct pmt_segmento *next;
};

typedef struct pmt_segmento t_seg;
```

Devido ao fato de um segmento poder ser referenciado por outros, deve-se saber, de antemão, quantos segmentos o referenciam antes de um eventual comando de deleção, a variável "n_ref" tem esta finalidade. Ao se tentar desenhar ou gravar em disco um segmento que referencie um outro que foi "deletado", pode-se até causar danos ao sistema e aos discos.

A variável "visible" assume os valores TRUE e FALSE, permitindo ou não que um segmento possa ser desenhado na tela em dado instante.

A variável "locked" também assume os valores TRUE e FALSE e evita que o segmento seja retirado da memória. Atua assim, dando prioridade de permanência ao segmento.

A variável "nome" é do tipo ponteiro e apontará para uma string de caracteres que será um nome dado ao segmento. Note que embora o nome possa ter até trinta e dois caracteres, a variável "nome" só ocupará dois bytes na memória.

Da mesma forma "bbox" é um apontador para uma estrutura do tipo t_retng que representará o "bounding box" do segmento.

As variáveis "data" e "eol" são apontadores para o começo e o fim, respectivamente, da lista de "elos", veja a figura 4.9.

A variável "matran" é um ponteiro a uma matriz de transformação que poderá ser utilizada para se desenhar o segmento pela função "draw_seg".

Finalmente a variável "next" aponta para o próximo segmento da lista, permitindo que todos os segmentos do sistema sejam tratados como uma lista e dando maior flexibilidade à estrutura de dados.

CAPÍTULO 5

Geração de Primitivas Gráficas

5.1 Introdução

Os algorítmos gráficos desenvolvidos para o "zmSAG" se destinam, com poucas exceções, à impressora, uma vez que no caso do traçador gráfico e vídeo pode-se aproveitar a interface gráfica destes dispositivos que oferece uma boa performance em termos de velocidade e acüidade gráfica.

Estes algoritmos também poderão ser utilizados em futuros periféricos gráficos do tipo matricial que venham a ser incorporados ao "zmSAG" e que não tenham uma interface gráfica própria.

5.2 Os Dispositivos Matriciais

Nos dispositivos gráficos chamados "Matriciais" a representação da imagem é feita por meio de um conjunto de pixels (pontos), que podem assumir posições sobre uma grade regular [1,5]. Em vista disto, o traçado de primitivas gráficas elementares como segmentos de reta, arcos de circunferência, elipses, etc., impõem a construção de algoritmos capazes de determinar, na matriz de pixels da superfície de exibição, quais pixels devem ser alterados, de forma a simular-se a aparência do elemento gráfico desejado. Os algoritimos que executam estas tarefas são chamados "algoritmos de conversão matricial" [5], por converterem numa representação matricial elementos gráficos expressos numa representação vetorial.

Em um dispositivo matricial, a estrutura na qual os pixels estão organizados sugere a adoção do sistema cartesiano de coordenadas para o endereçamento de cada pixel. Em geral, considera-se o espaço de exibição (figura 5.1) como um reticulado quadrangular de pixels de coordenadas satisfazendo a:

$$0 \leq x \leq x_{max}$$

$$0 \leq y \leq y_{max}$$

Onde x e y devem ser inteiros.

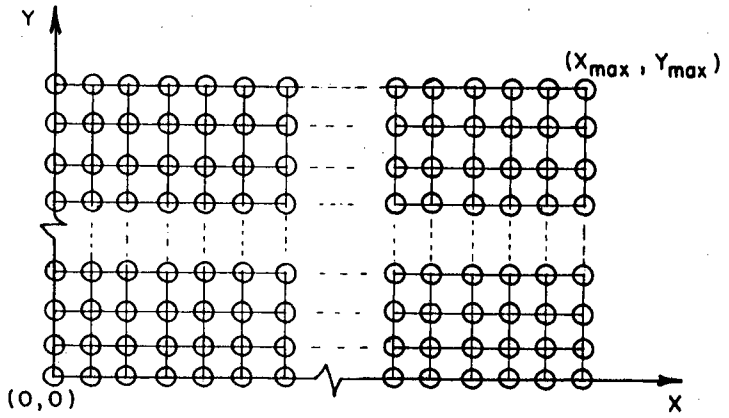


FIGURA 5.1 - Espaço de exibição de um dispositivo matricial

Cada pixel desta malha terá um atributo que, dependendo do dispositivo, poderá ser um atributo de cor ou um atributo do tipo ligado/desligado. O acesso ao atributo será feito através dos seguintes procedimentos :

SetPixel (x,y) - faz com que o pixel de coordenadas (x,y) passe a ficar ativo (ligado).

GetPixel (x,y) - verifica se o pixel de coordenadas (x,y) está ativado.

O segundo procedimento tem utilização importante no caso de preenchimento de áreas fechadas.

5.3 Conversão Matricial

Os algoritmos de conversão matricial de primitivas gráficas devem ser fundados em certas propriedades para que possam gerar representações aceitáveis pelo observador.

Algumas destas propriedades são:

i) intensidade ou espessura uniforme - a imagem não varia de intensidade ou espessura ao longo de sua extensão;

ii) continuidade - a imagem, sempre que possível, não deverá apresentar interrupções em seu traçado;

iii) precisão - a imagem das primitivas deve iniciar-se e terminar nos pontos especificados;

iv) uniformidade de traço - segmentos de diferentes inclinações devem rem possuir imagens de mesma intensidade.

O atendimento a todos esses requisitos nem sempre é possível sem que haja uma perda substancial na eficiência dos algoritmos e/ou um aumento significativo na complexidade do dispositivo gráfico.

O "zmSAG", procura obter a máxima coerência com os critérios descritos acima, apesar da utilizaçao de dispositivos periféricos de baixo custo, porém a preocupaçao maior se concentrou na obtenção de uma melhor performance dos algoritmos de conversão matricial, no que diz respeito à velocidade. Considerando que o mesmo deverá ser implantado em CPU's que não ultrapassam a faixa de 1,2 mips (microcomputadores PC's compatíveis ou supermicros à base do UNIX).

5.4 Algoritmos de conversão

5.4.1 O Segmento de Reta

Considere-se um segmento de reta cujos extremos D_1 e D_2 possuam coordenadas inteiras (x_1,y_1) e (x_2,y_2) respectivamente, de tal forma que sejam satisfeitas as seguintes relações:

$$0 \leq x_1 \leq x_2 \tag{5.1}$$

$$0 \leq y_1 \leq y_2 \tag{5.2}$$

$$y_2 - y_1 < x_2 - x_1 \tag{5.3}$$

Significando que o segmento de reta se encontra no primeiro octante, tendo inclinação positiva na direção de D_1 para D_2. Caso isto não seja satisfeito, a estrutura de simetria do sistema cartesiano permite que, através de reflexões em torno dos eixos coordenados e das diagonais [5], se chegue facilmente a esta condição (figura 5.2), tendo porém este caso, como consequência, a transformação reflexiva inversa de cada ponto obtido ao seu octante de origem. Com base nisto, é possível se imaginar que um algorítmo que gere um segmento no primeiro octante, poderá ser estendido de forma a que possa gerar segmentos em todos os octantes com poucas modificações.

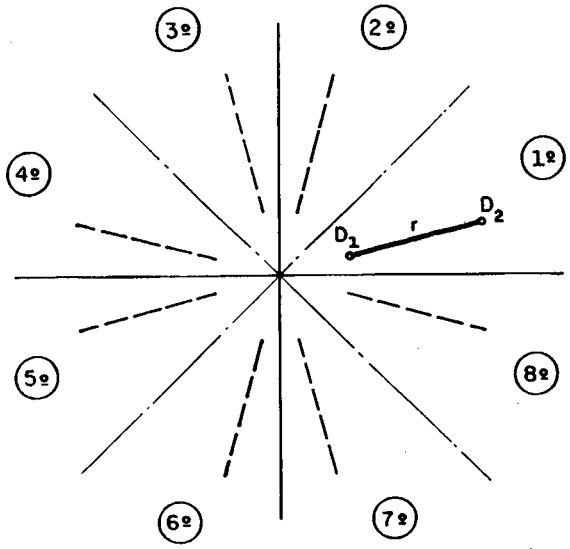


FIGURA 5.2 - Estrutura de simetria do sistema cartesiano

Posta esta simplificação, é de fácil percepção que o segmento de reta corta um número maior de verticais que de horizontais do reticulado, justificando o chamado

> Critério de conversão - em cada vertical do reticulado com abscissa entre x1 e x2, apenas o pixel mais próximo do segmento faz parte da sua imagem (figura 5.3).

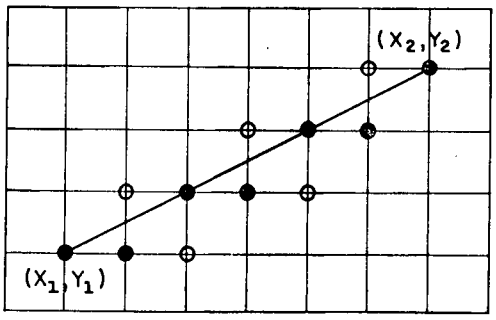


FIGURA 5.3 - Aproximação da imagem num dispositivo matricial

Baseando-se neste critério, Bresenham [13] propôs um algoritmo bastante simples que, como se vê mais adiante, dispensa o uso de operações trigonométricas, multiplicação ou divisão no "loop" principal.

No problema inicial, Bresenham se preocupava com o traçador gráfico (plotter) que tinha a capacidade de ligar dois pontos por uma linha contínua, podendo escolher oito direções possíveis (figura 5.4a).

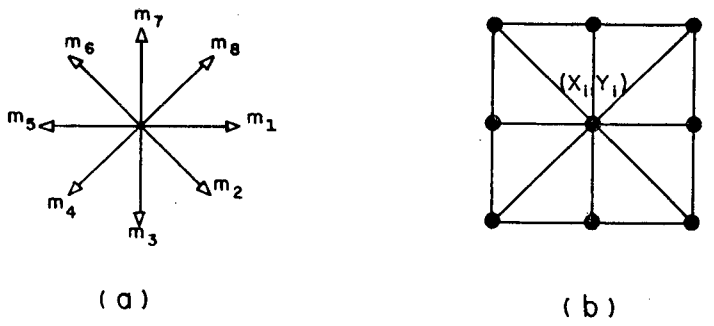


FIGURA 5.4 - Possibilidades do traçador gráfico (a) e sua adaptação em dispositivos matriciais (b)

O algoritmo de Bresenham pode ser adaptado para a geração de linhas em dispositivos matriciais [1], considerando-se que há, como no caso do traçador, oito pixels vizinhos e, portanto, oito movimentos possíveis a partir da coordenada (X_i, Y_i), (figura 5.4b).

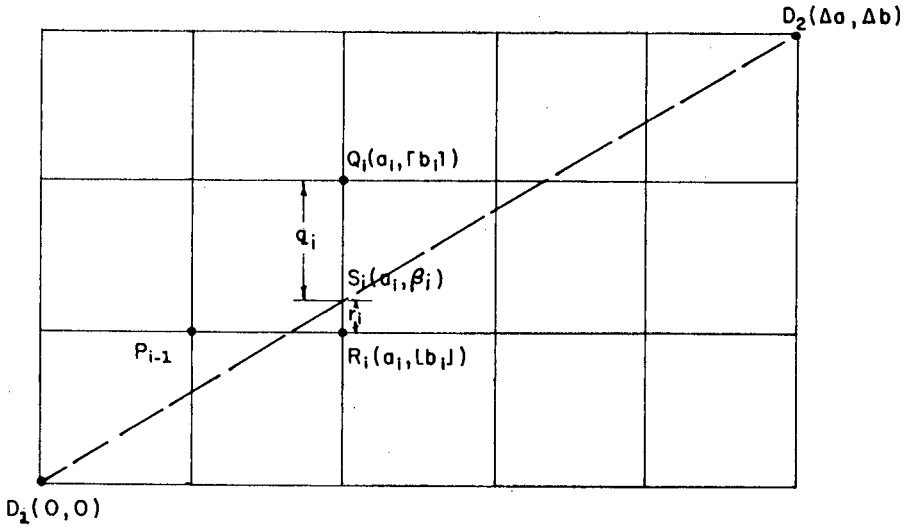


FIGURA 5.5 - O algoritmo de Bresenham se baseia na escolha dos pontos Q_i ou R_i através de verificação dos parâmetros r_i e q_i

Uma ampliação do segmento de reta D_1D_2 (figura 5.5) irá mostrar de forma mais detalhada os parâmetros utilizados no desenvolvimento do algoritmo.

Por questões de simplificação, efetua-se a translação do segmento, fazendo coincidir o ponto D_1 com a origem. Desta forma a expressão:

$$D_2(\Delta a,\Delta b) = (x_2 - x_1, y_2 - y_1) \tag{5.4}$$

fornece as novas coordenadas para D_2.

Supondo-se já conhecida a localização do "pixel" P_{i-1}, a próxima escolha deverá ser R_i, se $r_i < q_i$ ou Q_i se $r_i \geq q_i$ de acordo com o critério de conversão.

Desde que a > 0 sempre, uma vez que o segmento se encontra no primeiro octante, o termo

$$\delta_i = (r_i - q_i)\,\Delta a \tag{5.5}$$

terá sempre o sinal de $r_i - q_i$, ou seja, $\delta_i < 0$, Q_i será a melhor escolha, caso contrário $\delta_i \geq 0$, R_i deverá ser o escolhido.

Supondo (a_i,β_i) como coordenadas de S_i, tem-se $(a_i,\lfloor\beta_i\rfloor)$ e $(a_i,\lceil\beta_i\rceil)$ para coordenadas de R_i e Q_i respectivamente.

O operador $\lfloor\beta_i\rfloor$ significa o maior inteiro imediatamente inferior a β_i e $\lceil\beta_i\rceil$ o menor inteiro imediatamente superior a β_i.

Assim:

$$\delta_i = [(\beta_i - \lfloor\beta_i\rfloor) - (\lceil\beta_i\rceil - \beta_i)]\,\Delta a \tag{5.6}$$

porém,

$$\beta_i = \frac{\Delta b}{\Delta a}\, a_i \tag{5.7}$$

$$\delta_i = 2a_i\Delta b - (\lfloor\beta_i\rfloor + \lceil\beta_i\rceil)\,\Delta a \tag{5.8}$$

sendo ainda

$$a_i = a_{i-1} + 1 \tag{5.9}$$

$$\lceil\beta_i\rceil = b_{i-1} + 1 \tag{5.10}$$

$$\lfloor\beta_i\rfloor = b_{i-1} \tag{5.11}$$

pode-se reescrever

$$\delta_i = 2a_{i-1}\Delta b - 2b_{i-1}\Delta a + 2\Delta b - \Delta a \tag{5.12}$$

Das condições iniciais do problema, em P_o, $a_o = 0$ e $b_o = 0$, então:

$$\delta_i = 2\Delta b - \Delta a \tag{5.13}$$

e se

$$\delta_i \geq 0,$$

$$b_i = b_{i-1} + 1, \tag{5.14}$$

pode-se obter:

$$\begin{aligned}\delta_{i+1} &= 2(a_{i-1}+1)\Delta b - 2(b_{i-1}+1)\Delta a + 2\Delta b - \Delta a \\ &= \delta_i + 2\Delta b - 2\Delta a\end{aligned} \tag{5.15}$$

agora se

$$\delta_i < 0,$$

$$b_i = b_{i-1}, \tag{5.16}$$

obtem-se então:

$$\begin{aligned}\delta_{i+1} &= 2(a_{i-1}+1)\Delta b - 2b_i\Delta a + 2\Delta b - \Delta a \\ &= \delta_i + 2\Delta b\end{aligned} \tag{5.17}$$

Tudo isto pode ser resumido da seguinte forma; para δ inicial, tome-se:

$$\delta_i = 2\Delta b - \Delta a \tag{5.18}$$

e em cada iteração o próximo δ será:

$$\delta_{i+1} = \begin{cases} \delta_i + 2\Delta b - 2\Delta a & \text{se } \delta_i \geq 0 \\ \delta_i + 2\Delta b & \text{se } \delta_i < 0 \end{cases} \tag{5.19}$$

sendo:

$$\Delta a = x_2 - x_1 \tag{5.20}$$

$$\Delta b = y_2 - y_1 \tag{5.21}$$

E para segmentos de reta no primeiro octante, pode-se ter o seguinte algoritmo que mostra a elegância e eficiência do método:

```
Δa ← x2 - x1;
Δb ← y2 - y1;
δ  ← -Δa;
para x de x1 até x2, passo 1
    SetPixel (x, y);
    δ ← δ + 2*Δb
    se δ ≥ 0, entao
        y ← y + 1;
        δ ← δ - 2*Δa;
    fim se
fim para.
```

Observe que dentro do loop não há nenhum produto, uma vez que Δa e Δb são constantes os valores 2Δa e 2Δb também serão. Veja ainda que nem fora do "loop" tem-se algum tipo de produto, pois o produto de um inteiro qualquer por duas unidades pode ser permutado por um rapidíssimo deslocamento binário nos sistemas digitais. Em linguagem "C" tem-se:

$$2^*\Delta b \Leftrightarrow \Delta b << 1$$

que corresponde, em linguagem de máquina, a um rapidíssimo deslocamento binário de uma unidade da direita para a esquerda na variável inteira Db.

Embora tenha-se restringido o desenvolvimento do algoritmo à conversão matricial de segmentos no primeiro octante, o uso de reflexões em relação aos eixos de simetria fornece uma extensão do método a segmentos em outros octantes. Note-se porém, que se a imagem de um segmento no 1º octante é refletida para o 2º, o critério de conversão deve ser alterado substituindo-se vertical por horizontal, *x* por *y* e abcissa por coordenada. O algoritmo em sua forma completa pode ser visto no código fonte original do "zmSAG".

5.4.2 A Circunferência

Existem várias maneiras de se obter a conversão matricial de circunferências, porém em sua maioria se constituem ineficientes. Considere-se por exemplo, a circunferência de centro na origem, para a qual tem-se:

$$x^2 + y^2 = R^2 \qquad (5.22)$$

resolvendo em y

$$y = \pm \sqrt{R^2 - x^2} \qquad (5.23)$$

Para se obter um quarto de círculo, é necessário que se faça x variar de 0 a R com incrementos unitários. Além deste método ser bastante ineficiente, pode-se ver pela figura 5.6 o quão falho ele é no que diz respeito à continuidade da curva.

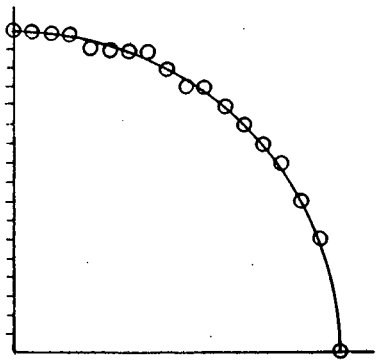


FIGURA 5.6 - Aspecto da circunferência quando da utilização de sua eq. para traçado em um dispositivo matricial

Um método similar, mas também ineficiente seria através da equação paramétrica da circunferência

$$y = R \operatorname{sen}(t) \qquad (5.24)$$

$$x = R \cos(t) \qquad (5.25)$$

Otimização através da simetria

Como no caso dos segmentos de reta, aqui também pode-se tirar vantagem da simetria da circunferência e, neste caso de uma forma muito mais eficiente [2,5].

Considere-se a circunferência de centro na origem (figura 5.7). Se o ponto (x,y) pertencer à circunferência, é possível a obtenção, por simetria, de sete outros pontos também pertencentes à esta circunferência. Desta forma pode-se utilizar a eq. (5.23) ou um outro método mais eficiente, a fim de se obter os valores de y para x variando de 0 a $R/\sqrt{2}$ (ponto onde $x = y$) e os respectivos sete pontos simétricos, formando a circunferência completa.

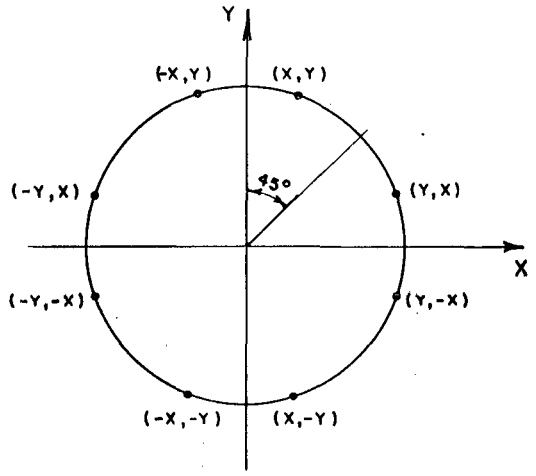


FIGURA 5.7 - Estrutura de simetria da circunferência

No caso em que o centro da circunferência não se encontra na origem é o bastante se fazer o deslocamento. De uma forma geral, para uma circunferência que tenha (x_0,y_0) como coordenadas de seu centro, o algoritmo em cada passo da iteração será:

```
SetPixel (x0 + x, y0 + y);
SetPixel (x0 + y, y0 + x);
SetPixel (x0 + y, y0 - x);
SetPixel (x0 + x, y0 - y);
SetPixel (x0 - x, y0 - y);
SetPixel (x0 - y, y0 - x);
SetPixel (x0 - y, y0 + x);
SetPixel (x0 - x, y0 + y);
```

O "zmSAG" se utiliza deste método de otimização tirando vantagem dos operadores "++" e "- -" existentes na linguagem "C". Nele são definidos, localmente, oito parâmetros que são função de x e y (figura 5.8).

Os parâmetros que levam o índice "ai" sempre são incrementados, os de índice "bi" são sempre decrementados em cada iteração. Os parâmetros de índice "aj" e "bj" serão incrementados e decrementados respectivamente somente quando "y" também o for (mais adiante será discutida a forma do incremento de y).

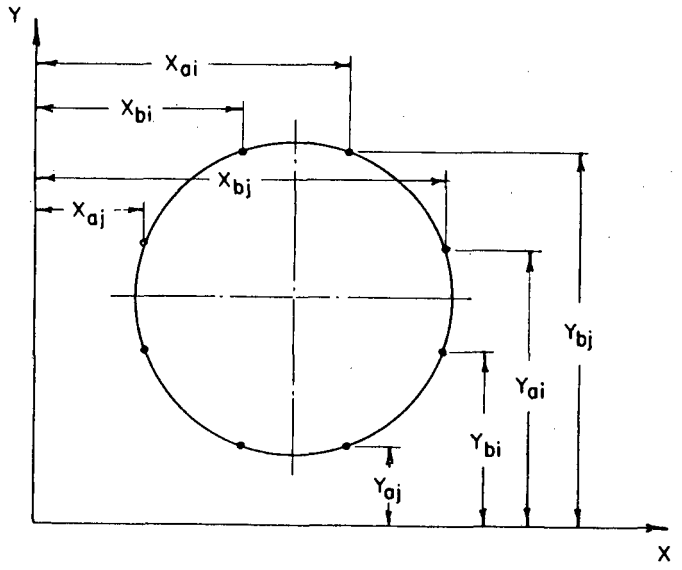


FIGURA 5.8 - Parâmetros utilizados no "zmSAG" para definir os oito pontos de simetria da circunferência

A parte do algoritmo que otimiza a construção da circunferência, em linguagem "C", toma a forma:

```
repita (enquanto x diferente de y)
    SetPixel (x_ai  ,y_bj  );
    SetPixel (x_bi  ,y_bj  );
    SetPixel (x_ai++,y_bi  );
    SetPixel (x_bi--,y_aj  );
    SetPixel (x_bi  ,y_ai  );
    SetPixel (x_bi  ,y_bi  );
    SetPixel (x_bj  ,y_ai++);
    SetPixel (x_bj  ,y_bi--);
    Se (houver decrementação em y)
        então
            x_bi ++;
            x_bj --;
            y_bj --;
            y_bi ++;
        fim se
    fim repita.
```

Algoritmo de Bresenham para a circunferência

Bresenham [12] desenvolveu um elegantíssimo e eficiente algoritmo incremental para a conversão matricial de circunferências. Concebido para ser utilizado em "plotters", o algoritmo gera todos os pontos de uma circunferência centrada na origem. O "zmSAG" usa uma adaptação do algoritmo o qual incrementa de x = 0 até x = $R/\sqrt{2}$ e usa o procedimento de simetria descrito anteriormente para obtenção da circunferência completa.

Utilizando a teoria de otimização proposta anteriormente e iniciando o traçado da circunferência a partir do ponto $P_o(0, R)$, figura 5.9a, apenas as opções de movimento m_1 e m_2 (mostradas na figura 5.4) serão necessárias, e o algoritmo de Bresenham pode ser simplificado como a seguir:

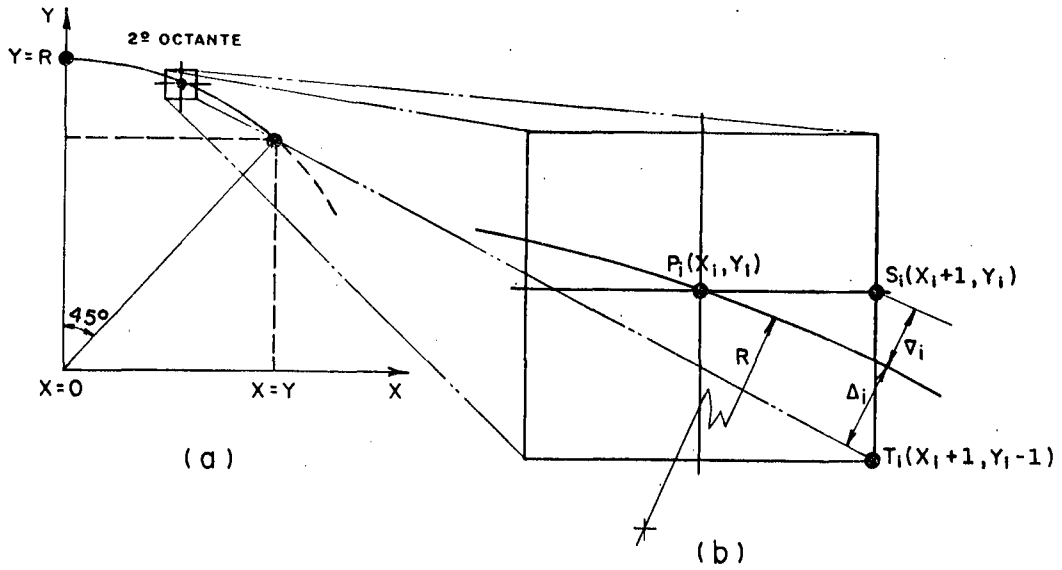


FIGURA 5.9 - Bresenham optava pela escolha dos pontos S_i ou T_i a partir dos valores de δ_i e δ_i

Partindo do ponto $P_i(x_i,y_i)$ o próximo pixel a ser aceso (ligado) poderá ser $S_i(x_i+1,y_i)$ ou $T_i(x_i+1,y_i-1)$. A melhor escolha pode ser feita verificando-se qual das seguintes diferenças absolutas tem menor valor.

$$\left|[(x_i + 1)^2 + y_i^2] - R^2\right| \tag{5.26}$$

ou

$$\left|[(x_i + 1)^2 + (y_i - 1)^2] - R^2\right| \tag{5.27}$$

De uma forma mais clara, do ponto de vista matemático, a escolha entre os dois pontos S_i e T_i pode ser feita observando-se o sinal de:

$$\delta_i = \left|[(x_i + 1)^2 + y_i^2] - R^2\right| - \left|[(x_i + 1)^2 + (y_i - 1)^2] - R^2\right| \tag{5.28}$$

se:

$\delta_i \leq 0$, então S_i será a melhor escolha

$\delta_i > 0$, então T_i será a melhor escolha

Observe que para a circunferência no 2º quadrante, y(x) é uma função monotonicamente decrescente, desta forma o segmento que vai do centro da circunferência ao ponto S_i excede ou é igual a R, ainda em conseqüência disto o segmento do centro a T_i é menor que R. Equacionando:

$$\nabla_i = [(x_i + 1)^2 + y_i^2] - R^2 \geq 0 \tag{5.29}$$

$$\Delta_i = [(x_i + 1)^2 + (y_i - 1)^2] - R^2 < 0 \tag{5.30}$$

e a equação (5.28) pode então ser reescrita

$$\begin{aligned}\delta_i &= [(x_i + 1)^2 + y_i^2] - R^2 + [(x_i + 1)^2 + (y_i - 1)^2] - R^2 \\ &= 2\{[(x_i + 1)^2 + (y_i - 1)^2] - R^2\} + 2y_i - 1 \\ &= 2\Delta_i + 2y_i - 1 \end{aligned} \tag{5.31}$$

Partindo agora para a obtenção de Δ_i nos pontos S_i e T_i. Sabendo-se ser:

$$\Delta_{i+1} = [(x_{i+1} + 1)^2 + (y_{i+1} - 1)^2] - R^2 \tag{5.32}$$

quando a opção for S_i, tem-se:

$$x_{i+1} = x_i + 1 \tag{5.33}$$

$$y_{i+1} = y_i \tag{5.34}$$

e então:

$$\begin{aligned}\Delta_{i+1} &= \{[(x_i + 1)^2 + (y_i - 1)^2] - R^2\} + 2(x_i + 1) + 1 \\ &= \Delta_i + 2x_{i+1} + 1 \end{aligned} \tag{5.35}$$

para a opção T_i

$$x_{i+1} = x_i + 1 \tag{5.36}$$

$$y_{i+1} = y_i - 1 \tag{5.37}$$

e fica-se com:

$$\Delta_{i+1} = \{[(x_i + 1)^2 + (y_i - 1)^2] - R^2\} + 2(x_i + 1) - 2(y_i - 1) + 2$$

$$= \Delta_i + 2x_{i+1} - 2y_{i+1} + 2 \qquad (5.38)$$

Em resumo;

Para S_i (i.e. $\delta_i \leq 0$)

$$x_{i+1} = x_i + 1 \qquad (5.39)$$

$$y_{i+1} = y_i \qquad (5.40)$$

$$\Delta_{i+1} = \Delta_i + 2x_{i+1} + 1 \qquad (5.41)$$

Para T_i (i.e. $\delta_i > 0$)

$$x_{i+1} = x_i + 1 \qquad (5.42)$$

$$y_{i+1} = y_i - 1 \qquad (5.43)$$

$$\Delta_{i+1} = \Delta_i + 2x_{i+1} - 2y_{i+1} + 2 \qquad (5.44)$$

Para o primeiro ponto ter-se-ia:

$$x_0 = 0 \qquad (5.45)$$

$$y_0 = R \qquad (5.46)$$

$$\Delta_0 = 2 - 2R \qquad (5.47)$$

e o algoritmo tomaria a forma:

```
enquanto x for menor que y, faça
    δ ← 2Δ + 2y - 1
    Δ ← Δ + 2x + 1
    se δ for maior que 0, então
        y ← y - 1
        Δ ← Δ - 2y + 1
    fim se
fim enquanto
```

A característica mais importante, a ser notada no algorítmo acima, é que não há nenhum produto a ser efetuado em seu "loop" principal, quando o mesmo

é adaptado à linguagem "C". Vale a pena ver um pequeno trecho deste algoritmo, nesta linguagem e na forma como ele foi implementado no "zmSAG", a fim de que se possa sentir a simplicidade e elegância do mesmo.

```
int  x  = 0, y = R,
     DT = 2 - R << 1, dt;

while (y > x++) {
    SetPixels(x, y);   /* liga 8 pixels */
    dt = (y + DT++) << 1;
    DT += x << 1;
    if (--dt > 0) {
        DT -= --y << 1;
        DT++;
    }
}
```

5.4.3 A Elipse

No caso da elipse, e das figuras cônicas em geral, se faz necessária a busca de novos critérios de conversão [5]. Se fosse utilizado, por exemplo, o critério de conversão de segmentos para a conversão matricial de um arco de elipse no primeiro quadrante, poder-se-ia ver pela figura 5.10 o quão pobre seria a imagem com respeito à continuidade da curva no trecho do arco de maior curvatura.

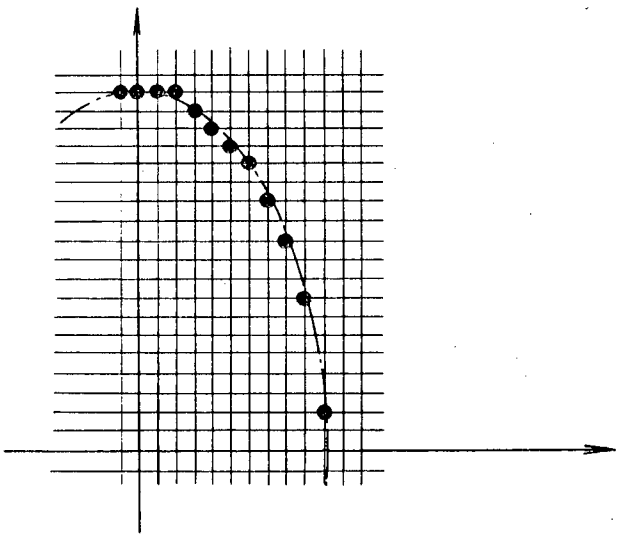


FIGURA 5.10 - Descontinuidade da imagem nas regiões em que a tangente é maior que um.

O que garante a continuidade da imagem gerada pelo critério usado na conversão de segmentos no primeiro octante, é o fato de naquele octante as interseções da curva (reta) com as verticais serem mais numerosas que com as horizontais. Isto não é verificado na região de maior curvatura da elipse da figura 5.10, porque a tangente ao arco naquela região não está no primeiro octante, e sim no quarto. Ou seja, a curva faz interseções mais numerosas com as horizontais que com as verticais.

Estas considerações justificam [5] um novo critério de conversão matricial para as cônicas:

a) Para cada vertical do reticulado que interceptar a curva em um ponto com tangente de módulo de inclinação menor que um, seleciona-se o pixel desta vertical mais próximo do ponto de interseção;

b) Para cada horizontal do reticulado que interceptar a curva em um ponto com tangente que tenha o módulo de sua inclinação maior que um, seleciona-se o pixel desta horizontal mais próximo do ponto de interseção.

O desenvolvimento do algoritmo de conversão para a elipse deve ser baseado no critério acima.

Seja a elipse com focos $F_1 = (-c,0)$ e $F_2 = (c,0)$, (figura 5.11). A sua equação pode ser expressa por:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad (5.48)$$

onde a é o seu semi-eixo maior, b é o seu semi-eixo menor e:

$$a^2 = b^2 + c^2 \qquad (5.49)$$

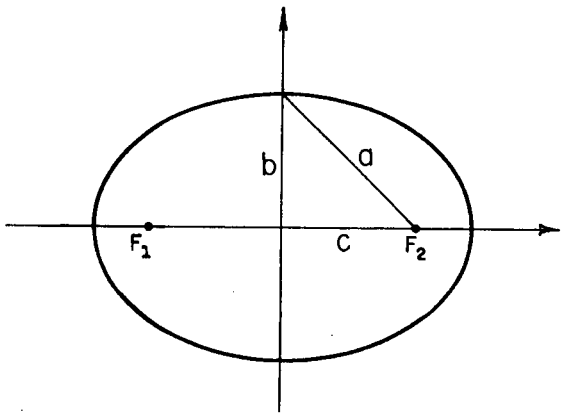


FIGURA 5.11 - Elipse de semi-eixos a e b

Embora esta elipse tenha centro na origem, e eixos de simetria horizontal e vertical, para se converter uma elipse desta classe com centro em (x_o, y_o) basta que se compute os pontos (x, y) da conversão da elipse centrada na origem e se execute a função:

SetPixel $(x_o + x, y_o + y)$

Por questões de simplicidade, é mais fácil desenvolver inicialmente o algoritmo para a parte do arco situada no 4º quadrante e com coeficiente da reta tangente menor que um. (Figura 5.12).

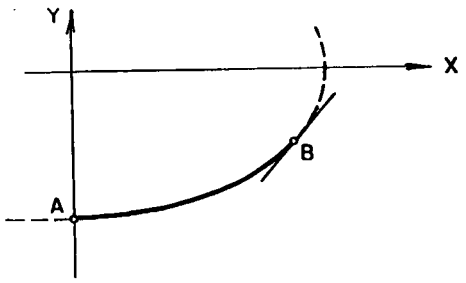


FIGURA 5.12 - Arco da elipse com coeficiente da reta tangente menor que um.

Suponha-se (figura 5.13) conhecido o pixel de coordenadas (x_i, y_i) deste trecho de arco, a próxima escolha deverá ser $P(x_i+1, y_i)$ ou $Q(x_i+1, y_i+1)$. Verificando-se o primeiro item do critério de conversão, a escolha será pelo ponto que estiver mais próximo da interseção da vertical de abcissa $x_i + 1$ com a curva.

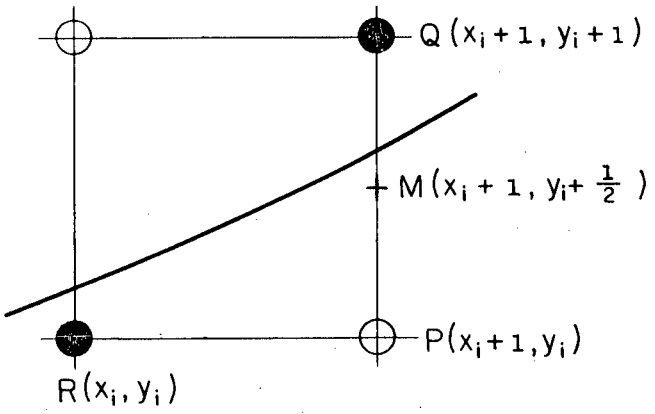


FIGURA 5.13 - Possibilidade de escolha dos pontos P ou Q verificando-se a localização do ponto M

Uma outra forma mais interessante de escolha dos pontos P ou Q seria pela verificação do ponto $M(x_i+1, y_i+1/2)$; se o mesmo recair fora da região interna à elipse a escolha se fará pelo ponto Q, recaindo dentro, se fará pelo ponto P.

Note que a elipse necessariamente passará entre os pontos P e Q uma vez que o coeficiente de sua tangente é menor que um no arco AB.

A eq. (5.48) pode também ser escrita na forma:

$$a^2y^2 + b^2x^2 = a^2b^2 \tag{5.50}$$

se for feito:

$$f(x, y) = a^2y^2 + b^2x^2 - a^2b^2 \tag{5.51}$$

então para um ponto $P(x_i, y_i)$:

se $f(x_i, y_i) < 0 \Rightarrow$ P é interior à elipse

se $f(x_i, y_i) > 0 \Rightarrow$ P é exterior à elipse

se $f(x_i, y_i) = 0 \Rightarrow$ P é um ponto da elipse

Desta forma, a partir de um ponto $R(x_i, y_i)$ pertencente a elipse convertida, pode-se deduzir qual será o próximo valor de y_{i+1}, verificando-se a expressão:

$$\Delta_i = f(x_i+1, y_i+1/2)$$

$$= a^2(y_i+1/2)^2 + b^2(x_i+1)^2 - a^2b^2 \tag{5.52}$$

Se

$$\Delta_i > 0 \Rightarrow y_{i+1} = y_i + 1 \tag{5.53}$$

$$\Delta_i < 0 \Rightarrow y_{i+1} = y_i \tag{5.54}$$

note que sempre se terá

$$x_{i+1} = x_i + 1 \tag{5.55}$$

Qualquer que seja o caso. É possível também se descobrir o valor de Δ_{i+1} a partir de Δ_i;

Se $\Delta_i < 0$, então:

$$\begin{aligned}\Delta_{i+1} &= a^2(y_{i+1}+1/2)^2 + b^2(x_{i+1}+1)^2 - a^2b^2 \\ &= a^2(y_i+1/2)^2 + b^2[(x_i+1)+1]^2 - a^2b^2 \\ &= a^2(y_i+1/2)^2 + b^2(x_i+1)^2 - a^2b^2 + 2b^2(x_i+1) + b^2 \\ &= \Delta_i + 2b^2x_{i+1} + b^2\end{aligned} \tag{5.56}$$

Se $\Delta_i \geq 0$, então:

$$\begin{aligned}\Delta_{i+1} &= a^2(y_{i+1}+1/2)^2 + b^2(x_{i+1}+1)^2 - a^2b^2 \\ &= a^2[(y_i+1/2)+1]^2 + b^2[(x_i+1)+1]^2 - a^2b^2 \\ &= a^2(y_i+1/2)^2 + b^2(x_i+1)^2 - a^2b^2 + a^2[2(y_i+1/2)+1] + 2b^2x_{i+1} + b^2 \\ &= \Delta_i + 2a^2y_{i+1} + 2b^2x_{i+1} + b^2\end{aligned} \tag{5.57}$$

Nas equações (5.56) e (5.57) ainda há o inconveniente de um produto para o processamento computacional, isto pode ser evitado da seguinte maneira:

faça-se:

$$u_i = 2b^2x_i + b^2 \tag{5.58}$$

$$v_i = 2a^2y_i \tag{5.59}$$

então, se $\Delta_i < 0$

$$u_{i+1} = 2b^2x_{i+1} + b^2$$
$$= 2b^2(x_i + 1) + b^2$$
$$= 2b^2x_i + b^2 + b^2$$
$$= u_i + 2b^2 \qquad (5.60)$$

$$v_{i+1} = 2a^2y_{i+1}$$
$$= 2a^2y_i$$
$$= v_i \qquad (5.61)$$

e se $\Delta_i \geq 0$

$$u_{i+1} = 2b^2x_{i+1} + b^2$$
$$= 2b^2(x_i + 1) + b^2$$
$$= u_i + 2b^2 \qquad (5.62)$$

$$v_{i+1} = 2a^2y_{i+1}$$
$$= 2a^2(y_i + 1)$$
$$= 2a^2y_i + 2a^2$$
$$= v_i + 2a^2 \qquad (5.63)$$

Utilizando as eq. (5.60), (5.61) e (5.63), é possível transformar (5.56) e (5.57) da seguinte forma:

se $\Delta_i < 0$

$$\Delta_{i+1} = \Delta_i + u_i + 2b^2 \qquad (5.64)$$

e se $\Delta_i \geq 0$

$$\Delta_{i+1} = \Delta_i + u_i + v_i + 2a^2 + 2b^2 \qquad (5.65)$$

Iniciando-se o algoritmo no ponto A(0,-b), figura 5.12, tem-se para valores dos parâmetros iniciais:

$$\Delta_0 = b^2 - a^2b + a^2/4 \qquad (5.66)$$

$$u_0 = b^2 \qquad (5.67)$$

$$v_0 = 2a^2(-b) \qquad (5.68)$$

Levando em consideração que os cálculos vistos até aqui são para a parte da curva em que a tangente tem coeficiente menor que um, não possível então se obter dois valores consecutivos para Δ_i maiores que zero, pois isto implicaria necessariamente numa tangente de coeficiente maior que um. A figura 5.14, ilustra bem este fato.

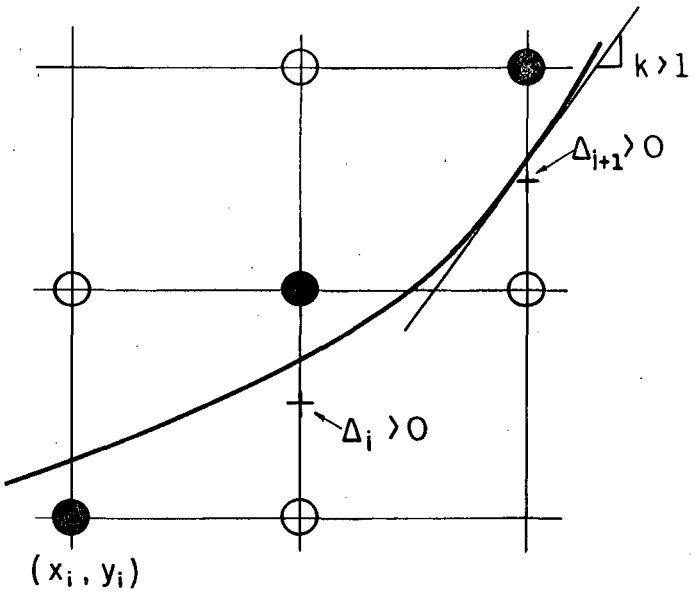


FIGURA 5.14 - Aspecto da curva para que se tenham dois D_i's seguidos maiores que zero.

Sendo assim, dois valores positivos, não nulos, consecutivos para a variável Δ indicará que a curva já atingiu o ponto *B* (figura 5.12) e o seu traçado deve ser encerrado. Este fato determina a decisão, ao final do "loop", pelo encerramento do mesmo em função do sinal de Δ.

De uma maneira mais clara; o algoritmo tomaria a seguinte forma:

```
x ← 0
y ← -b
u ← b*b
v ← -2*a*a*b
Δ ← a*a/4 - a*a*b
loop
    SetPixel(x_o+x,y_o+y)
    x ← x + 1
    Δ ←  Δ + u
    u ← u + 2*b*b
    Se Δ maior que 0, faça
        y ← y + 1
        v ← v + 2*a*a
        Δ ← Δ + v
    fim se
enquanto Δ for menor ou igual a 0
```

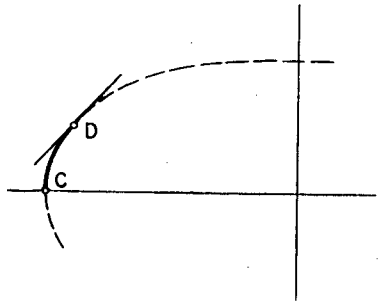


FIGURA 5.15 - Arco de elipse com coeficiente da reta tangente maior que um.

O mesmo raciocínio descrito até aqui pode ser utilizado na confecção do trecho CD de arco de elipse, (figura 5.15). Obtém-se então:

$$\Delta_0 = a^2 - ab^2 + b^2/4 \tag{5.69}$$

$$u_0 = 2b^2(-a) \tag{5.70}$$

$$v_0 = a^2 \tag{5.71}$$

e quando $\Delta_i < 0$,

$$x_{i+1} = x_i \tag{5.72}$$

$$y_{i+1} = y_i + 1 \tag{5.73}$$

$$u_{i+1} = u_i \tag{5.74}$$

$$v_{i+1} = v_i + 2a^2 \tag{5.75}$$

$$\Delta_{i+1} = \Delta_i + v_{i+1} \tag{5.76}$$

se $\Delta_i \geq 0$

$$x_{i+1} = x_i + 1 \tag{5.77}$$

$$y_{i+1} = y_i + 1 \tag{5.78}$$

$$u_{i+1} = u_i + 2b^2 \tag{5.79}$$

$$v_{i+1} = v_i + 2a^2 \tag{5.80}$$

$$\Delta_{i+1} = \Delta_i + u_{i+1} + v_{i+1} \tag{5.81}$$

Com isto, é possível que se complete toda a elipse fazendo-se uso da simetria da mesma.

Com a utilização deste critério, é possível também obter-se a conversão da elipse de eixos principais não paralelos aos eixos coordenados, figura 5.16. Apenas deve-se ter o cuidado de selecionar regiões com derivadas (inclinação da tangente) menores e maiores que um separadamente, e observar que ainda se tem uma figura simétrica em relação ao seu centro.

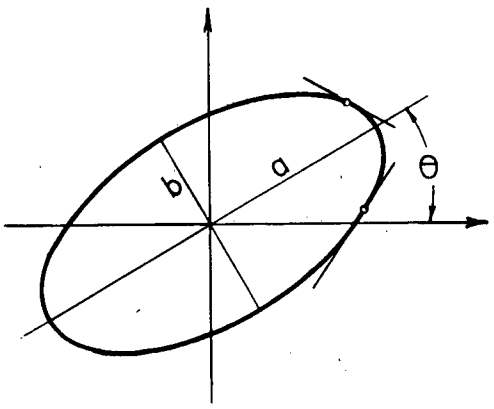


FIGURA 5.16 - Elipse de eixos principais não paralelos aos eixos coordenados.

CAPÍTULO 6

Transformações Geométricas

6.1 Introdução

Sem dúvida, o fato da "Computação Gráfica" estar se firmando cada vez mais entre os Projetistas, Arquitetos e Engenheiros, se deve, em grande parte, à facilidade de manipulação de desenhos e alteração de suas características, tais como escala, orientação, cizalhamento, etc., por parte do usuário numa iteração rápida e direta com o computador.

As operações que permitem uma alteração na forma do desenho armazenado no computador são chamadas [7] *transformações geométricas*. São, por assim dizer, operações matemáticas que permitem alterações uniformes de uma imagem definida sobre um sistema de coordenadas. Estas alterações não afetam a estrutura do desenho, mas sim o aspecto que ele vai assumir ao ser apresentado sob diferentes orientações e escalas.

O "zmSAG" permite transformações geométricas apenas sobre segmentos, os quais (vide capítulo 4) são formados por primitivas gráficas que têm como parâmetros as coordenadas cartesianas de pontos no espaço. As transformações agem sobre estas coordenadas de modo que, ao se desenhar novamente o segmento com as novas coordenadas transformadas, o desenho também ficará transformado.

Para que se efetúe uma transformação qualquer, o "zmSAG" prevê uma série de funções para manipulação de matrizes (vide apêndice C) e inserção de segmentos.

6.2 Transformações em 2D

Qualquer ponto P(x,y), no plano *xy* (figura 6.1), pode ser transladado a uma nova posição P'(x,y) pela simples adição de valores constantes a cada uma de suas coordenadas, ou seja:

$$x' = x + dx$$
$$y' = y + dy \qquad (6.1)$$

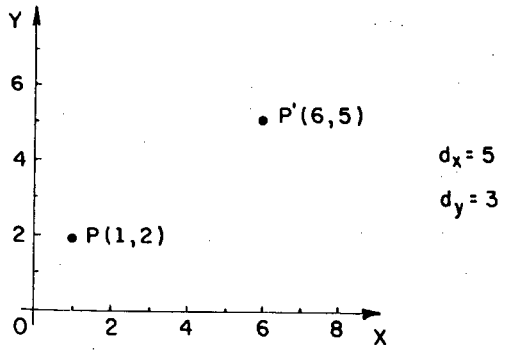


FIGURA 6.1 - Exemplo de translação em 2D com d_x=5 e d_y=3

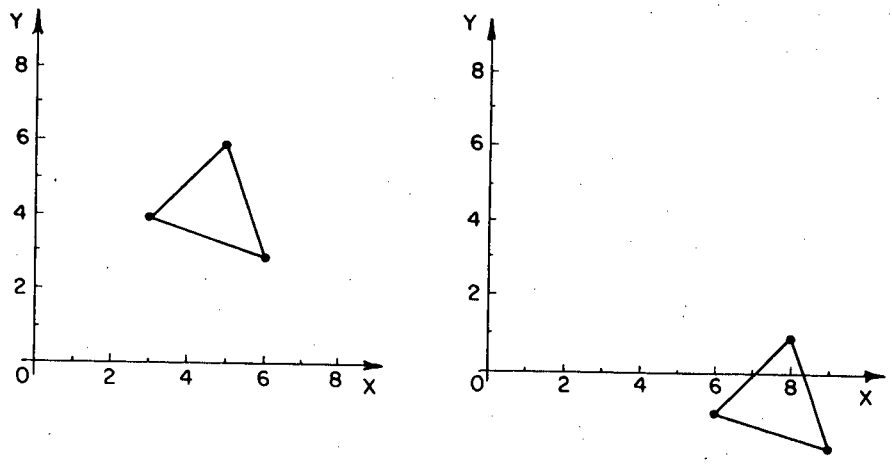


FIGURA 6.2 - Exemplos de translação de objetos (a) desenho original, (b) após transformação com d_x = 3 e d_y = -5.

Assim, um objeto qualquer pode ser transladado aplicando-se (6.1) a cada um de seus pontos. Este processo poderá tornar-se longo se for aplicado, por exemplo, a cada ponto de um segmento de reta. Felizmente, todos os pontos deste segmento podem ser movidos efetuando-se apenas a translação dos dois pontos de sua extremidade e os ligando novamente. Isto é também verdadeiro para escalonamento e rotação de objetos [1]. A figura 6.2 mostra alguns exemplos de transformação de translação.

Define-se o escalonamento de um ponto P(x,y) ao longo do eixo x, por P'(s_x·x,y) sendo s_x chamado "fator de escalonamento em x". Da mesma forma tem-se o escalonamento em y dado por (x,s_y·y). Generalizando, o escalonamento ao longo dos dois eixos sobre um ponto P(x,y) pode ser obtido [1] com a utilização da matriz:

$$S = \begin{bmatrix} S_x & 1 \\ 1 & S_y \end{bmatrix} \qquad (6.2)$$

da seguinte forma:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 1 \\ 1 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad (6.3)$$

ou de forma genérica

$$P' = S.P \qquad (6.4)$$

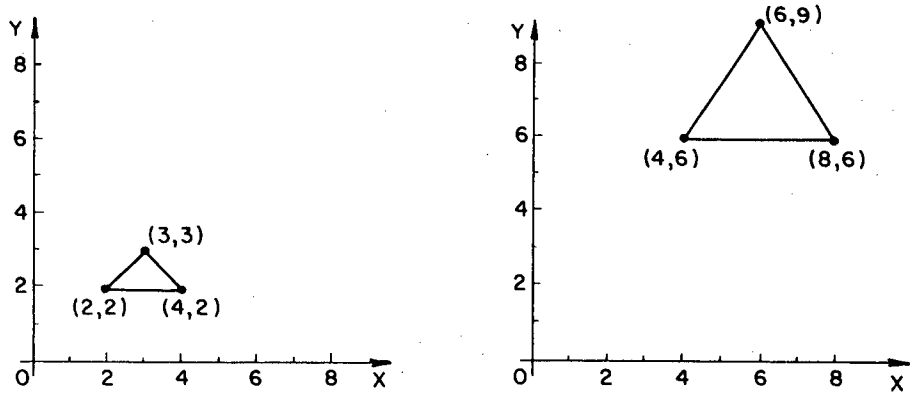


FIGURA 6.3 - Exemplo de escalonamento com $S_x = 2$ e $S_y = 3$.

Um ponto em R^2 também pode ser rotacionado de um ângulo θ em torno da origem, como ilustrado na figura 6.4. Esta rotação pode ser definida matematicamente por:

$$\begin{aligned} x' &= x.\cos\theta - y.\sen\theta \\ y' &= x.\sen\theta + y.\cos\theta \end{aligned} \qquad (6.5)$$

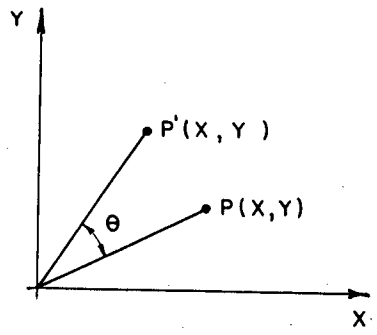


FIGURA 6.4 - Exemplo de rotação do ponto P, de um ângulo θ em torno da origem.

ou matricialmente:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta \\ \text{sen}\,\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} y \\ y \end{bmatrix} \tag{6.6}$$

6.2.1 Coordenadas Homogêneas

Até o presente momento foi discutido o escalonamento, a rotação e translação de pontos em duas dimensões, embora existam outros casos de transformações em 2D; estes são os mais interessantes numa abordagem simplificada. Apesar de simples, estes três casos não permitem uma simplificação matricial generalizada com utilização de matrizes 2x2, pois não é possível a introdução de constantes de translação.

Este problema poderá ser resolvido facilmente [11] pela introdução de uma terceira componente aos vetores (x,y) e (x',y'), convertendo-os à forma (x,y,1) e (x',y',1). Para a translação, a matriz de transformação torna-se então:

$$M = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.7}$$

e a transformação de translação pode agora ser reescrita matricialmente:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ x \\ 1 \end{bmatrix} = \begin{bmatrix} x+d_x \\ x+d_y \\ 1 \end{bmatrix} \tag{6.8}$$

Veja que isto só foi possível com a introdução de uma terceira coordenada em E^2. Matematicamente [10,11] a definição para coordenadas homogêneas, no espaço euclidiano E^2, tem a seguinte forma:

> definição: quando $h \neq 0$, (x,y,h) são as coordenadas homogêneas de um ponto ordinário (x/h,y/h) no espaço euclidiano bidimensional E^2.
>
> Se (x',y') e (x,y,h) estão homogeneamente relacionados, pode-se escrever $(x',y') \approx (x,y,h)$.

Ao se expressar os pontos de um determinado objeto em coordenadas homogêneas, as três transformações relatadas acima podem ser tratadas como simples produto de matrizes e goza de todas as propriedades existentes no cálculo matricial.

Estas três transformações têm para suas matrizes o seguinte:

i. transformação de escalonamento,

$$M_S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.9}$$

ii. transformação de translação,

$$M_T = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.10}$$

iii. transformação de rotação,

$$M_R = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta & 0 \\ \text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.11}$$

6.3 Composição de transformações

A composição de transformações pode ser bastante simplificada utilizando-se das propriedades do cálculo de matrizes. Suponha-se um ponto de coordenadas (x,y,1), sobre o qual aplica-se a matriz de transformação M_1, obtendo-se o ponto $(x_1,y_1,1)$.

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = M_1 \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.12}$$

Suponha-se agora que sobre este ponto aplica-se a transformação M_2 obtendo-se $(x_2,y_2,1)$.

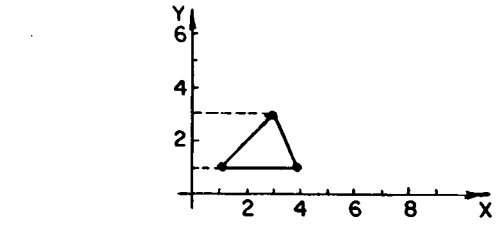
$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = M_2 \cdot \begin{bmatrix} x_2 \\ y_1 \\ 1 \end{bmatrix} \tag{6.13}$$

É possível se demonstrar, utilizando-se das propriedades do cálculo matricial, que existe a matriz M_3 que aplicada a (x,y,1) irá fornecer diretamente $(x_2,y_2,1)$. Esta matriz é obtida simplesmente como o produto de M_2 por M_1, nesta ordem.

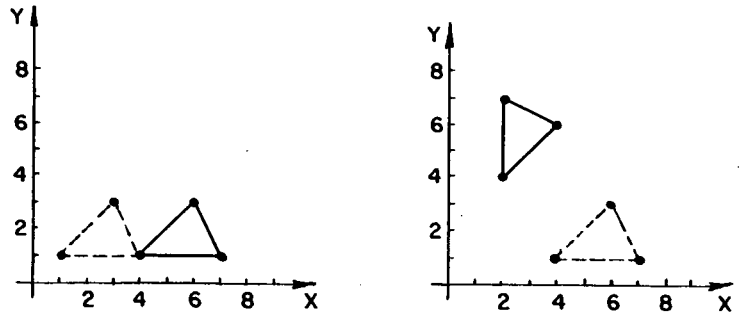
$$M_3 = M_2 . M_1 \qquad (6.14)$$

Com isto é possível se obter, por exemplo, em uma só matriz, uma transformação que seja ao mesmo tempo translação, rotação e escalonamento.

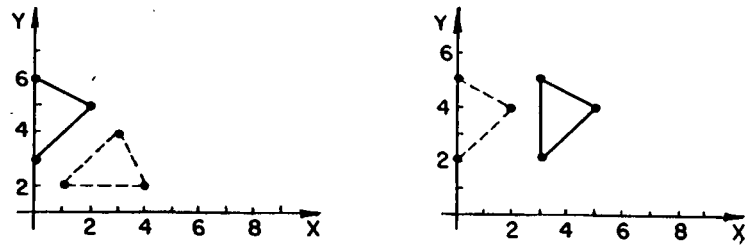
É de se notar, entretanto, que não sendo admitida a propriedade comutativa para o produto de matrizes, a ordem em que as transformações são aplicadas sobre a imagem, influencia diretamente no resultado final. Sendo assim, transladar um ponto e depois rotacioná-lo, não é o mesmo que rotacioná-lo e depois transladá-lo. Este fato é melhor visto ilustrativamente na figura 6.5 a seguir.



a) Desenho original



b) Translação ($T_X = 3, T_Y = 0$) seguida de rotação ($\alpha = 45°$)



c) Rotação ($\alpha = 45°$) seguida de translação ($T_X = 3, T_Y = 0$)

FIGURA 6.5 - Efeito da combinação de transformações em diferentes ordens.

6.4 Transformações no "zmSAG"

O "zmSAG" tira proveito, exclusivo, da composição de matrizes para efetuar as seguintes transformações:

i escalonamento em relação a um ponto arbitrário - se constitui numa transformação de escalonamento mais genérica, onde se fixa um ponto arbitrário em relação ao qual o desenho será escalonado.

ii rotação em torno de um ponto arbitrário - a transformação aqui irá rotacionar o objeto, não em torno da origem, mas sim de um ponto de referência predeterminado.

Em ambos os casos a transformação é obtida pela combinação de três outras: translação do ponto de referência à origem, rotação ou escalonamento (conforme seja o caso) em torno da origem e, finalmente, translação da origem para o ponto de referência.

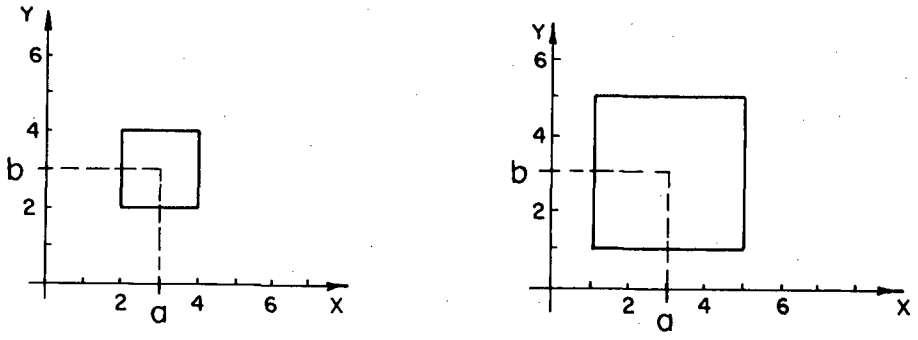


FIGURA 6.6 - Exemplo de escalonamento do objeto em relação ao seu centro.

Seja, por exemplo, um objeto que tenha uma forma quadrada (figura 6.6), para um escalonamento em relação ao seu centro, a matriz de transformação será dada por:

$$M = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} S_x & 0 & a(1-S_x) \\ 0 & S_y & b(1-S_y) \\ 0 & 0 & 0 \end{bmatrix} \qquad (6.15)$$

Onde se nota que houve primeiramente uma translação, dos pontos de referência (a,b), à origem do sistema de coordenadas, para só então se efetuar o escalonamento, evitando-se assim deformações em direções não desejadas, e finalmente após o escalonamento, se efetua a translação de volta ao ponto de origem.

No "zmSAG", a exemplo da norma GKS [18], foi definida a função *evaluate_transformation_matrix* que, recebendo os argumentos abaixo relacionados, devolve a matriz de transformação:

$$M = M_5.M_4.M_3.M_2.M_1 \tag{6.16}$$

com:

$$M_1 = \begin{bmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{bmatrix} \tag{6.17}$$

$$M_2 = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.18}$$

$$M_3 = \begin{bmatrix} \cos\theta & -\text{sen}\,\theta & 0 \\ \text{sen}\,\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.19}$$

$$M_4 = \begin{bmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{bmatrix} \tag{6.20}$$

$$M_5 = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.21}$$

argumentos passados à função *evaluate transformation matrix*

a,b - coordenadas de um ponto de referência, em torno do qual será feito o escalonamento e/ou rotação.

d_X,d_Y - deslocamento segundo os eixos coordenados *x* e *y* respectivamente.

S_X,S_Y - fatores de escalonamento segundo *x* e *y* respectivamente.

teta - ângulo de rotação no sentido anti-horário.

Um fato importante a ser notado em (6.16), uma vez que a ordem influencia, é que primeiro é efetuado o escalonamento, seguido da rotação e por último o deslocamento.

Após composta, a matriz M final tem a seguinte forma:

$$M = \begin{bmatrix} S_x\cos\theta & -S_y\operatorname{sen}\theta & [(a+d_x) - (S_x a\cos\theta - S_y b\operatorname{sen}\theta)] \\ S_x\operatorname{sen}\theta & S_y\cos\theta & [(b+d_y) - (S_x a\operatorname{sen}\theta + S_y b\cos\theta)] \\ 0 & 0 & 1 \end{bmatrix} \qquad (6.22)$$

Note que a matriz composta não necessariamente deverá efetuar as três transformações a que se propõe. Se por exemplo, todos os argumentos passados à *evaluate_transformation_matrix* forem nulos[1], com exceção de d_X e d_Y, a matriz retornada efetuará apenas a transformação de deslocamento.

Perceba-se ainda que com esta função é possível compor qualquer transformação no espaço euclidiano bidimensional (E^2).

6.5 Considerações de Eficiência

A composição de transformações do parágrafo anterior, em duas dimensões, vai produzir, na pior das hipóteses, uma matriz quadrada de ordem 3 que tem a forma:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \qquad (6.23)$$

com:

$$a_{ij} \neq 0,\ i = 1,2\ ;\ j = 1,3$$

(1) Quando os argumentos S_x e S_y são nulos, a função *evaluate_transformation_matrix* os transforma imediatamente em valores unitários

Para um ponto P(x,y,1) em coordenadas homogêneas, o produto M.P vai produzir nove operações de multiplicação e seis de adição. O fato de a última linha da matriz em (6.23) ser sempre constante simplifica as operações de forma bastante prática, como a seguir:

$$\begin{aligned} x' &= a_{11}.x + a_{12}.y + a_{13} \\ y' &= a_{21}.x + a_{22}.y + a_{23} \end{aligned} \qquad (6.24)$$

reduzindo o processo a quatro multiplicações e quatro adições. O tempo ganho é por demais significante, especialmente em se tratando de operações sobre segmentos que normalmente são compostos por centenas ou milhares de pontos por figura. Conclui-se então, que embora o tratamento matricial seja conveniente e muito utilzado na composição em 2D, no processo computacional deverá haver uma simplificação nos cálculos tirando-se vantagem da estrutura da matriz (6.23).

CAPÍTULO 7

Dispositivos Gráficos de Saída

7.1 Introdução

O núcleo do "zmSAG" é formado por um conjunto de procedimentos tais, que se pode dispensar as subrotinas oriundas dos dispositivos externos (com exceção da função SetPixel que será vista mais adiante), para geração de primitivas gráficas nos mesmos. Este fato permite concluir então, que o "zmSAG" independe dos dispositivos externos, tendo total portabilidade de hardware para os periféricos de saída. Isto é verdade, desde que o dispositivo permita a construção de uma pequena interface (normalmente em linguagem "C") que o associe ao núcleo do "zmSAG", e que será composta apenas pela função SetPixel.

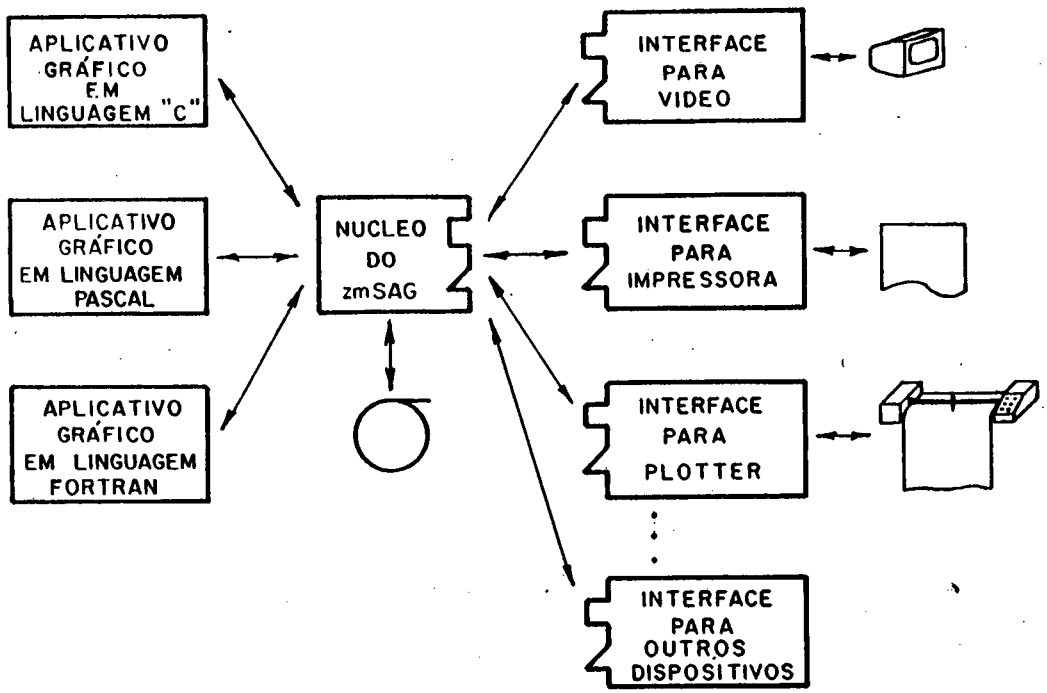


FIGURA 7.1 - Iteração do núcleo do "zmSAG" com os diversos módulos e periféricos

A figura 7.1 esquematiza graficamente, a independência do núcleo gráfico com os periféricos.

Até o presente momento, já existem as interfaces para vídeo, impressora e traçador gráfico. Para o vídeo usa-se o padrão IBM-PC compatível utilizando as placas gráficas CGA, EGA, VGA e Hércules. No caso da impressora, foi usado o padrão EPSON; e para plotter o padrão SUMMGRAPHICS adotado pela DIGICOM, uma das principais fornecedoras deste tipo de periférico no Brasil.

A construção e uso de cada interface é feita de uma forma muito simples, por exemplo, no caso do vídeo foi construída a função vd$_pset, em liguagem "C", que recebe os argumentos *x* e *y* e seta no vídeo o pixel de coordenadas (x,y). No caso da impressora tem-se a função pr$_pset que funciona da mesma forma setando um ponto na impressora.

Após a seleção, através de um comando do usuário do dispositivo a ser utilizado, um ponteiro de função (*pset)(x) (característica existente apenas na linguagem "C") é apontado para a função vd$_pset ou pr$_pset, dependendo do dispositivo escolhido ter sido o vídeo ou a impressora. A partir de então o núcleo do "zmSAG" utiliza-se apenas do comando pset(x,y), não lhe interessando em que dispositivo está atuando.

É necessário deixar claro aqui que embora a forma de interfacear os periféricos externos exposta acima tenha sido colocada de uma forma simplificada para um melhor entendimento do leitor, existirão casos em que apenas uma simples interface do tipo "SetPixel", apesar de funcionar muito bem, não terá uma performance satisfatória considerada com relação a performance dos procedimentos dos dispositivos; isto se deve principalmente à baixa velocidade na transmissão e recebimento de dados entre o "zmSAG" e o dispositivo. Um caso típico é o do traçador gráfico (plotter) que comunicando dados através de uma interface serial, não pode ultrapassar a velocidade de transmissão e recepção de 9600 bits por segundo, o que é extremamente lento quando se trata de processamento gráfico. Sendo assim, para dispositivos deste tipo, se faz necessária a confecção de uma interface mais elaborada, preservando-se contudo, o princípio do ponteiro de função para que se tenha uma resposta rápida quando do tratamento de um outro periférico de saída.

7.2 Terminal de vídeo

A tendência atual dos sistemas de CAD [3] segue para os terminais de vídeo, CRT'S de varredura, para aplicações gráficas de um modo geral e, principalmente quando se necessita de um certo dinamismo de imagens.

Os terminais de varredura são dispositivos tipicamente matriciais a partir de seu hardware, pois na sua maioria permitem um endereçamento do "pixel" diretamente. Isto simplifica sobremaneira a construção da interface com o "zmSAG", tornando muito simples a função SetPixel. Sendo assim, será interessante aqui abordar apenas a parte desta interface que agiliza o processo de movimentação, sobreposição e troca de imagens no terminal de vídeo.

7.2.1 Tratamento de imagens bit a bit

A manipulação de primitivas gráficas através dos algoritmos tradicionais, por construir a figura iterativamente ponto a ponto, não é satisfatória quando se pretende dar a idéia de movimento à imagem gerada. Com o uso dos algoritmos convencionais, se faria necessário apagar a primitiva anterior e construir uma outra, de mesmas proporções à sua frente. Isto ocasionaria um efeito piscante na tela do vídeo, prejudicando a idéia de movimento.

Os terminais de vídeo apresentam a imagem em sua tela após decodificar uma área da memória RAM, conhecida como "buffer de vídeo" (frame buffer). Sendo assim, tomando-se um segmento de memória dentro deste "buffer de vídeo" e o recolocando mais a frente, obtém-se a mesma imagem que este segmento representava, agora em outra parte da tela. O "zmSAG" explora esta facilidade de várias formas:

- Construindo uma imagem em outro local da memória RAM, para depois sobrepô-la em uma determinada parte do "buffer de vídeo";
- Copiando um determinado segmento do "buffer de vídeo" para uso posterior;
- Movendo para um segmento do "buffer de vídeo", bytes nulos, dando a idéia de "limpeza" de uma determinada região da tela de vídeo.

O procedimento de construção de uma imagem fora do "buffer de vídeo", utiliza uma função "SetPixel" redirecionada para um ARRAY de caracteres da mesma forma utilizada para impressora, com utilização do vetor BPR que pode ser visto no item 7.3.1 mais à frente.

A transferência da imagem para a área do buffer de vídeo no "zmSAG", pode ser feita de três maneiras:

Por Sobreposição - Os bytes transferidos são colocados em cima dos já existentes, eliminando os mesmos.

Pela Operação Logica OR - Efetuando-se a operação lógica OR bit a bit dos bytes a serem transferidos com os já existentes.

Pela Operação Lógica XOR - Efetuando-se a operação lógica XOR bit a bit dos bytes a serem transferidos com os já existentes.

O efeito obtido por cada uma dessas três operações pode ser melhor entendido observando-se a figura 7.2, na ordem em que são apresentadas.

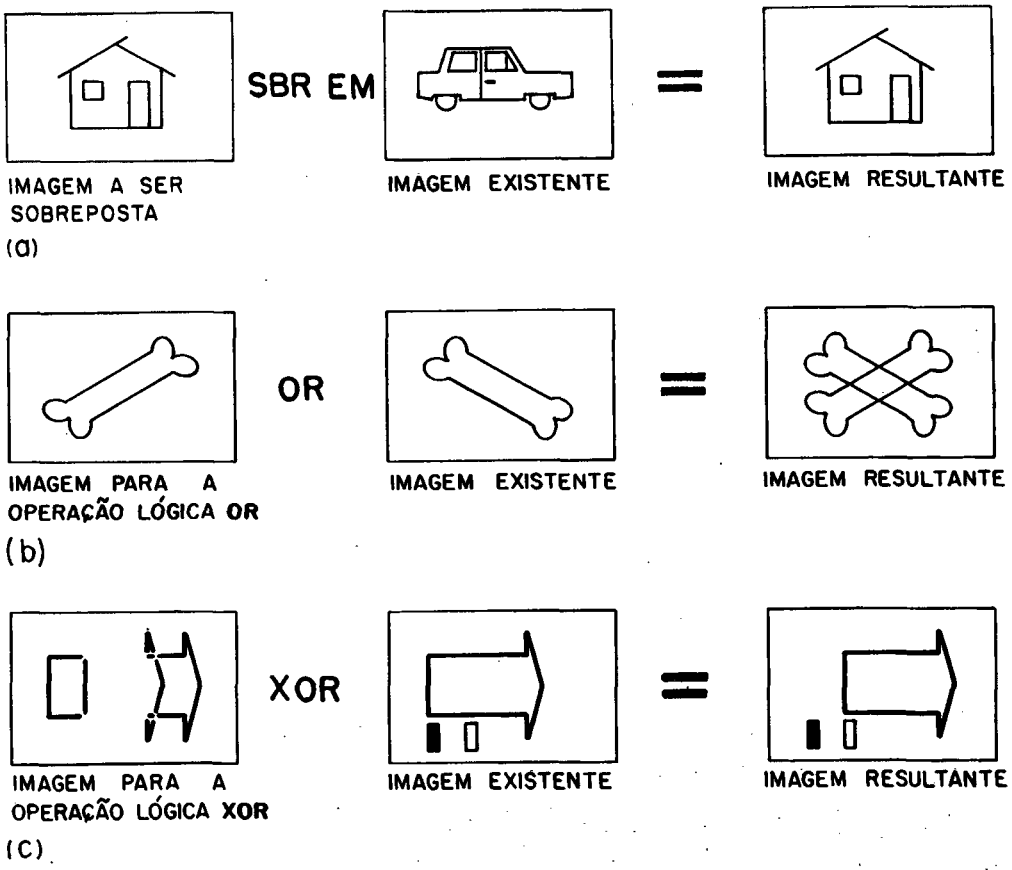


FIGURA 7.2 - Resultado da sobreposição de imagem em (a), da operação lógica OR em (b) e da operação XOR em (C).

O modo XOR tem grande utilização na movimentação ("draging") de imagens na tela. Para um entendimento melhor deste modo veja o efeito de sua operação no quadro 7.1.

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

QUADRO 7.1 - Operação lógica XOR bit a bit

Para que o efeito de movimentação fique o mais real possível, deve ser feito um estudo da imagem a ser movimentada a fim de se construir a imagem a ser sobreposta (com XOR), levando-se em consideração a tabela verdade do quadro 7.1. A figura 7.2c dá uma idéia de como isto deve ser feito.

Os procedimentos encarregados do tratamento de imagem bit a bit podem ser encontrados no código fonte original do "zmSAG" no módulo "vídeo.c".

7.3 A Impressora

7.3.1 O Sistema EPSON de Impressão

Apesar da pouca eficiência, em termos de resolução de pontos, as impressoras gráficas do tipo matriciais são largamente utilizadas em sistemas CAE/CAD e na computação gráfica de um modo geral, principalmente quando da necessidade de obtenção de listagens em meio à confecção do projeto ou de "hardcopies" de fases do mesmo.

No "zmSAG", a interface de impressora desenvolvida é específica para o padrão EPSON [23], porém a adaptação para outros padrões de impressão poderá ser feita com facilidade para qualquer impressora gráfica tipo matricial, pois a teoria

utilizada, como se vê a seguir, é bastante simples e se aplica a qualquer dispositivo do tipo "hardcopy" que mapeie bit a bit.

7.3.2 Densidade de Impressão

O padrão EPSON apresenta quatro possíveis opções de densidade de impressão no modo gráfico: na primeira imprimem-se 72 pontos por polegada na vertical e 60 pontos por polegada na horizontal. Na densidade dupla são impressos 72 pontos por polegada na vertical e 120 pontos por polegada na horizontal. Na densidade *tripla* e *quádrupla*, tem-se 144 x 120 e 144 x 240 respectivamente, como resolução por polegada quadrada.

A seleção de uma dessas resoluções é feita pelo envio à impressora de uma seqüência de caracteres de contrôle. Assim para ao se efetuar a seleção "simples densidade", por exemplo, seria necessário o envio da seguinte seqüência:

$$1Bh, 4Bh, N_1, N_2$$

Onde N_1 e N_2 fornecerá, como se verá mais adiante, o número máximo de colunas possíveis. Em liguagem BASIC, esta seqüência seria remetida com o seguinte comando:

```
PRINT CHR$(27);CHR$(75);CHR$(N1);CHR$(N2);
```

Só após esta seleção é que se poderá enviar os caracteres que controlarão a impressão de pontos em cada linha.

Por razões de ordem prática, o "zmSAG" se utiliza preferencialmente do modo de impressão "simples densidade", tendo uma resolução de 72 x 60 pontos por polegada quadrada.

7.3.3 Matriz de Impressão

As impressoras matriciais têm na sua "cabeça" de impressão oito agulhas dispostas na vertical (figura 7.3), que podem ser acionadas individualmente, ou em conjunto impressionando o papel através de uma fita umedecida com o líquido de impressão.

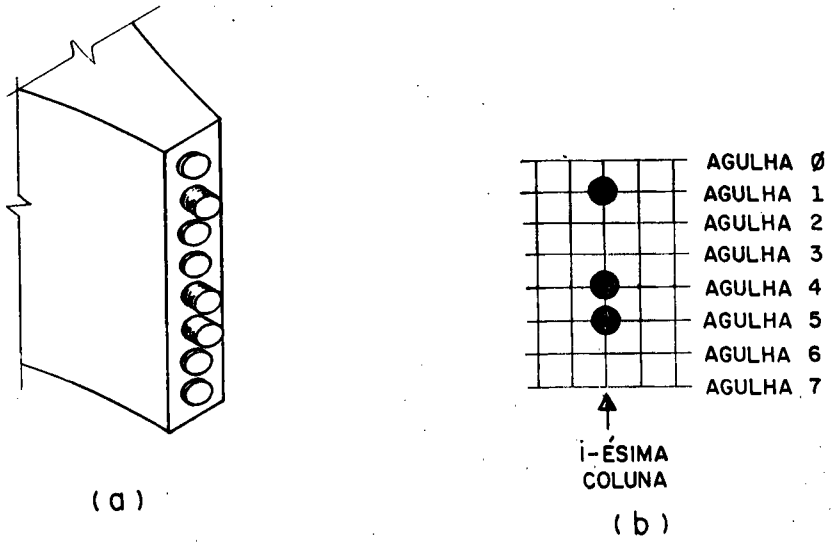


FIGURA 7.3 - Formato da cabeça de impressão (a) e impressão deixada (b) se acionada como em (a) na i-esima coluna.

Dois motores de passo efetuam os movimentos; horizontal da cabeça de impressão e de giro do rolo que contém o papel. Os intervalos de deslocamento, na densidade simples são de 1/72 da polegada na horizontal e de 10/75 da polegada na vertical, sendo que para cada passo vertical podem ser impressos oito pontos.

Pelo exposto acima, percebe-se que existem "linhas de varredura" contendo oito sub-linhas que são formadas pelas agulhas. Uma vez que o motor de passo do rolo não permite retorno, devem ser previsto de antemão todos os pontos a serem colocados na linha antes que seja efetuada a liberação de seus caracteres de controle à impressora. Isto faz com que não se possa endereçar, por hardware, individualmente cada ponto, que é a característica básica de um dispositivo matricial. Contudo, pode-se classificar a impressora como dispositivo matricial, devendo-se isto ao fato de, através de software, se poder simular um endereçamento ponto a ponto. Isto é o que será visto nas próximas seções.

7.3.4 O Endereçamento Ponto a Ponto

7.3.4a Com o uso de ARRAY

Foi visto no capítulo 5 que uma das características principais de um dispositivo matricial é a possibilidade de acesso ao atributo de cada "pixel" através dos procedimentos:

SetPixel (x, y)

e:

GetPixel (x, y)

O objetivo aqui, é simular estes procedimentos, através de software, para a impressora.

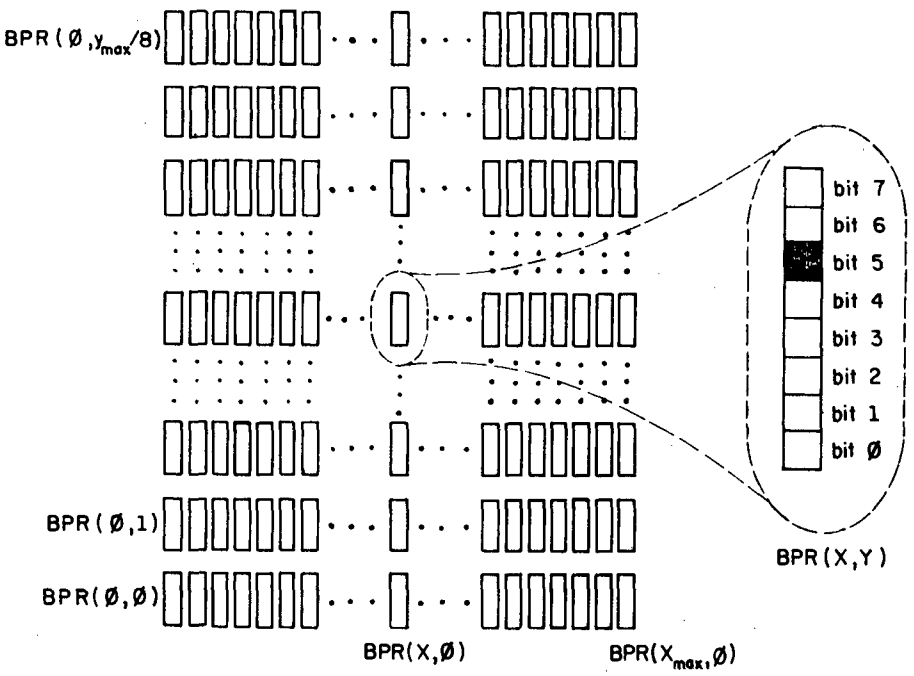


FIGURA 7.4 - Simulação do sistema gráfico da impressora através do array BPR().

Desejando-se trabalhar com uma resolução de x_{max} x y_{max} pontos na impressora. Pode ser utilizada uma área de memória (buffer) para comportar esta resolução através do ARRAY de caracteres:

$$BPR\ (x_{max}, y_{max} / 8)$$

Cada caracter é composto por uma variável do tipo byte. A figura 7.4 permite que se entenda melhor a disposição deste ARRAY, mostrando-o de uma forma gráfica.

Se por exemplo, se fizer necessário acessar o ponto de coordenadas (x,y), o elemento do array a ser modificado será:

$$BPR\ (a, b)$$

com:

$$a = x \tag{7.1}$$

e:

$$b = \begin{cases} y/8 & \text{se (resto da divisão de y por 8) = 0} \\ y/8 + 1 & \text{se (resto da divisão de y por 8) } \neq 0 \end{cases} \tag{7.2}$$

Após isto, o i-ésimo bit do elemento BPR (a,b) do array pode ser configurado, efetuando-se a operação lógica "OR" bit a bit entre ele e a variável AUX = 128 (1000 0000 em binário) deslocada de uma quantidade igual ao resto da divisão de *y* por *8* para a direita. Em linguagem "C", isto poderia ser feito da seguinte forma:

```
BPR(a,b) = BPR(a,b) || (AUX >> (y mod 8))
```

Onde a expressão *y mod 8* representa o resto da divisão de *y* por 8 e as duas barras o operador XOR nesta linguagem.

Caso não se disponha na linguagem em uso do operador binário ">>" pode-se fazer uso do array auxiliar POT_2, como no algoritmo abaixo, onde já é suposto existir, globalmente, o array POT_2(8), sendo:

```
POT_2 (0) = 128
POT_2 (1) = 64
POT_2 (2) = 32
POT_2 (3) = 16
POT-2 (4) = 8
POT_2 (5) = 4
POT_2 (6) = 2
POT_2 (7) = 1

procedimento SetPixel (x, y)
    b ← y/8
    resto  y mod 8
    Se resto ≠ 0, então
        b ← b + 1
    fim se
    BPR (x, b) ← BPR (x, b) or POT_2 (resto)
fim procedimento
```

O procedimento acima, juntamente com o array global BPR() permite que se transforme a impressora num dispositivo gráfico matricial. Infelizmente, este método requer o uso de uma grande área de memória que será ineficientemente utilizada, como é caso dos elementos do array, que não serão utilizados por representarem uma seqüência de oito pontos desativados. Mesmo utilizando o modo gráfico "simples densidade" da impressora, em uma página de impressão normal tem-se um total de 576 pontos na horizontal por 660 na vertical, e seria necessário dimensionar BPR(576,83) gastando um total de 47.808 bytes, dos quais mais de 70% não seriam utilizados quando da impressão normal de primitivas gráficas. Estes bytes todos só seriam necessários se a impressão fosse obtida em modo reverso.

7.3.4b Com utilização de listas encadeadas

O "zmSAG" se utiliza do recurso de alocação dinâmica de memória, existente na linguagem "C", para construir um "buffer" de impressão que será utilizado de forma a não conter elementos nulos. Esta alocação é feita em cima do conceito de listas encadeadas (veja apêndice D), definindo-se a estrutura na qual a área de informação tem dois campos e apenas um elo de ligação.

```
Struct t_col {
    int             col;
    unsigned char   byte;
    struct t_col    *prox;
}
```

O nó cabeça é composto por um array unidimensional de ponteiros para uma estrutura do tipo t_col, que representa cada linha de impressão. A configuração final do buffer, após codificadas todas as primitivas gráficas no mesmo, pode ser vista esquematicamente no desenho da figura 7.5.

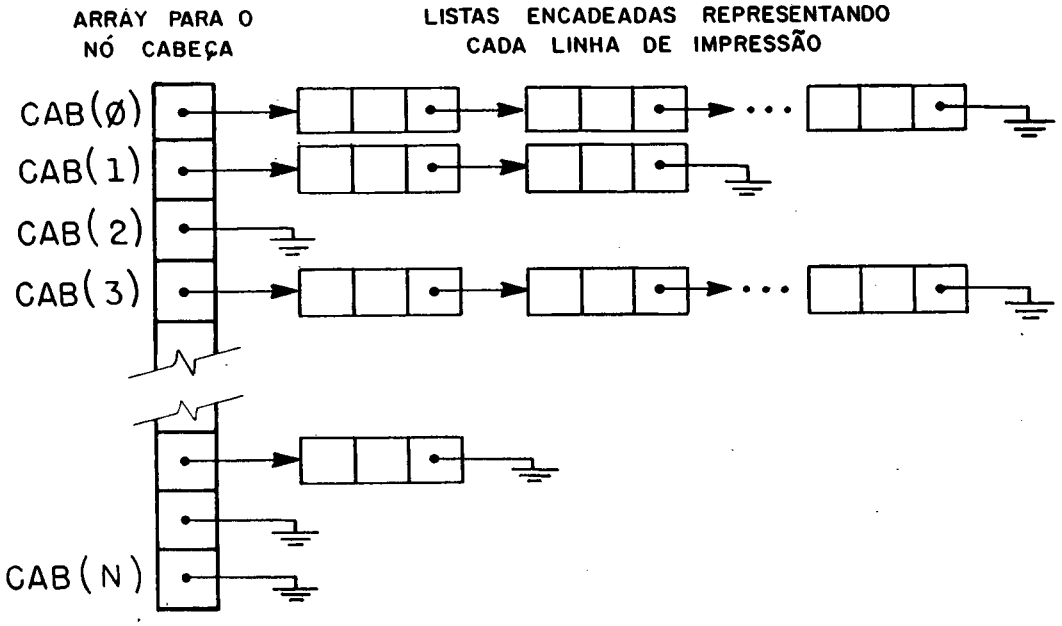


FIGURA 7.5 - Configuração do buffer de impressão utilizando listas encadeadas.

Na estrutura t_col, a variável col representa a coluna na qual a variável byte deverá ser impressa. Por exemplo, uma seqüência de impressão composta pelo pequeno trecho da lista encadeada da figura 7.6, informaria ao procedimento de impressão o seguinte:

- imprima o caractere de código 128 na coluna 8.
- imprima o caractere de código 79 na coluna 22.
- imprima o caractere de código 05 na coluna 30.

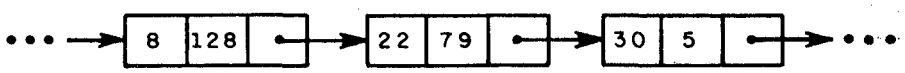


FIGURA 7.6 - Pequeno trecho de uma seqüência de impressão.

O procedimento "SetPixel" completo se encontra no código fonte original do "zmSAG" com o nome de pr$_pset no módulo "printer.c". Lá também poderá ser visto o procedimento send_draw que envia essa estrutura de dados à impressora.

CAPÍTULO 8
Um Exemplo de Aplicação

8.1 Introdução

Visando um perfeito entendimento da estrutura básica do Sistema de Apoio Gráfico, descreve-se a seguir um programa exemplo que, por sua simplicidade e clareza, torna transparente boa parte da teoria exposta ao longo deste trabalho.

De acordo com a elaboração deste programa, far-se-á necessária uma abordagem dos seguintes itens nesta ordem:

. Inicializando o "zmSAG";

. Criação de primitivas e segmentos;

. manipulação de segmentos;

Este programa desenhará, em qualquer estação de trabalho desejada, um alicate de pressão; que compõe-se de um "mecanismo de quatro barras não Grashof", com a flexibilidade de permitir uma visualização, em separado, de diversas posições do mesmo.

8.2 Inicilizando o "zmSAG"

Para colocar o sistema em operação, antes de tudo, deve-se fazer uma chamada às seguintes funções:

```
zm_open_SAG();

zm_activate_ws();

zm_set_window();
```

A primeira função, zm_openSAG(), é responsável pela abertura do "zmSAG", colocando-o no estado de operação "GKOP"; significando que o sistema encontra-se pronto para operar. Caso seja feita uma tentativa de chamar qualquer outra função do sistema antes da zm_openSAG(), o programa abortará. Para esta função,

como argumento deve ser passado o nome de um arquivo que conterá as mensagens de erro devolvidas pelo sistema. Neste caso, o nome escolhido foi "erros.txt" (as aspas são obrigatórias). Este arquivo deverá ser consultado, pelo usuário, sempre que houver a impossibilidade de executar o programa.

Com relação à função zm_set_window() não existe a obrigatoriedade de chamá-la, desde que existe uma opção default no sistema. Entretanto é recomendável que o usuário defina sua própria janela, pois só assim haverá uma garantia de que o desenho será apresentado nas proporções desejadas. Do contrário a janela ficará padronizada no tamanho A0 (841 x 1189 milímetros) e o desenho provavelmente ficará com as dimensões reduzidas quando apresentado em qualquer estação de trabalho, caso não guarde as devidas proporções com este formato. Os argumentos desta função são: um identificador da janela e as dimensões da mesma.

A função zm_activate_ws() é usada para ativar as diferentes estações de trabalho e, como a primeira função citada, é também obrigatória. Sendo assim, qualquer tentativa de criar-se primitivas ou segmentos sem que pelo menos uma das estações esteja ativa, acarretará num erro fatal. É passado para o sistema, através desta, a estação de trabalho desejada e a opção à ela associada. Aqui a estação escolhida foi o vídeo, através da opção "DEFAULT".

No exemplo citado anteriormente isto foi feito da seguinte forma,

```
zm_openSAG ( "erros.txt" );
zm_set_window ( 1, -20, -20, 280, 200 );
zm_activate_ws ( VIDEO, DEFAULT );
```

8.3 Criando Segmentos

Após a inicialização do sistema, neste exemplo, foram usadas as seguintes primitivas gráficas:

- Rectangle
- circle
- Polyline
- Disjointpline
- Circular_arc

No código do programa o uso de primitivas dá-se através de chamadas à funções, passando-se os argumentos requeridos. Toda chamada a uma função, com o intuito de construir uma primitiva, pode acontecer ou dentro de um segmento ou fora deste. No exemplo somente foram requeridas primitivas dentro de segmentos; como pode-se verificar num trecho do programa onde é desenhado uma das partes do alicate:

```
zm_create_segment (1);

    zm_circular_arc ( 55.5, 47, 178.3, 282, 37.2);
    zm_circular_arc ( 79.5, 24.9, 64, 159, 32);
    zm_circular_arc ( 123.95, 6.95, 88.7, 97.1, 15);
    zm_circular_arc ( 122.6, 61.7, 246.2, 266.9, 20.4);

    zm_polyline ( Npt_plp, plLx, plLy );
    zm_disjpolyline ( Npt_pld, plx, ply );
    zm_rectangle ( 250, 24, 262, 39.5 );

zm_close_segment ();
```

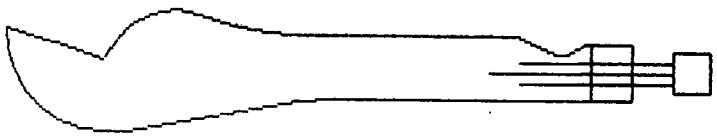


FIGURA 8.1 - Componente do alicate gerado pelo trecho de programa que cria o segmento "1".

A diferença fundamental entre as duas maneiras de desenhar-se uma primitiva reside no fato de que uma primitiva criada dentro de um segmento fica associada ao mesmo, voltando a ser redesenhada toda vez que o segmento for referenciado, enquanto que se esta for criada pela chamada de uma função posicionada fora do segmento só terá um efeito imediato, perdendo-se após o seu uso.

Um segmento na sua forma mais simples é formado apenas por primitivas e atributos, sem referenciar outros segmentos já definidos. Para criá-lo, além das funções que definem as primitivas, se faz também necessário o uso de duas novas funções: zm_create_segment() e zm_close_segment().

A função zm_create_segment() deve ser chamada toda vez em que se desejar definir um novo segmento e, a partir dela, até que seja encontrada a função zm_close_segment(), todas as primitivas relacionadas através de chamadas de funções passarão a fazer parte deste. Esta função, zm_create_segment(), exige a passagem de um argumento (do tipo inteiro) que corresponderá ao número do segmento para futuras referências.

A função zm_close_segment() fecha o segmento. Como só é criado um segmento por vez, esta função deve ser sempre usada para fechar o segmento quando o usuário terminar de definí-lo.

Estas duas últimas funções podem ser observadas no trecho de programa citado anteriormente.

8.4 Manipulando Segmentos

Após a criação do segmento, este poderá ser utilizado de diversas maneiras. No programa exemplo apenas foram operadas as transformações de rotação e translação.

As funções que realizam estas transformações nos segmentos são as seguintes:

- zm_eval_transf_matx();
- zm_SetSeg_transform();
- zm_copy_segment_to_ws();

A primeira delas possui sua síntax como a seguir:

void zm_eval_transf_matx (x_0, y_0, d_x, d_y, ang, f_x, f_y, mat)

O significado de cada argumento pode ser visto no item 6.4 do capítulo 6.

Sendo assim, esta função é indispensável à manipulação de segmentos, pois sem ela, não seria possível a construção da matriz de transformação. Entretanto cabe à função zm_SetSeg_transform() associar a matriz de transformação calculada em zm_eval_tranf_matx() ao referido segmento.

Para tornar mais claro, a seguir é apresentado (figura 8.2) um trecho do exemplo que translada e rotaciona a garra do alicate:

```
zm_eval_transf_matx ( 13, 13.9, 68.2, 19.3, 20, 1, 1, mat);
zm_SetSeg_transform ( 3, mat );
zm_copy_segment_to_ws ( 3, VIDEO );
```

FIGURA 8.2 - Rotação e translação de segmentos no caso do alicate.

A função zm_eval_transf_matx() constrói a matriz responsável pela transformação de translação de 68.2 unidades na direção "x"; 19.3 unidades na direção "y"; além de rotacionar a garra de 20 graus em torno do ponto (13, 13.9). Por fim, coloca o ponteiro "mat" apontando para esta matriz.

Na chamada à função zm_SetSeg_tranform() são passados dois argumentos. Um deles é o segmento que será transformado, e o outro é o ponteiro para a matriz construída pela função zm_eval_tranf_matx().

Finalmente a operação é concluída com uma chamada à função zm_copy_segment_to_ws(). Esta, envia o segmento especificado para ser desenhado na estação de trabalho também especificada. Como argumentos desta função tem-se a estação de trabalho, no caso "VIDEO", e o segmento escolhido.

A página seguinte (figuras 8.3 e 8.4) são apresentadas várias posições do alicate, que se compõe de seis segmentos em sua forma completa. O programa utilizado tem a simplicidade do exemplo aqui exposto, apenas se utiliza de vários segmentos e transformações. A utilizaçao do programa se fez em cima de uma CPU do tipo PC-XT, com placa adaptadora de gráficos colorida do tipo CGA, com velocidade de processamento muito baixa se comparada aos sistemas dedicados ao

processamento gráfico de imagens, ainda assim, devido a pouca quantidade de dados, a performance foi excelente, sendo as imagens geradas quase que instantaneamente no vídeo do computador. Como dispositivo de saída externa, foi utilizada uma impressora matricial, compatível com o padrão EPSON, de nove agulhas, sendo portanto muito pobre a capacidade de resoluçao da imagem. Melhor resolução poderia ter sido obtida com o uso de um traçador gráfico.

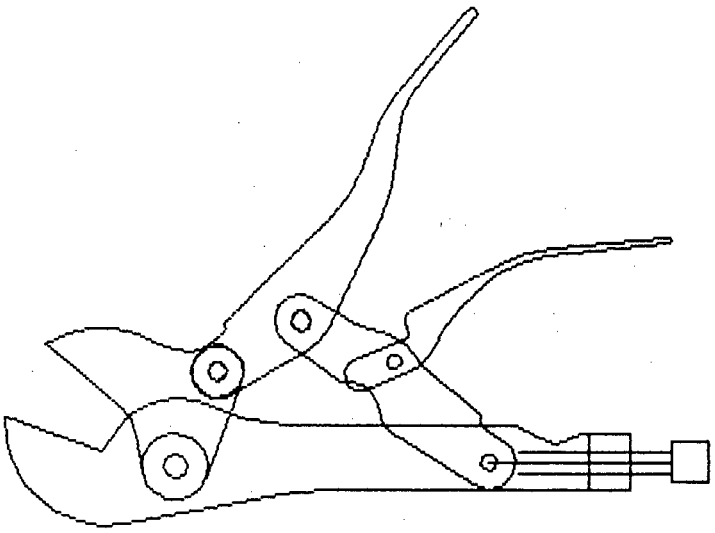


FIGURA 8.3 - Alicate de pressão composto por vários segmentos.

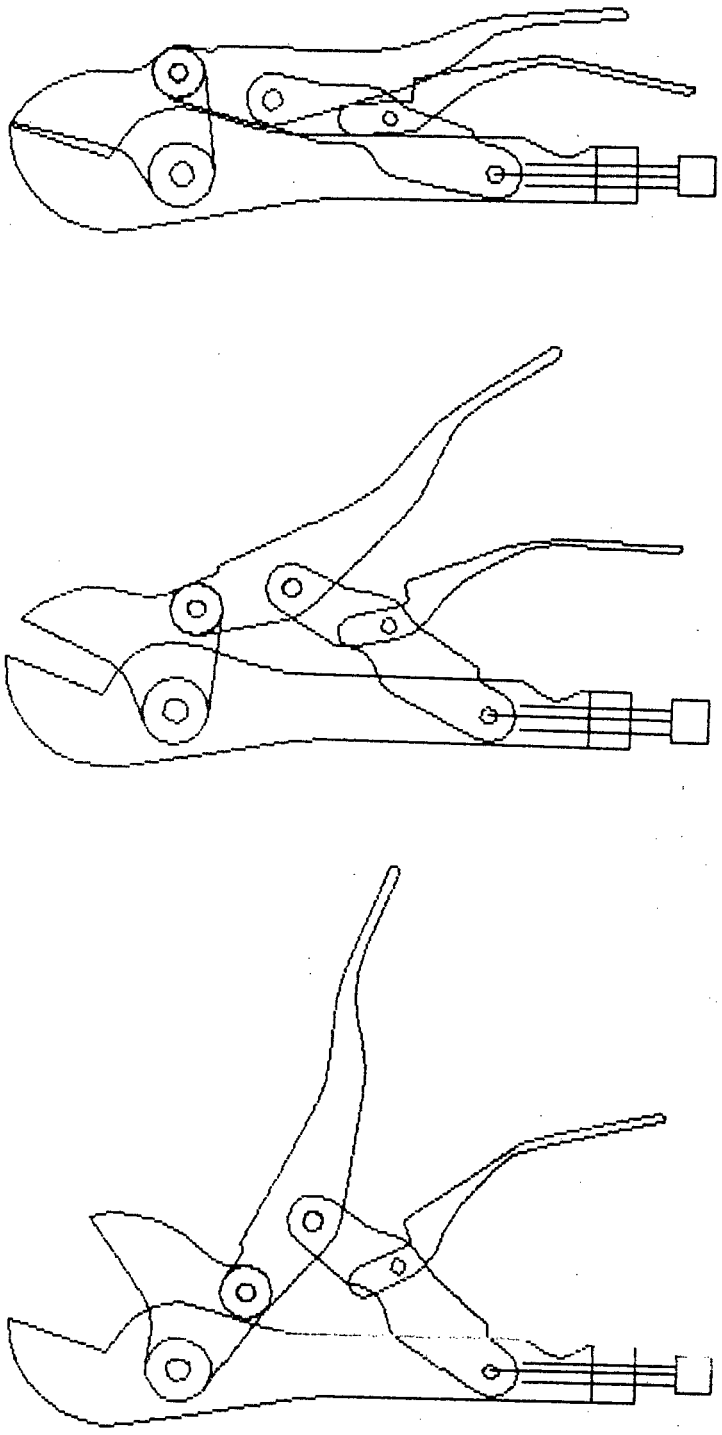


FIGURA 8.4 - Várias posições do mecanismo "alicate de pressão" geradas no dispositivo de impressão.

CAPÍTULO 9
Conclusão

Por se constituir num "pacote gráfico" seguindo, apenas em parte, a norma GKS, o "zmSAG" não vai encontrar similar, no que diz respeito a sintaxe de suas funções. Isto irá prejudicar um pouco a portabilidade a outros sistemas ou de outros sistemas ao "zmSAG". Porém, este fato não se dá meramente por acaso, e sim pela simplicidade que traz ao programador algumas primitivas não padronizadas, tais como *circle, arc, ellipse, elliptical_arc, etc.*, que, dentro do GKS, teriam de ser implementadas através de "GDP's". Um outro fator que influenciou bastante alguns "escapes" da norma GKS, foi o fato de não se ter ainda um "binding" da mesma direcionado à linguagem "C", tendo se trabalhado todo o tempo com o "binding" do FORTRAN.

Apesar de ser interessante para efeitos de melhoramentos, somente uma futura implementação do "zmSAG" em supermicros, poderá permitir que se compare o mesmo com outros "softwares" do gênero, uma vez que os "pacotes gráficos" de cunho profissional, baseados no GKS, somente são implementados em máquinas deste porte ou superior, tendo em vista a necessidade de grande quantidade de memória volátil exigida por estes. Para o mundo dos PC's, existem no mercado alguns compiladores de linguagens que trazem uma pequena interface gráfica, porém estas são muito simplificadas para efeito de comparação e, provavelmente levariam a conclusões de pouca confiabilidade.

É ainda importante se mencionar o fato de que os "pacotes gráficos" mais elaborados, requerem ou requereram anos de trabalho de uma equipe de pessoas especializadas nos vários tópicos exigidos na confecção de um núcleo gráfico. Sendo assim, não é de se esperar que o "zmSAG", no seu estado atual vá superar quaisquer outros sistemas, de mesmas características, existentes no mercado.

Em vista do acima exposto, pode-se concluir que o "zmSAG" se constitui numa excelente experiência acadêmica e num marco importante para um desenvolvimento mais elaborado, em nível de Universidade e por uma equipe de trabalho, em cima do seu código fonte.

Finalmente, este sistema tem como finalidade básica auxiliar no desenvolvimento de aplicativos em CAD, CADD, CNC, etc., desta forma não se apresenta como uma ferramenta final e sim como um elo entre os programas aplicativos e o sistema compugráfico. Vendo por este prisma, será necessário o uso intensificado do mesmo por programadores, que utilizem processamento gráfico, para que se possa chegar a uma conclusão definitiva sobre o trabalho. Com relação a isto, e levando-se em consideração o fato de o mesmo ser um trabalho de cunho acadêmico e não profissional, sempre estará aberto o espaço para sugestões e críticas dos colegas da área, bem como para aqueles que se interessem em aprimorar o mesmo, quer seja na construção de drivers ou na otimização de subrotinas, uma vez que o mesmo tem uma arquitetura aberta e simplificada para este fim.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] - FOLEY, James D.,"Fundamentals of Interactive Computer Graphics", Addison Wesley Publishing Company, 1982.

[2] - ROGERS, D. F. and Adams, J. A.,"Mathematical Elements for Computer Graphics", McGraw-Hill, New York, 1976.

[3] - HARRINGTON, Steven, "Computer Graphics - A programming Approach", McGraw-Hill, 1983.

[4] - NEWMAN, W. M. and Sproull, R. F., "Principles of Interactive Computer Graphics", McGraw-Hill, 1979.

[5] - PERSIANO, Ronaldo C. M. e Oliveira, A. A. F., "Introdução à Computação Gráfica", Belo Horizonte, UFMG, 1986.

[6] - VENETIANER, Tomas, "Desmistificando a Computação Gráfica", McGraw-Hill, São Paulo, 1988.

[7] - MASSOLA, A. M. A., Tori, R., Filgueiras, L. e Arakaki, R., "Fundamentos de Computação Gráfica", LTC Editora S.A., São Paulo, 1987.

[8] - LÉON-LASTRA, Gerardo, "Text and 3D viewing in a Core-like Graphics Package", in CAD/CAM as a Basis - Ed. by J. L. Encarnação.

[9] - ENNS, Steve, "Implement Bézier and B-spline curves for interactive graphics", revista Byte, pg 225 a 230, dezembro, 1986.

[10] - RIESENFELD, R. F., "Homogeneous Coordinates and Projective Planes in Computer Graphics", IEEE Comput. Graphics and Applications Vol 1 No. 1 (1980) pp. 50-55.

[11] - HEBEZ, "Homogeneous Coordinates for Computer Graphics", Computer-aided design, Vol 15, No. 6 (1983) pp. 361-365.

[12] - BRESENHAM, J. E., "A Linear Algorithm for Incremental Digital Display of Circular Arcs", Communications of the ACM, 20(2), February 1977, pp. 100-106.

[13] - BRESENHAM, J. E., "Algorithm for Computer Control of Digital Plotter", IBM Syst. J., 4(1) 1965, pp. 25-30.

[14] - CLARKSON, T. B., "Standardized Graphics on the IBM Personal Computer", IBM Syst. J., 24(1) 1985, pp. 3-13.

[15] - MAGALHAES, L. P., "Computação Gráfica", Ed. Papirus, Campinas, 1986.

[16] - TOZZI, C. L., "PAC - Projeto Auxiliado por Computador", Campinas, 1986.

[17] - CUNHA, G. J., "Computação Gráfica e suas Aplicações em CAD", Ed. Atlas, São Paulo, 1987.

[18] - CUNHA, G. J., "Computação Gráfica, o Padrão GKS", Ed. Atlas, São Paulo, 1987.

[19] - SANTO, H. P., "Métodos Gráficos e Geometria Computacionais", Dinalivro, Lisboa.

[20] - PFAFF, G., "Computer Graphics Programming GKS - The Graphics Standard", Springer-Verlag, 1987.

[21] - SOBELMAN, G. E. and Krekelberg, D. E., "Advanced C: Techniques and Applications", Que Corporation, 1985.

[22] - CARLBOM, Ingrid and Paciorek, Joseph, "Planar Geometric Projections and Viewing Transformations", ACM - Computing Surveys, Vol. 10, No. 4, December 1978.

[23] - MIRSHAWKA, Vítor, "Imprimindo Maravilhas com a Grafix", Nobel, São Paulo, 1985.

[24] - BROW, M. Pardo, "Técnicas de Pesquisa em Tabelas", Edgard Blücher, São Paulo, 1979.

APÊNDICE A
Funções do GKS

A. Lista de Funções

A.1 Funções de Controle

OPEN GKS
CLOSE GKS
OPEN WORKSTATION
CLOSE WORKSTATION
ACTIVATE WORKSTATION
DEACTIVATE WORKSTATION
CLEAR WORKSTATION
REDRAW ALL SEGMENTS ON WORKSTATION
UPDDATE WORKSTATION
SET DEFERRAL STATE
MESSAGE
ESCAPE

A.2 Funções de Saída

POLYLINE
POLYMARKER
TEXT
FILL AREA
CELL ARRAY
GENERALIZED DRAWING PRIMITIVE (GDP)

A.3 Funções para Definição de Atributos Independentes de ET

SET POLYLINE INDEX
SET LINETYPE
SET LINEWIDTH SCALE FACTOR
SET POLYLINE COLOUR INDEX
SET POLYMARKER INDEX
SET MARKER TYPE
SET MARKER SIZE SCALE FACTOR

SET POLYMARKER COLOUR INDEX
SET TEXT INDEX
SET TEXT FONT AND PRECISION
SET CHARACTER EXPANSION FACTOR
SET CHARACTER SPACING
SET TEXT COLOUR INDEX
SET CHARACTER HEIGHT
SET CHARACTER UP VECTOR
SET TEXT PATH
SET TEXT ALIGNMENT
SET FILL AREA INDEX
SET FILL AREA INTERIOR STYLE
SET FILL AREA STYLE INDEX
SET FILL AREA COLOUR INDEX
SET PATTERN SIZE
SET PATTERN REFERENCE POINT
SET ASPECT SOURCE FLAGS
SET PICK IDENTIFIER

A.4 Funções para Definição de Atributos Dependentes de ET

SET POLYLINE REPRESENTATION
SET POLYMARKER REPRESENTATION
SET TEXT REPRESENTATION
SET FILL AREA REPRESENTATION
SET PATTERN REPRESENTATION
SET COLOUR REPRESENTATION

A.5 Funções de Transformação de Normalização

SET WINDOW
SET VIEWPORT
SET VIEWPORT INPUT PRIORITY
SELECT NORMALIZATION TRANSFORMATION
SET CLIPPING INDICATOR

A.6 Funções de Transformação de ET

SET WORKSTATION WINDOW
SET WORKSTATION VIEWPORT

A.7 Funções de Segmento

CREATE SEGMENT
CLOSE SEGMENT
RENAME SEGMENT
DELETE SEGMENT
DELETE SEGMENT FROM WORKSTATION
ASSOCIATE SEGMENT WITH WORKSTATION
COPY SEGMENT TO WORKSTATION
INSERT SEGMENT

SET SEGMENT TRANSFORMATION
SET VISIBILITY
SET HIGHLIGHTING
SET SEGMENT PRIORITY
SET DETECTABILITY

A.7 Funções de Entrada

INITIALISE LOCATOR
INITIALISE STROKE
INITIALISE VALUATOR
INITIALISE CHOISE
INITIALISE PICK
INITIALISE STRING

SET LOCATOR MODE
SET STROKE MODE
SET VALUATOR MODE
SET CHOICE MODE
SET PICK MODE
SET STRING MODE

REQUEST LOCATOR
REQUEST STROKE
REQUEST VALUATOR
REQUEST CHOICE
REQUEST PICK
REQUEST STRING

SAMPLE LOCATOR
SAMPLE STROKE

SAMPLE VALUATOR
SAMPLE CHOICE
SAMPLE PICK
SANMPLE STRING

AWAIT EVENT
FLUSH DEVICE EVENTS
GET LOCATOR
GET STROKE
GET VALUATOR
GET CHOICE
GET PICK
GET STRING

A.8 Funções de Metafile

WRITE ITEM TO GKSM
GET ITEM TYPE FROM GKSM
READ ITEM FRM GKSM
INTERPRET ITEM

A.9 Funções de Pergunta

INQUIRE OPERATING STATE VALUE
INQUIRE LEVEL OF GKS
INQUIRE LIST OF AVAILABLE WORKSTATION TYPES
INQUIRE WORKSTATION MAXIMUM NUMBERS
INQUIRE MAXIMUM NORMALIZATION TRANSFORMATION NUMBER
INQUIRE SET OF OPEN WORKSTATIONS
INQUIRE SET OF ACTIVE WORKSTATIONS
INQUIRE CURRENT PRIMITIVE ATTRIBUTE VALUES
INQUIRE CURRENT INDIVIDUAL ATTRIBUTE VALUES
INQUIRE CURRENT NORMALIZATION TRANSFORMATION NUMBER
INQUIRE LIST OF NORMALIZATION TRANSFORMATION NUMBERS
INQUIRE NORMALIZATION TRANSFORMATION
INQUIRE CLIPPING INDICATOR
INQUIRE NAME OF OPEN SEGMENT
INQUIRE SET OF SEGMENT NAMES IN USE
INQUIRE MORE SIMULTANEOUS EVENTS
INQUIRE WORKSTATION CONNECTION AND TYPE

INQUIRE WORKSTATION STATE
INQUIRE WORKSTATION DEFERRAL AND UPDATE STATES
INQUIRE LIST OF POLYLINE INDICES
INQUIRE POLYLINE REPRESENTATION
INQUIRE LIST OF POLYMARKER INDICES
INQUIRE POLYMARKER REPRESENTATION
INQUIRE LIST OF TEXT INDICES
INQUIRE TEXT REPRESENTATION
INQUIRE TEXT EXTENT
INQUIRE LIST OF FILL AREA INDICES
INQUIRE FILL AREA REPRESENTATION
INQUIRE LIST OF PATTERN INDICES
INQUIRE PATTERN REPRESENTATION
INQUIRE LIST OF COLOUR INDICES
INQUIRE COLOUR REPRESENTATION
INQUIRE WORKSTATION TRANSFORMATION
INQUIRE SET OF SEGMENT NAMES ON WORKSTATION
INQUIRE LOCATOR DEVICE STATE
INQUIRE STROKE DEVICE STATE
INQUIRE VALUATOR DEVICE STATE
INQUIRE CHOICE DEVICE STATE
INQUIRE PICK DEVICE STATE
INQUIRE STRING DEVICE STATE
INQUIRE WORKSTATION CATEGORY
INQUIRE WORKSTATION CLASSIFICATION
INQUIRE MAXIMUM DISPLAY SURFACE SIZE
INQUIRE DINAMIC MODIFICATION OF WORKSTATION ATTRIBUTES
INQUIRE DEFAULT DEFERRAL STATE VALUES
INQUIRE POLYLINE FACILITIES
INQUIRE PREDEFINED POLYLINE REP{RESENTATION
INQUIRE POLYMARKER FACILITIES
INQUIRE PREDEFINED POLYMARKER REPRESENTATION
INQUIRE TEXT FACILITIES
INQUIRE PREDEFINED TEXT REPRESENTATION
INQUIRE FILL AREA FACILITIES
INQUIRE PREDEFINED FILL AREA REPRESENTATION
INQUIRE PATTERN FACILITIES
INQUIRE PREDEFINED PATTERN REPRESENTATION
INQUIRE COLOUR FACILITIES
INQUIRE PREDEFINED COLOUR REPRESENTATION

INQUIRE LIST OF AVAILABLE GENERALIZED DRAWING PRIMITIVES
INQUIRE GENERALIZED DRAWING PRIMITIVE
INQUIRE MAXIMUM LENGTH OF WORKSTATION STATE TABLES
INQUIRE NUMBER OF SEGMENT PRIORITIES SUPPORTED
INQUIRE DYNAMIC MODIFICATION OF SEGMENT ATTRIBUTES
INQUIRE NUMBER OF AVAILABLE LOGICAL INPUT DEVICES
INQUIRE DEFAULT LOCATOR DEVICE DATA
INQUIRE DEFAULT STROKE DEVICE DATA
INQUIRE DEFAULT VALUATOR DEVICE DATA
INQUIRE DEFAULT CHOICE DEVICE DATA
INQUIRE DEFAULT PICK DEVICE DATA
INQUIRE DEFALUT STRING DEVICE DATA
INQUIRE SET OF ASSOCIATED WORKSTATIONS
INQUIRE SEGMENT ATTRIBUTES
INQUIRE PIXEL ARRAY DIMENSIONS
INQUIRE PIXEL ARRAY
INQUIRE PIXEL
INQUIRE INPUT QUEVE OVERFLOW

A.10 Funções Utilitárias

EVALUATE TRANSFORMATION MATRIX
ACCUMULATE TRANSFORMATION MATRIX

A.11 Funções de Tratamento de Erro

EMERGENCY CLOSE GKS
ERROR HADLING
ERROR LOGGING

APÊNDICE B

Descrição Sumária do Núcleo "zmSAG"

B.1 Introdução

O "zmSAG" (Biblioteca de Apoio Gráfico) é composto de rotinas simples que executam todas as funções gráficas fundamentais de forma independente do dispositivo. Esta independência é obtida pela definição de um dispositivo gráfico virtual (coordenadas normalizadas) e dispositivos lógicos de entrada. Para acionar equipamentos reais, cujos comandos não são normalizados, são utilizadas interfaces adequadas que convertem as instruções para comandos e coordenadas próprios dos dispositivos.

Suas rotinas, dentro dos limites impostos pelo hardware (PC-compatível) à Computação Gráfica, são parte de um subconjunto da norma GKS no nível 1a. Os usuários que se iniciarem com o zmSAG, não terão dificuldade na utilização de outros "pacotes gráficos"mais elaborados que sigam o GKS ou outra normalização qualquer.

B.2 Dispositivo Virtual

Para o programador do aplicativo gráfico, haverá apenas um único tipo de dispositivo gráfico de saída que terá uma área quadrangular de 1 x 1. Os comandos enviados ao dispositivo virtual passarão por uma interface específica para cada dispositivo lógico real, que fará a conversão. Este procedimento é transparente ao usuário programador.

B.3 Sistema de Coordenadas

O "zmSAG" permite que o usuário trabalhe normalmente nas suas unidades sem que haja uma preocupação com as unidades do dispositivo virtual, a não ser para definição de viewports. As funções gráficas são executadas com os parâmetros fornecidos em coordenadas cartesianas bidimensionais. Estas coordenadas, denominadas globais (WC), não se referem a nenhuma unidade de medida específica, sendo portanto, adimensionais.

Os desenhos gerados através do zmSAG, são transformados nas coordenadas dos dispositivos automaticamente em duas etapas: transformação de

normalização e transformação de estação de trabalho. Para isto são definidos dentro do "zmSAG", três espaços de coordenadas:

. coordenadas globais - coordenadas utilizadas no programa de aplicação para descrever os objetos gráficos do usuário;

. coordenadas normalizadas - coordenadas independentes do dispositivo e com valores no intervalo [0,1] x [0,1];

. coordenadas do dispositivo - coordenadas específicas para cada dispositivo gráfico do sistema.

B.3.1 "Window" e "Viewport"

Define-se "window" como sendo os limites de uma área retangular no espaço de coordenadas de partida e "viewport" como os limites de outra área retangular no espaço de coordenadas destino. (preservou-se o uso das expressões em inglês para não ficar em discordância com o GKS).

Com isto poderá haver duas transformações envolvendo os sistemas de coordenadas descritos no parágrafo anterior:

. transformação de normalização (ou de "window/viewport") transforma coordenadas globais (WC) em coordenadas normalizadas (NDC), através de uma "window" definida em coordenadas globais para uma "viewport" em coordenadas normalizadas;

. transformação de estação de trabalho - faz o mapeamento das coordenadas normalizadas (NDC) para as coordenadas do periférico (DC).

No "zmSAG", a transformação de estação de trabalho é feita automaticamente quando a estação está aberta.

B.4 Entrada Gráfica

De forma a organizar as várias possibilidades de aquisição de dados gráficos num sistema uniforme e não criar dependências em relação ao hardware, o "zmSAG" trata os dispositivos lógicos de entrada como meios virtuais de interação com o sistema que, fisicamente podem ser implementados por teclados, mouse, mesas digitalizadoras, etc...

Para obtenção de dados gráficos, o "zmSAG" prevê os seguintes "dispositivos virtuais":

. Locator - fornece a posição de um ponto em coordenadas do usuário;

. Stroke - fornece uma seqüência de pontos em coordenadas do usuário;

. Valuator - fornece um valor real;

. Choice - fornece um número inteiro;

. String - fornece uma seqüência de caracteres alfanuméricos.

B.5 Primitivas Gráficas

As primitivas usados no "zmSAG" são:

. Polyline - geração de um conjunto de linhas conectando uma seqüência de pontos fornecida;

. Polygon - idêntico a "polyline", exceto que o último ponto é ligado ao primeiro, e a área é preenchida com a cor ou hachúria em andamento;

. Disjointpline - geração de uma seqüência de linhas conectadas de dois em dois pontos, de um array fornecido;

. Polymarker - geração de símbolos centrados numa sequência de pontos fornecida;

. Circle - desenha um círculo;

. Circular_arc - desenha um arco de círculo;

. Elipse - desenha uma elipse;

. Elliptical_arc - desenha um arco de elipse;

. Rectangle - desenha um retângulo preenchendo-o com a cor ou hachúria em andamento;

. Text - geração de uma sequência de caracteres numa posição fornecida;

. GPD (*Generalized Drawing Primitive*) - permite o traçado de primitivas não padronizadas no GKS (arcos, círculos, elipses, etc).

B.6 Segmentação

As primitivas podem ser agrupadas em conjuntos e identificadas por um nome. Esses conjuntos são chamados segmentos. Os segmentos podem ser encarados como elementos do mais alto nível, com os quais as imagens podem ser construídas e editadas mais facilmente.

No "zmSAG", as primitivas que não pertencem a algum segmento não são passíveis de acesso pelo programa de aplicação depois de geradas.

APÊNDICE C
Rotinas Básicas do "zmSAG"

C. Principais Rotinas do "zmSAG"

A seguir são relacionadas as principais funções que podem ser chamadas pelo usuário. Nesta primeira versão, estas funções podem ser chamadas apenas de um programa escrito em linguagem "C", em versões futuras está prevista a implementação em Pascal e FORTRAN.

C.1 Funções de Controle

zm_activate_wt (est)

parâmetros:

est - identificador da estação

função:

Ativar a estação de trabalho cujo nome é especificado. Todas as rotinas passarão a ter efeito sobre esta estação de trabalho, continuando a afetar as que já se encontram ativas.

zm_deactive_wt (est)

parâmetros:

est - identificador da estação

função:

Desativar a estação de trabalho cujo nome é especificado. As rotinas deixam de ter efeito sobre esta estação.

zm_clear_wt (est)

parâmetros:

est - identificador da estação

função:

Limpar a estação de trabalho

C.2 Funções de saída

Elementos de linha

zm_polyline (n, vetx, vety)

parâmetros:

n - número de vértices

vetx - ponteiro para um vetor contendo as coordenadas x, no sistema do usuário, dos pontos que definem a polilinha.

vety - ponteiro para um vetor contendo as coordenadas y, no sistema do usuário, dos pontos que definem a polilinha.

função:

Desenhar uma sequência de segmentos consecutivos, começando no primeiro ponto do vetor fornecido e terminando no último. Os segmentos serão desenhados em todas as superfícies de apresentação das estações ativas no momento.

zm_circular_arc (x0, y0, start, end, rad)

parâmetros:

x0 - coordenada x do centro do arco

y0 - coordenada y do centro do arco

start - ângulo onde se inicia o arco

end - ângulo onde termina o arco

rad - raio do arco

função:

Desenhar um segmento circular nas estações ativas

zm_elliptical_arc (x0, y0, start, end, radx, rady)

parâmetros:

x0 - coordenada x do centro do arco de elipse

y0 - coordenada y do centro do arco de elipse

start - ângulo onde se inicia o arco de elipse

end - ângulo onde termina o arco de elipse

radx - raio segundo o eixo x do arco de elipse

rady - raio segundo o eixo y do arco de elipse

função:

Traçar um arco de elipse nas estções ativas

zm_GDP ()

Primitiva não padronizada a ser implementada em versões futuras para melhorar a portabilidade.

C.3 Elemento de Marca

zm_polymarker (n, vetx, vety)

parâmetros:

n - número de marcas

vetx - ponteiro para um vetor contendo as coordenadas x, no sistema do usuário, dos centros dos marcadores

vety - ponteiro para um vetor contendo as coordenadas y, no sistema do usuário, dos centros dos marcadores

função:

Desenhar uma sequência de marcas nas posições fornecidas.

C.4 Elementos de Area

zm_polygon (n, vetx, vety)

parâmetros:

n- número de vértices do polígono

vetx - ponteiro para um vetor contendo as coordenadas x, no sistema do usuário, dos vértices do polígono.

vety - ponteiro para um vetor contendo as coordenadas y, no sistema do usuário, dos vértices do polígono.

função:

Desenhar um polígono preenchendo-o com uma cor ou textura previamente especificada.

zm_rectangle (minx, miny, maxx, maxy)

parâmetros:

minx,miny - coordenadas do canto inferior esquerdo do retângulo

maxx,maxy - coordenadas do canto superior direito do retângulo

função:

Desenhar um retângulo, preenchendo-o com uma cor ou textura previamente especificada.

zm_circle (x0, y0, rad)

parâmetros:

x0,y0 - coordenadas do centro do círculo

rad - raio do círculo

função:

Desenhar um círculo, preenchendo-o com uma cor ou textura previamente especificada.

zm_elipse (x0, y0, radx, rady)

parâmetros:

x0 - coordenada x do centro da elipse

y0 - coordenada y do centro da elipse

radx - raio segundo o eixo x da elipse

rady - raio segundo o eixo y da elipse

função:

Desenhar uma elipse, preenchendo-a com uma cor ou textura previamente especificada.

C.5 Atributos de saída

zm_set_linetype(tipo)

parâmetros:

tipo- especifica o tipo de linha

cont - linha contínua ————————

trac - linha tracejada — — — — — —

pont - linha pontilhada • • • • • • • • • •

tcpt - linha traço-ponto —·—·—·—·—

função:

Definir o tipo das linhas que serão traçadas.

zm_set_line_thickness (esp)

parâmetros:

esp- espessura da linha

norm - espessura normal

med - espessura média

gros - linha grossa

função:

Definir a espessura das linhas que serão traçadas.

zm_set_line_color (cor)

parâmetros:

cor - modifica atributo de cor

função:

Especificar a cor das linhas do desenho a ser usada.

zm_set_markertype (tipo)

parâmetros:

tipo - tipo de marcador

cruz - +
xis - ×
quad - □
circ - ○
triang - △
ponto - •

função:

Modificar o tipo de marcador utilizado pela função "polymarker".

zm_set_markercolor (cor)

parâmetros:

cor - especifica cor de marcador

função:

Modificar a cor com que serão desenhados os marcadores.

zm_set_text_font (fonte)

parâmetros:

fonte - fonte de caracteres a ser usada.

função:

Especifica novo estilo de caracteres a ser utilizado.

zm_set_character_height (altura)

parâmetros:

altura - altura do caractere

função:

Definir nova altura para o caractere.

zm_set_text_path (ang)

parâmetros:

ang - ângulo, em graus, e no sentido horário a partir da vertical

função:

Definir a direção do texto.

zm_set_text_alignment (horiz, vert)

parâmetros:

horiz - alinhamento horizontal

LEFT
CENTRE
RIGHT

vert - alinhamento vertical

TOP
HALF
BOTTOM

função:

Definir o alinhamento do texto.

zm_set_textcolor (cor)

parâmetros:

cor - cor do caractere

função:

Definir nova cor de texto.

zm_set_fill_style (estilo)

parâmetros:

estilo - estilo interior da FILL AREA

SOLID

PATTERN

HATCH

função:

Definir o estilo interior para as funções que manipulam áreas.

zm_set_fill_color (cor)

parâmetros:

cor - índice de cor do FILL AREA

função:

Definir uma cor para uso das funções que preenchem áreas.

zm_set_pattern (ptipo)

parâmetros:

ptipo - tipo de pattern

função:

Selecionar um novo tipo de pattern.

C.6 Funções de Transformação

zm_set_window (xmin, ymin, xmax, ymax)

parâmetros:

xmin,ymin - coordenadas x e y respectivamente, no sistema do usuário, do canto inferior esquerdo da janela.

xmax,ymax - coordenadas x e y respectivamente, no sistema do usuário, do canto superior direito da janela.

função:

Definir um retângulo, no espaço de trabalho do usuário, que deverá ser mapeado para a "viewport" do dispositivo virtual.

zm_set_viewport (xmin, ymin, xmax, ymax)

parâmetros:

xmin,ymin - coordenadas x e y respectivamente, no sistema normalizado, do canto inferior esquerdo da janela.

xmax,ymax - coordenadas x e y respectivamente, no sistema normalizado, do canto superior direito da janela.

função:

Definir um retângulo, dentro da superfície de apresentação do dispositivo virtual, onde deverá ser mapeado o retângulo da janela do usuário.

zm_set_wt_window (est, xmin, ymin, xmax, ymax)

parâmetros:

est - identificador da estação

xmin,ymin - coordenadas x e y respectivamente, no sistema normalizado, do canto inferior esquerdo da janela.

xmax,ymax - coordenadas x e y respectivamente, no sistema normalizado, do canto superior direito da janela.

função:

Definir um retângulo, dentro da superfície de apresentação do dispositivo virtual, que deverá ser mapeado para a superfície total do dispositivo físico de saída.

zm_set_clipping (ind)

parâmetros:

ind - indicador de recorte

clipON

clipOFF

função:

Definir se as primitivas gráficas de saída devem ser recortadas ou não dentro da janela definida pelo usuário.

C.7 Funções de Segmentação

zm_create_segment (name)

parâmetros:

name - nome do segmento a ser criado

função:

Abrir um segmento de nome especificado, que será constituído por todas as primitivas enviadas até que o segmento seja fechado.

zm_close_segment ()

parâmetros:

sem parâmetros

função:

Encerrar a definição de um segmento que se encontrava em criação.

zm_delete_segment (name)

parâmetros:

name - nome do segmento a ser eliminado

função:

Remover, do sistema (WISS), o segmento de nome especificado.

zm_rename_segment (oldname, newname)

parâmetros:

oldname - nome atual do segmento

newname - novo nome a ser atribuído ao segmento

função:

Trocar o nome de um segmento já existente no sistema.

zm_insert_segment (name, matran)

parâmetros:

name - nome do segmento a ser inserido

matran - ponteiro para a matriz de transformação

função:

Inserir um segmento, que já faz parte do sistema, em outro segmento que se encontra em criação.

zm_copy_seg_to_workstation (name)

parâmetros:

name - nome do segmento a ser copiado

função:

Enviar para a superfície de apresentação do dispositivo virtual, as primitivas que compõe o segmento especificado.

zm_set_segment_visibilit (name, vis)

parâmetros:

name - nome do segmento

vis - visibilidade

VISIBLE

INVISIBLE

função:

Ligar ou desligar o atributo de visibilidade do segmento.

C.8 Funções de Entrada

zm_locator (est, disp)

parâmetros:

est - identificador de estação

disp - número do dispositivo localizador

função:

Retornar um ponteiro para uma estrutura do tipo *ponto*, com os valores das coordenadas x e y (coordenadas do usuário) de um ponto solicitado pelo usuário.

zm_stroke (est, disp)

parâmetros:

est - identificador de estação

disp - número do dispositivo vetorizador

função:

Retornar um ponteiro para uma seqüência, em coordenadas do usuário, através do dispositivo vetorizador especificado.

zm_choice (est, disp)

parâmetros:

est - identificador de estação

disp - número do dispositivo de escolha

função:

Devolver um valor (numérico ou charater) que será interpretado pelo programa aplicativo como uma opção do usuário.

zm_string (est, disp)

parâmetros:

est - identificador de estação

disp - número do dispositivo

função:

Devolver uma seqüência de caracteres introduzidos através do dispositivo especificado, pertencente à estação dada.

APÊNDICE D
Listas Encadeadas

D.1 Estrutura de Informação

Uma estrutura de informação é [24] um "recipiente" que permite o armazenamento de informações. Esse "recipiente" é constituído por elementos moleculares chamados nós. Cada nó, por sua vez, é constituído por dois tipos de campos (figura D.1):

- campos de informação
- campos de associação ou relacionamento

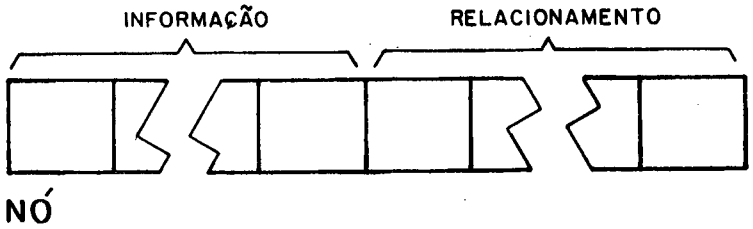


FIGURA D.1 - Nó de uma estrutura de informação

Em outras palavras, os vários nós interrelacionados através de seus campos de relacionamento constituem a estrutura de informação.

Em cima deste conceito são baseadas três estruturas:

- Arrays ou matrizes
- Listas lineares
- Arvores

As listas lineares são de grande utilização nos algorítmos dedicados à computação, gráfica por permitirem que se trabalhe com alocação dinâmica de memória.

D.2 Listas Lineares

Embora com algumas modificações em estruturação de dados, a lista simplesmente encadeada tem uso quase que dominante. Neste tipo de estrutura o nó tem o seguinte formato:

- um ou mais campos de informação, chamados genericamente de "INFO";
- e um campo de relacionamento, chamado "ELO".

Cada nó (figura D.2), terá associado um único endereço de armazenamento, aqui designado pela letra "E".

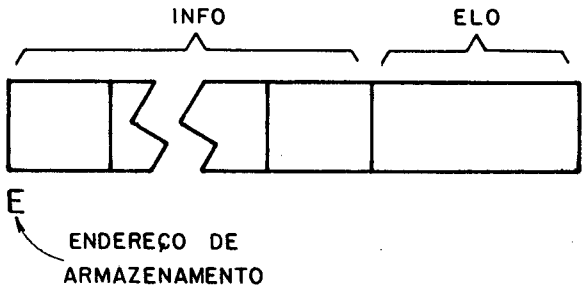


FIGURA D.2 - Nó de uma estrutura do tipo "Lista simplesmente encadeada".

O campo INFO conterá a informação associada a cada nó. O campo de ELO armazenará o endereço do nó subsequente na estrutura.

FIGURA D.3 - Dois nós relacionados pelo endereço 12.

O primeiro nó da figura D.3 tem como endereço 26, e tem armazenada a informação AC, tem ainda como sucessor o nó da estrutura de endereço 12. O segundo nó tem armazenada a informação BE, e o seu campo ELO tem valor zero, significando que não há sucessor.

Graficamente a figura D.3 pode ser representada como na figura D.4 abaixo.

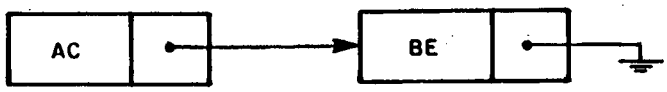


FIGURA D.4 - Representação gráfica de dois nós relacionados.

Nó cabeça

O endereço do primeiro nó de uma lista será guardado num campo, que genericamente é chamado "CAB".

Assim na estrutura de dados em forma de lista linear completa e fisicamente acessível pode ser esquematizada como na figura D.5.

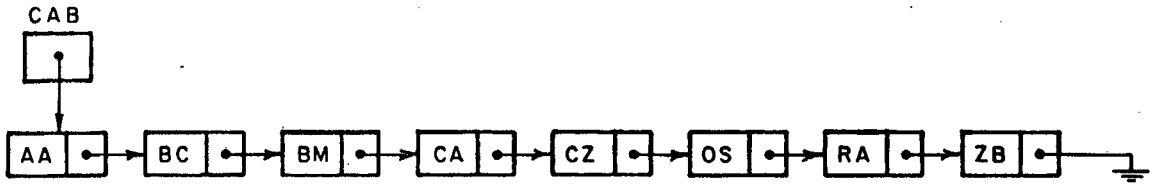


FIGURA D.5 - Esquema da estrutura de dados do tipo "lista linear".

Se CAB for nulo, significará que a lista estará vazia.