

# Apostila de Computação Gráfica

Agma Juci Machado Traina  
Maria Cristina Ferreira de Oliveira

16 de maio de 2006

### **Resumo**

Este é o material utilizado no Instituto de Ciências matemáticas e de Computação da USP-São Carlos para as disciplinas de computação gráfica ministradas pelas Profa. Dra. Agma Juci Machado Traina e Profa. Dra. Maria Cristina Ferreira de Oliveira.

# Sumário

<b>Sumário</b>	<b>2</b>
<b>Lista de Figuras</b>	<b>5</b>
<b>1 Introdução à Computação Gráfica</b>	<b>8</b>
1.1 Sistemas Gráficos . . . . .	9
1.2 Aplicações da CG . . . . .	10
1.3 Hardware Gráfico . . . . .	11
1.4 Resolução Gráfica . . . . .	11
1.5 Sistemas de Coordenadas . . . . .	12
1.6 Exercícios . . . . .	14
<b>2 Dispositivos de Visualização</b>	<b>15</b>
2.1 Dispositivos Gráficos Vetoriais . . . . .	15
2.1.1 Traçadores Digitais . . . . .	15
2.1.2 Dispositivos de Vídeo Vetoriais (Vector Refresh Display Tubes) . . . . .	15
2.1.3 Terminais CRT com memória (Direct View Storage Tubes) . . . . .	17
2.2 Primitivas de Software para Dispositivos Vetoriais . . . . .	17
2.3 Dispositivos Gráficos Matriciais . . . . .	17
2.3.1 Impressoras . . . . .	17
2.3.2 Dispositivos de Vídeo de Varredura (Raster Scanning VDUs) . . . . .	18
2.3.3 Primitivas de Software . . . . .	20
2.4 Exercícios . . . . .	21
2.5 Dispositivos de Entrada . . . . .	21
2.5.1 Teclado . . . . .	22
2.5.2 Light Pen . . . . .	22
2.5.3 Joystick . . . . .	22
2.5.4 Mouse . . . . .	22
2.5.5 Mesa Digitalizadora (Tablet) . . . . .	22
2.5.6 Data Glove . . . . .	22
2.5.7 Outros dispositivos . . . . .	22
<b>3 Traçado de Curvas em Dispositivos Gráficos Matriciais</b>	<b>23</b>
3.1 Simetria e Reflexão . . . . .	24
3.2 Conversão Matricial de Segmentos de Reta . . . . .	24
3.2.1 Características Desejáveis para os Algoritmos de conversão Matricial de Segmentos de Retas . . . . .	25
3.2.2 Critério Adotado . . . . .	25
3.2.3 Algoritmos . . . . .	26
3.2.4 Algoritmo do “Ponto-Médio” . . . . .	26
3.3 Conversão Matricial de Circunferências . . . . .	29
3.3.1 Simetria de ordem 8 . . . . .	30
3.3.2 Algoritmo do “Ponto-Médio” para Circunferências . . . . .	31
3.4 Conversão Matricial de Elipses . . . . .	34
3.5 Correção no Traçado . . . . .	37
3.6 Antialiasing . . . . .	38
3.6.1 Amostragem de Áreas não Ponderada . . . . .	38

<b>4 Preenchimento de Polígonos</b>	<b>42</b>
4.1 Retângulos . . . . .	42
4.2 Polígonos de Forma Arbitrária . . . . .	43
4.2.1 Arestas Horizontais . . . . .	45
4.2.2 Slivers . . . . .	46
4.2.3 Algoritmo para Conversão Matricial de Segmento de Reta que Utiliza “Coerência de Arestas” de um Polígono . . . . .	46
<b>5 Transformações 2D e 3D</b>	<b>50</b>
5.1 Transformações em 2D . . . . .	50
5.2 Coordenadas Homogêneas e Matrizes de Transformação . . . . .	52
5.3 Transformações 2D Adicionais: Espelhamento e Shearing . . . . .	55
5.3.1 Espelhamento (Mirror) . . . . .	55
5.3.2 Shearing . . . . .	56
5.4 Transformações entre sistemas de coordenadas . . . . .	57
5.5 Composição de Transformações . . . . .	58
5.6 Transformação Janela - Porta de Visão (“Window-to-Viewport”) . . . . .	59
5.7 Eficiência . . . . .	61
5.8 Transformações em 3D . . . . .	63
5.8.1 Composição de Transformações em 3D . . . . .	64
<b>6 Observação de Cenas 3D</b>	<b>69</b>
6.1 Pipeline de observação (“viewing pipeline”) . . . . .	69
6.2 Coordenadas de Observação . . . . .	70
6.2.1 Especificação do sistema de coordenadas de observação . . . . .	70
6.2.2 Transformação do sistema de coordenadas do mundo para o sistema de coordenadas de observação . . . . .	71
6.3 Projeções . . . . .	73
6.3.1 Projeção Perspectiva . . . . .	73
6.3.2 Desenvolvimento Matemático para Projeções Paralelas . . . . .	77
<b>7 Recorte de Primitivas 2D</b>	<b>78</b>
7.1 Recorte de segmentos de reta . . . . .	78
7.1.1 Recorte de Pontos Extremos . . . . .	79
7.1.2 Algoritmo de Cohen-Sutherland para Recorte de Segmentos de Reta . . . . .	79
7.2 Recorte de Circunferências . . . . .	81
<b>8 Curvas e Superfícies em Computação Gráfica</b>	<b>86</b>
8.1 Representação de Curvas . . . . .	86
8.2 Curve Fitting x Curve Fairing . . . . .	87
8.2.1 Ajuste de curvas (curve fitting) . . . . .	87
8.2.2 Aproximação de curvas (curve fairing) . . . . .	87
8.3 Representações Paramétricas e Não Paramétricas (explícita e implícita) . . . . .	87
8.3.1 Limitações das representações não paramétricas . . . . .	88
8.4 Curvas de Bézier . . . . .	90
<b>9 Apostila Modelagem</b>	<b>93</b>
<b>10 Rendering</b>	<b>94</b>
<b>11 Cores e Sistemas de Cores</b>	<b>95</b>
11.1 Percepção de Cor . . . . .	95
11.2 Sistemas de Cores Primárias . . . . .	96
11.3 Modelo XYZ . . . . .	97
11.4 Modelo RGB (Red, Green, Blue) . . . . .	99
11.5 Modelo HSV (Hue, Saturation, Value) . . . . .	100
11.6 Modelo HLS (Hue, Lightness, Saturation) . . . . .	100

<b>12 Processamento Digital de Imagens</b>	<b>102</b>
12.1 Introdução . . . . .	102
12.2 Considerações Sobre Imagens . . . . .	102
12.3 Tabelas “Look-up” . . . . .	103
12.4 Tipos de Manipulação de Imagens . . . . .	103
12.5 Transformações Radiométricas . . . . .	104
12.5.1 Operações Pontuais sobre Imagens . . . . .	105
12.5.2 Operações Locais Sobre a Imagem . . . . .	107
<b>Bibliografia</b>	<b>113</b>
<b>A Histórico</b>	<b>114</b>

# Listas de Figuras

1.1	Relacionamento entre as 3 subáreas da Computação Gráfica.	8
1.2	Esquema básico de um hardware de computação gráfica.	12
1.3	Sistemas de coordenadas e suas transformações.	13
2.1	Estrutura interna de um CRT.	16
2.2	Conversão Digital-Analógica para Visualização num CRT.	16
2.3	Uma seqüência de bits na memória de imagem é convertida para uma seqüência de pixels na tela.	18
2.4	Representação esquemática de uma imagem matricial e sua representação num <i>frame buffer</i> .	18
2.5	Varredura por rastreio fixo.	19
2.6	Representação esquemática de um CRT por varredura colorido.	19
2.7	Organização de uma vídeo <i>look-up table</i> .	20
3.1	Representação de segmentos de reta horizontais, verticais e diagonais.	23
3.2	Reflexão de uma imagem com relação a um eixo diagonal.	24
3.3	Rotação de 90° obtida através de 2 reflexões, uma horizontal (a) e outra na diagonal (b).	24
3.4	Conversões matriciais de um segmento de reta resultante de diferentes critérios.	25
3.5	Imagens de segmentos de reta convertidos pelo critério explícito acima.	26
3.6	Grade de pixels para o Algoritmo o Ponto-Médio (M) e as escolhas E e NE.	27
3.7	Grade de pixels para o Algoritmo o Ponto-Médio (M) e as escolhas E e NE.	29
3.8	Um arco de $\frac{1}{4}$ de circunferência, obtido variando-se $x$ em incrementos unitários, e calculando e arredondando $y$ .	29
3.9	Oito pontos simétricos em uma circunferência.	31
3.10	Malha de pixels para o Algoritmo do Ponto-Médio para circunferências, ilustrando a escolha entre os pixels E e SE.	32
3.11	Segundo octante da circunferência gerado com o algorítimo do Ponto-Médio e primeiro octante gerado por simetria.	32
3.12	Elipse padrão centrada na origem.	36
3.13	As duas regiões adotadas, definidas pela tangente a 45°.	36
3.14	Segmento de reta renderizado com o algorítmo do ponto médio em diferentes escalas. (a) é uma ampliação da região central de (b).	38
3.15	Segmento de reta definido com uma espessura diferente de zero.	39
3.16	A intensidade do pixel é proporcional à área coberta.	39
3.17	Filtro definido por um cubo para um pixel definido por um quadrado.	40
3.18	Filtro cônico com diâmetro igual ao dobro da largura de um pixel.	40
4.1	Esta figura ilustra o processo de linha de varredura para um polígono arbitrário. As intersecções da linha de varredura 8 com os lados FA e CD possuem coordenadas inteiras, enquanto as intersecções com os lados EF e DE possuem coordenadas reais.	43
4.2	Linhos de varredura em um polígono. Os extremos em preto, e os pixels no interior em cinza. (a) Extremo calculado pelo algoritmo do "Meio-Ponto". (b) Extremo interior ao polígono.	44
4.3	Tratamento dos lados horizontais de um polígono.	45
4.4	Exemplo de uma conversão matricial de um <i>Sliver</i> .	46
4.5	ET para o polígono de Figura 4.1.	48
4.6	AET para o polígono da Figura 4.1 a) linha de varredura 9 b) linha de varredura 10. Note que a coordenada x da aresta DE em (b) foi arredondada para cima.	48
5.1	Translação de uma casa.	51
5.2	Mudança de escala de uma casa. Como a escala é não uniforme, sua proporção é alterada.	51

5.3	Esta figura mostra a rotação da casa por $45^\circ$ . Da mesma forma que para a escala, a rotação também é feita em relação a origem. . . . .	52
5.4	Derivando a equação de rotação. . . . .	52
5.5	O espaço de Coordenadas Homogêneas $XYW$ , com o plano $W = 1$ e o ponto $P(X, Y, W)$ projetado sobre o plano $W = 1$ . . . . .	53
5.6	Um cubo unitário é rodados $45$ graus, posteriormente escalado não uniformemente. Obtem-se dessa forma uma transformação afim. . . . .	54
5.7	Reflexão de um objeto em torno do eixo $x$ . . . . .	55
5.8	Reflexão de um objeto em torno de um eixo perpendicular ao plano $xy$ , passando pela origem. . . . .	56
5.9	Rotação em relação ao Ponto $P_1$ , por um ângulo $\theta$ . . . . .	58
5.10	Escala e rotação de uma casa em relação ao ponto $P_1$ . . . . .	59
5.11	Janela em Coordenadas do mundo e porta de visão em coordenadas de tela. . . . .	60
5.12	Duas portas de visão associadas a mesma janela. . . . .	60
5.13	Os passos da transformação janela - porta de visão. . . . .	61
5.14	Primitivas gráficas de saída em coordenadas do mundo são recortadas pela janela. O seu interior é apresentado na tela (viewport). . . . .	61
5.15	Sistema de Coordenadas dado pela Regra da Mão Direita. . . . .	63
5.16	Transformando $P_1$ , $P_2$ e $P_3$ da posição inicial em (a) para a posição final em (b). . . . .	65
5.17	Rotação dos pontos $P_1$ , $P_2$ e $P_3$ . . . . .	66
5.18	Rotação em relação ao eixo $x$ . $P_1$ e $P_2$ de comprimento $D_2$ é rotacionado em direção ao eixo $z$ , pelo ângulo positivo $f$ . . . . .	66
5.19	Rotação em relação ao eixo $z$ . . . . .	67
5.20	Os vetores unitários $R_x$ , $R_y$ e $R_z$ , os quais são transformados nos eixos principais. . . . .	68
6.1	Atributos da câmera [Schröeder, 1998]. . . . .	69
6.2	Movimentos de câmera [Schröeder, 1998]. . . . .	70
6.3	Movimentos de câmera [Schröeder, 1998]. . . . .	71
6.4	Um VCS baseado na regra da mão direita com $x_v$ , $y_v$ e $z_v$ .relativos ao sistema de coordenadas do mundo [Hearn and Baker, 1994]. . . . .	71
6.5	Orientações do plano de observação conforme o vetor normal. O vetor $(1, 0, 0)$ dá a orientação em (a) e $(1, 0, 1)$ , a orientação em (b) [Hearn and Baker, 1994]. . . . .	72
6.6	Taxomia de projeções [Plastock and Kalley, 1999]. . . . .	73
6.7	Linha AB e sua projeção A'B': (a) perspectiva; (b) ortogonal. . . . .	73
6.8	Projeções de um cubo (com 1 ponto de fuga) sobre um plano cortando o eixo Z, apresentando o ponto de fuga. . . . .	74
6.9	Projeções perspectivas com 2 pontos de fuga ( o plano de projeção intercepta 2 eixos ( $x$ e $z$ )). . . . .	74
6.10	Projeções perspectivas com 3 pontos de fuga ( o plano de projeção intercepta os 3 eixos). . . . .	75
6.11	A esfera B é bem maior que a esfera A, porém ambas aparecem com o mesmo tamanho quando projetadas no plano de visão. . . . .	75
6.12	Confusão visual da perspectiva (objeto atrás do centro de projeção). . . . .	76
6.13	Projeção em perspectiva de um ponto $P=(x, y, z)$ na posição $(xp, yp, zp)$ sobre o plano de projeção [Hearn and Baker, 1994]. . . . .	76
6.14	Projeção ortogonal de um ponto no plano de projeção [Hearn and Baker, 1994]. . . . .	77
7.1	Exemplos de recorte de segmentos de reta. . . . .	78
7.2	Códigos das regiões. . . . .	80
7.3	Funcionamento do algoritmo de Cohen-Sutherland para recorte de segmentos. . . . .	80
7.4	Algoritmo de Sutherland-Hodgman para Recorte de Polígonos. . . . .	83
7.5	Exemplos de recorte de polígonos. (a) Múltiplos componentes. (b) Caso convexo (c) Caso côncavo com muitas arestas exteriores. . . . .	83
7.6	Quatro situações possíveis no recorte de polígonos. . . . .	84
8.1	Representação de superfícies através de malhas. . . . .	86
8.2	Aproximação de curvas por segmentos de reta conectados. . . . .	87
8.3	Aproximação de curvas por segmentos de reta conectados. . . . .	88
8.4	Pontos de uma circunferência. . . . .	90
8.5	Representação de uma curva de Bézier definida pelos pontos $B_0$ , $B_1$ , $B_2$ e $B_3$ . . . . .	90
8.6	Funções de blending para vários valores de n. . . . .	91
11.1	Espectro eletromagnético [Gro94]. . . . .	95

11.2 Ilustração da mistura de cores aditivas [For94]. . . . .	97
11.3 Ilustração da mistura de cores subtrativas [For94]. . . . .	97
11.4 Ilustração da obtenção de tints, shades e tones. . . . .	97
11.5 Diagrama de cromaticidade do CIE [Hea94]. . . . .	98
11.6 Representação de escalas de cor no diagrama de cromaticidade do CIE [Hea94]. . . . .	99
11.7 Cubo do RGB [Fol96]. . . . .	100
11.8 Cone hexagonal do HSV [Fol96]. . . . .	100
11.9 Cone duplo do HLS [Fol96]. . . . .	101
 12.1 Efeitos sobre uma imagem, de se reduzir o tamanho da grade de amostragem. . . . .	103
12.2 Uma imagem de 512 x 512 pixels, quantizada em (a) 256 níveis, (b) 128 níveis, (c) 64 níveis, (d) 32 níveis, (e) 16 níveis, (f) 8 níveis, (g) 4 níveis e (h) 2 níveis de cinza. . . . .	104
12.3 Representação de um pixel $P$ com 8-bits (8 planos) de profundidade. . . . .	104
12.4 Organização de uma LUT, para um sistema de pixels de 8 bits. . . . .	105
12.5 Imagens e histogramas de intensidade. . . . .	106
12.6 Efeito de uma transformação dos níveis de cinza efetuado sobre o histograma. . . . .	107
12.7 Manipulação de janelas de intensidade da imagem através do histograma. . . . .	107
12.8 Sinais “ruído” (a) e “degrau” (b). . . . .	109
12.9 Sinais “ruído” (a) e “degrau” (b). . . . .	109
12.10 Efeitos de filtros lineares ao “ruído” e à “borda”: (a) efeitos de atenuação; (b) borramento da borda; (c) ampliação do ruído; (d) realce da borda. . . . .	110
12.11 Efeito do filtro da mediana sobre o “ruído” e o “degrau”. . . . .	110
12.12 Uma transformação geométrica de 90° graus. . . . .	111

# Capítulo 1

## Introdução à Computação Gráfica

A Computação Gráfica é a área da ciência da computação que estuda a geração, manipulação e interpretação de modelos e imagens de objetos utilizando computador. Tais modelos vêm de uma variedade de disciplinas, como física, matemática, engenharia, arquitetura, etc.

Pode-se relacionar a Computação Gráfica com 3 sub-áreas [Persiano and Oliveira, 1989]:

- **Síntese de Imagens:** área que se preocupa com a produção de representações visuais a partir das especificações geométrica e visual de seus componentes. É freqüentemente confundida com a própria Computação Gráfica. As imagens produzidas por esta sub-área são geradas a partir de dados mantidos nos chamados *Display-Files*.
- **Processamento de Imagens:** envolve as técnicas de transformação de Imagens, em que tanto a imagem original quanto a imagem resultado apresentam-se sob uma representação visual (geralmente matricial). Estas transformações visam melhorar as características visuais da imagem (aumentar contraste, foco, ou mesmo diminuir ruídos e/ou distorções). As imagens produzidas/utilizadas por esta sub-área são armazenadas/recuperadas dos chamados *Raster-Files*.
- **Análise de Imagens:** área que procura obter a especificação dos componentes de uma imagem a partir de sua representação visual. Ou seja a partir da informação pictórica da imagem (a própria imagem!) produz uma informação não pictórica da imagem (por exemplo, as primitivas geométricas elementares que a compõem).

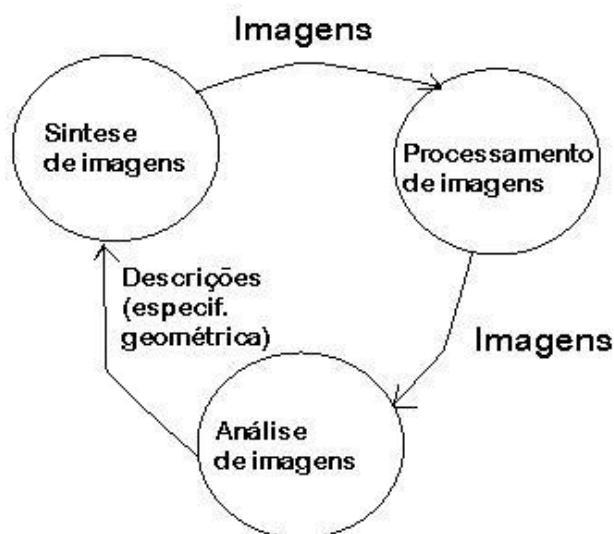


Figura 1.1: Relacionamento entre as 3 subáreas da Computação Gráfica.

Na última década somou-se a esse contexto a área de Visualização de Dados, também chamada Visualização Computacional [Minghim and Oliveira, 1997, Schröeder et al., 1996], que usa técnicas de Computação Gráfica para representar informação, de forma a facilitar o entendimento de conjuntos de dados numéricos de alta complexidade. Exemplos de áreas de aplicação são: visualização de imagens médicas, meteorologia, dados financeiros, visualização de programas, dinâmica dos fluidos, e muitas

outras. Nelas, o que existe em comum é que a representação gráfica (superfícies, partículas, ícones) são geradas automaticamente a partir do conjunto de dados. Ao usuário cabe definir parâmetros e atributos da imagem para melhor “navegar” seu conjunto de dados. Dessa maneira, a visualização de dados partilha de características da síntese, do processamento e da análise de dados.

Atualmente a Computação Gráfica é altamente interativa: o usuário controla o conteúdo, a estrutura e a aparência dos objetos e suas imagens visualizadas na tela, usando dispositivos como o teclado e o mouse. Entretanto, até o início dos anos 80, a computação gráfica era uma disciplina restrita e altamente especializada. Devido principalmente ao alto custo do hardware, poucos programas aplicativos exploravam gráficos. O advento dos computadores pessoais de baixo custo, como o IBM-PC e o Apple Macintosh, com terminais gráficos de varredura (raster graphics displays), popularizou o uso de gráficos na interação usuário-computador.

Os *displays* gráficos de baixo custo possibilitaram o desenvolvimento de inúmeros aplicativos baratos e fáceis de usar, que dispunham de interfaces gráficas - planilhas, processadores de texto, programas de desenho... As interfaces evoluíram e introduziu-se o conceito de desktop - uma metáfora para uma mesa de trabalho. Nessas interfaces gráficas, através de um gerenciador de janelas (*window manager*) o usuário pode criar e posicionar janelas que atuam como terminais virtuais, cada qual executando aplicativos independentemente. Isso permite que o usuário execute vários aplicativos simultaneamente, e selecione um deles a um simples toque no mouse. Ícones (*icons*) são usados para representar arquivos de dados, programas e abstrações de objetos de um escritório - como arquivos, caixas de correio (*mailboxes*), impressoras, latas de lixo - nas quais são executadas operações análogas às da vida real. Para ativar os programas, o usuário pode selecionar ícones, ou usar buttons e menus dinâmicos. Objetos são manipulados diretamente através de operações de *pointing* e *clicking* feitas com o mouse. Atualmente, mesmo aplicativos que manipulam texto (como processadores de texto) ou dados numéricos (como planilhas) usam interfaces desse tipo, reduzindo sensivelmente a interação textual por meio de teclados alfanuméricos.

A computação gráfica não é mais uma raridade: é parte essencial de qualquer interface com o usuário, é indispensável para a visualização de dados em 2D e 3D e tem aplicações em áreas como educação, ciências, engenharia, medicina, publicidade, lazer, militar, ...

A computação gráfica trata da síntese de imagens de objetos reais ou imaginários a partir de modelos computacionais. Processamento de imagens é uma área relacionada que trata do processo inverso: a análise de cenas, ou a reconstrução de modelos de objetos 2D ou 3D a partir de suas imagens.

Note que a síntese de imagens parte da descrição de objetos tais como segmentos de reta, polígonos, poliedros, esferas, etc.; e produz uma imagem que atende a certas especificações e que pode, em última instância, ser visualizada em algum dispositivo (terminal de vídeo, *plotter*, impressora, filme fotográfico...). As imagens em questão constituem uma representação visual de objetos bi- ou tridimensionais descritos através de especificações abstratas.

O processamento de imagens parte de imagens já prontas para serem visualizadas, as quais são transferidas para o computador por mecanismos diversos - digitalização de fotos, tomadas de uma câmera de vídeo, ou imagens de satélite - para serem manipuladas visando diferentes objetivos.

## 1.1 Sistemas Gráficos

A Computação Gráfica (especialmente as componentes relativas a gráficos 3D e a gráficos 3D interativos) desenvolveu-se de modo bem diverso: de simples programas gráficos para computadores pessoais à programas de modelagem e de visualização em *workstations* e supercomputadores. Como o interesse em CG cresceu, é importante escrever aplicações que possam rodar em diferentes plataformas. Um padrão para desenvolvimento de programas gráficos facilita esta tarefa eliminando a necessidade de escrever código para um *driver* gráfico distinto para cada plataforma na qual a aplicação deve rodar. Para se padronizar a construção de aplicativos que se utilizam de recursos gráficos e torná-los o mais independentes possível de máquinas, e portanto facilmente portáveis, foram desenvolvidos os chamados Sistemas Gráficos.

Vários padrões tiveram sucesso integrando domínios específicos. Por exemplo, a linguagem *Postscript* que se tornou um padrão por facilitar a publicação de documentos estáticos contendo gráficos 2D e textos. Outro exemplo é o sistema *XWindow*, que se tornou padrão para o desenvolvimento de interfaces gráficas 2D em *workstations* UNIX. Um programador usa o X para obter uma janela em um *display* gráfico no qual um texto ou um gráfico 2D pode ser desenhado. A adoção do X pela maioria dos fabricantes de *workstation* significa que um único programa desenvolvido em X pode ser rodado em uma variedade de *workstation* simplesmente recompilando o código. Outra facilidade do X é o uso de redes de computadores: um programa pode rodar em uma *workstation* e ler a entrada e ser exibido em outra *workstation*, mesmo de outro fabricante.

Para gráficos 3D foram propostos vários padrões. A primeira tentativa foi o Sistema Core - *Core Graphics System* - (1977 e 1979) pelos americanos. Mas a primeira especificação gráfica realmente pa-

dronizada foi o GKS - *Graphical Kernel System*, pela ANSI e ISO em 1985. O GKS é uma versão mais elaborada que o Core. O GKS suporta um conjunto de primitivas gráficas interrelacionadas, tais como: desenho de linhas, polígonos, caracteres, etc., bem como seus atributos. Mas não suporta agrupamentos de primitivas hierárquicas de estruturas 3D. Um sistema relativamente famoso é PHIGS (*Programmer's Hierarchical Interactive Graphics System*). Baseado no GKS, PHIGS é um padrão ANSI. PHIGS (e seu descendente, PHIGS+) provêem meios para manipular e desenhar objetos 3D encapsulando descrições de objetos e atributos em uma *display list*. A *display list* é utilizada quando o objeto é exibido ou manipulado, uma vantagem é a possibilidade de descrever um objeto complexo uma única vez mesmo exibindo-o várias vezes. Isto é especialmente importante se o objeto a ser exibido deve ser transmitido por uma rede de computadores. Uma desvantagem da *display list* é a necessidade de um esforço considerável para reespecificar um objeto que está sendo modelado interativamente pelo usuário. Uma desvantagem do PHIGS e PHIGS+ (e GKS) é que eles não têm suporte a recursos avançados como mapeamento de textura.

O *XWindow* ganhou uma extensão para o PHIGS, conhecida como PEX, de modo que o X pudesse manipular e desenhar objetos 3D. Entre outras extensões, PEX soma-se de modo imediato ao PHIGS, assim um objeto pode ser exibido durante a sua definição sem a necessidade da *display list*. O PEX também não suporta recursos avançados e só está disponível aos usuários do *XWindow*.

O sistema gráfico mais popular atualmente é o OpenGL (*GL - Graphics Library*) que provê características avançadas e pode ser utilizado em modo imediato ou com *display list*. OpenGL é um padrão relativamente novo (sua primeira versão é de 1992) e é baseado na biblioteca GL das *workstations* IRIS da *Silicon Graphics*. Atualmente um consórcio de indústrias é responsável pela gerenciamento da evolução do OpenGL. Existe uma implementação livre (código fonte disponível) do OpenGL conhecida com *MesaGL* ou *Mesa3D*.

Como os outros sistemas gráficos, OpenGL oferece uma interface entre o software e o hardware gráfico. A interface consiste em um conjunto de procedimentos e funções que permitem a um programador especificar os objetos e as operações que os envolvem produzindo imagens de alta qualidade. Como o PEX, o OpenGL integra/permite a manipulação de objetos (desenhos) 3D ao X, mas também pode ser integrado em outros sistemas de janela (por exemplo, Windows/NT) ou pode ser usado sem um sistema de janela. OpenGL provê controle direto sobre operações gráficas fundamentais em 3D e 2D, incluindo a especificação de parâmetros como matrizes de transformação e coeficientes de iluminação, métodos de *antialiasing* e operações sobre pixels, mas não provê mecanismos para descrever ou modelar objetos geométricos complexos.

## 1.2 Aplicações da CG

A lista de aplicações é enorme, e cresce rapidamente. Uma amostra significativa inclui:

- **Interfaces:** a maioria dos aplicativos para computadores pessoais e estações de trabalho atualmente dispõem de interfaces gráficas baseadas em janelas, menus dinâmicos, ícones, etc.
- **Traçado interativo de gráficos:** aplicativos voltados para usuários em ciência, tecnologia e negócios geram gráficos que ajudam na tomada de decisões, esclarecem fenômenos complexos e representam conjuntos de dados de forma clara e concisa.
- **Automação de escritórios e editoração eletrônica:** o uso de gráficos na disseminação de informações cresceu muito depois do surgimento de software para editoração eletrônica em computadores pessoais. Este tipo de software permite a criação de documentos que combinam texto, tabelas e gráficos - os quais tanto podem ser “desenhados” pelo usuário ou obtidos a partir de imagens digitalizadas.
- **Projeto e desenho auxiliado por computador:** em CAD, sistemas gráficos interativos são utilizados para projetar componentes, peças e sistemas de dispositivos mecânicos, elétricos, eletromecânicos e eletrônicos. Isto inclui edifícios, carcaças de automóveis, aviões e navios, chips VLSI, sistemas óticos, redes telefônicas e de computador. Eventualmente, o usuário deseja apenas produzir desenhos precisos de componentes e peças. Mais frequentemente, o objetivo é interagir com um modelo computacional do componente ou sistema sendo projetado, de forma a testar propriedades estruturais, elétricas ou térmicas, até atingir um projeto satisfatório.
- **Simulação e animação para visualização científica, lazer, arte e publicidade:** uma das áreas que mais evoluíram na década de 80 foi a visualização científica. Cientistas e engenheiros perceberam que não poderiam interpretar as quantidades prodigiosas de dados produzidas por programas em supercomputadores sem resumir os dados e identificar tendências e fenômenos através

de representações gráficas. Como resultado, surgiram animações computadorizadas do comportamento variante no tempo de objetos reais ou simulados. Tais animações podem ser utilizadas para estudar entidades matemáticas abstratas e modelos matemáticos de fenômenos como fluxo de fluidos, relatividade, reações químicas e nucleares, deformação de estruturas mecânicas sob diferentes tipos de pressão, etc. Outras aplicações tecnológicas avançadas incluem a produção de desenhos animados e efeitos especiais para filmes e comerciais de TV, que requerem mecanismos sofisticados para modelar objetos e para representar luz e sombra.

- **Controle de processos:** sistemas de controle de tráfego aéreo e espacial, sistemas de controle de refinarias e de usinas de energia mostram graficamente os dados coletados por sensores conectados a componentes críticos dos sistemas, de forma que os operadores possam responder adequadamente a condições críticas.
- **Cartografia:** a computação gráfica é usada para produzir representações precisas e esquemáticas de fenômenos naturais e geográficos obtidos a partir da coleta de dados.
- **Arte:** A arte por computador vem crescendo imensamente nos últimos anos. É possível utilizar novos recursos de computação gráfica para produzir efeitos artísticos, como a extração de texturas, padrões e estruturas a partir de fotos digitalizadas.
- **Gráficos de Apresentação (Presentation Graphics):** É a utilização de técnicas gráficas para demonstração de resultados, idéias e gráficos, com o intuito de mostrar ou transmitir conhecimento específico como, por exemplo, em uma aula, ou reunião, ou na contrução de material didático.

Neste curso, estamos interessados principalmente em técnicas de síntese de imagens bem como seu relacionamento com o processamento de imagens. Também não pretendemos explorar todo o escopo de aplicações da CG, mas estudar os conceitos gerais envolvidos na programação das primitivas gráficas que estão por trás dessas aplicações.

### 1.3 Hardware Gráfico

Um sistema de hardware para computação gráfica consiste essencialmente de dispositivos gráficos de entrada e saída (I/O) ligados a um computador (Figura 1.2). Ao conjunto de dispositivos de I/O gráficos alocados para utilização por uma única pessoa por vez denomina-se genericamente de “estação de trabalho gráfica”, ou *graphics workstation*. Um sistema gráfico multi-usuário pode ter várias estações gráficas, de forma que mais de um dos vários dispositivos de I/O disponíveis podem estar conectados e utilizando o computador hospedeiro. Como em CG é freqüente a manipulação de grandes quantidades de dados, o computador deve ser equipado de memória secundária com alta capacidade de armazenagem. Além disso, um canal de comunicação de alta velocidade é necessário para reduzir os tempos de espera. Isto normalmente é feito através de comunicação local sobre um barramento paralelo com velocidade de transmissão de dados da ordem de um milhão de bits por segundo. Se o equipamento gráfico está distante do processador (conexão remota), um canal de comunicação serial pode ser necessário. Transmissão serial assíncrona pode ser feita a velocidades de até 19.2 kbps (milhares de bits por segundo). Mesmo tal velocidade pode ser muito lenta para alguns objetivos como animação gráfica de alta resolução, onde cada *frame* de imagem pode ter um *megabyte* de dados. Um sistema de conexão ideal nesse caso é uma Local Area Network (LAN) como a *Ethernet*. Os dispositivos de saída gráficos podem ser de natureza digital ou analógica, resultando em duas classes de gráficos, denominados *vector graphics* (gráficos vetoriais), que desenham figuras traçando sequências de segmentos de reta (vetores); e *raster graphics* (gráficos de varredura, ou matriciais), que desenham figuras pelo preenchimento de uma matriz de pontos (pixels).

### 1.4 Resolução Gráfica

Virtualmente todos os dispositivos de I/O gráficos usam uma malha retangular de posições endereçáveis - a qual é denominada “retângulo de visualização”. A “resolução gráfica” de um dispositivo é o número de posições (ou pontos, ou pixels) horizontais e verticais que ele pode distinguir. Existem 4 parâmetros que definem a resolução:

1. *ndh* - o número de posições endereçáveis horizontalmente.
2. *ndv* - o número de posições endereçáveis verticalmente.
3. *width* - a largura do retângulo de visualização em mm.

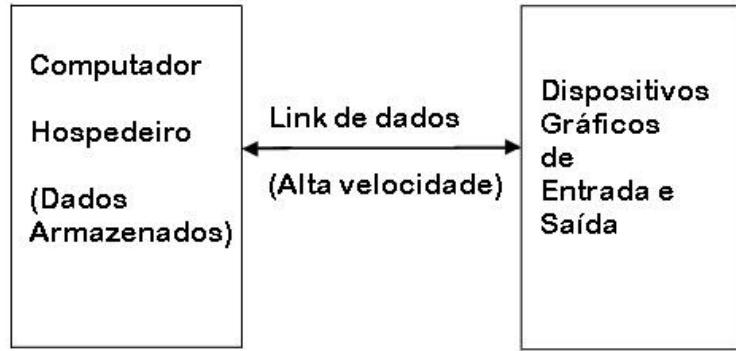


Figura 1.2: Esquema básico de um hardware de computação gráfica.

4. *height* - a altura do retângulo de visualização em mm.

A partir desses 4 parâmetros, vários números interessantes podem ser calculados:

1. **resolução horizontal:**  $horiz\_res = \frac{ndh}{width}$
2. **tamanho ponto horizontal:**  $horiz\_dot\_size = \frac{width}{ndh}$
3. **resolução vertical:**  $vert\_res = \frac{ndv}{height}$
4. **tamanho ponto vertical:**  $vert\_dot\_size = \frac{height}{ndv}$
5. **total pontos endereçáveis:**  $total\_nr\_dots = ndh \cdot ndv$
6. **resolução de área:**  $area\_res = \frac{total\_nr\_dots}{(width \cdot height)}$
7. **razão de aspecto gráfica:**  $aspect\_ratio = \frac{vert\_dot\_size}{horiz\_dot\_size}$
8. **razão de aspecto física:**  $physical\_aspect\_ratio = \frac{height}{width}$

Note que *horiz\_res*, *vert\_res* e *area\_res* definem resoluções físicas, enquanto que *ndh*, *ndv* e *total\_nr\_dots* definem resoluções gráficas. Dispositivos de visualização podem ter a mesma resolução gráfica, com resoluções físicas muito diferentes. O ideal seria ter um *aspect ratio* igual ou próximo de 1.

## 1.5 Sistemas de Coordenadas

Na CG é necessário definir sistemas de coordenadas para quantificar os dados que estão

sendo manipulados. Já vimos que os dispositivos de visualização gráfica matriciais consistem de uma matriz de pixels endereçáveis, e um gráfico é formado “acendendo” ou “apagando” um pixel. Os pixels são endereçados por dois números inteiros que dão suas coordenadas horizontal e vertical, *dCx*, e *dCy*, respectivamente, onde:

$$0 \leq dCx \leq ndhm1 \equiv ndh - 1 \quad (1.1)$$

$$0 \leq dCy \leq ndvm1 \equiv ndv - 1 \quad (1.2)$$

Na matriz de pixels, o valor *dCx* + 1 dá o número da coluna, e *dCy* + 1 dá o número da linha do pixel endereçado. O pixel endereçado como (0,0) está geralmente no canto inferior esquerdo do retângulo de visualização. As coordenadas (*dCx*, *dCy*) são chamadas de coordenadas do dispositivo, e podem assumir apenas valores inteiros. Coordenadas do dispositivo podem variar bastante para diferentes equipamentos, o que levou à utilização de coordenadas normalizadas do dispositivo (*NDC* - *normalized device coordinates*), para efeito de padronização (*ndCx*, *ndCy*). NDCs são variáveis reais, geralmente definidas no intervalo de 0 a 1:

$$0 \leq ndCx \leq 1 \quad (1.3)$$

$$0 \leq ndcy \leq 1 \quad (1.4)$$

A coordenada NDC  $(0, 0)$  corresponde à origem  $(0, 0)$  nas coordenadas do dispositivo, e a coordenada NDC  $(1, 1)$  refere-se ao pixel no canto superior direito, que corresponde ao pixel  $(ndhm1, ndvm1)$  nas coordenadas do dispositivo. A vantagem da utilização de NDCs é que padrões gráficos podem ser discutidos usando um sistema de coordenadas independente de dispositivos gráficos específicos. Obviamente, os dados gráficos precisam ser transformados do sistema de coordenadas independente para o sistema de coordenadas do dispositivo no momento de visualização. O mapeamento de NDCs (reais) para coordenadas do dispositivo (inteiros) é “linear”, por exemplo:

$$dcx = \text{round}(ndcx.ndhm1) \quad (1.5)$$

$$dcy = \text{round}(ndcy.ndvm1) \quad (1.6)$$

Dois outros sistemas de coordenadas são úteis. O primeiro é o sistema de coordenadas físico,  $(pcx, pcy)$ , onde  $pcx$  é a distância física ao longo do eixo  $x$  a partir do extremo esquerdo do retângulo de visualização, e  $pcy$  é a distância física ao longo do eixo  $y$  a partir do extremo inferior. As unidades de medida utilizadas são polegadas ou milímetros. A transformação de coordenadas físicas para coordenadas do dispositivo é dada por:

$$dcx = \text{trunc}(ndhm1 \frac{pcx}{width}) \quad (1.7)$$

$$dcy = \text{trunc}(ndvm1 \frac{pcy}{height}) \quad (1.8)$$

O segundo é o sistema de coordenadas do mundo, ou sistema de coordenadas do usuário, que consiste de coordenadas cartesianas  $(x, y)$ , num intervalo qualquer definido pelo usuário:

$$xmin \leq x \leq xmax \quad (1.9)$$

$$ymin \leq y \leq ymax \quad (1.10)$$

Os parâmetros que definem o intervalo de valores de  $x$  e  $y$ ,  $xmin$ ,  $ymin$ ,  $xmax$  e  $ymax$ , definem uma área retangular no espaço bidimensional, denominada de janela. A transformação de coordenadas do usuário  $(x, y)$  para NDCs  $(ndcx, ndcy)$ , denominada transformação de visualização, é dada por:

$$ndcx = \frac{x - xmin}{xmax - xmin} \quad (1.11)$$

$$ndcy = \frac{y - ymin}{ymax - ymin} \quad (1.12)$$

Para visualizar dados num dispositivo gráfico qualquer, é necessário transformá-los das coordenadas do usuário para NDCs, e de NDCs para coordenadas do dispositivo. Da mesma forma, dados de entrada gráficos precisam ser transformados de coordenadas do dispositivo para NDCs, e depois para coordenadas do usuário (Figura 1.3).

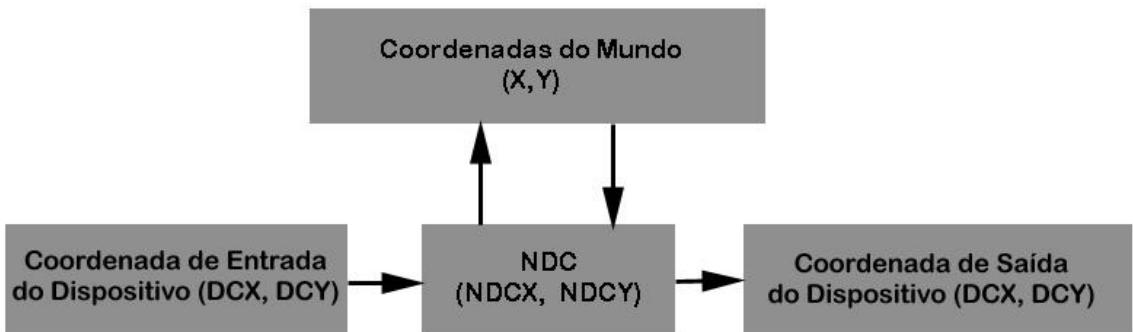


Figura 1.3: Sistemas de coordenadas e suas transformações.

## 1.6 Exercícios

Escreva os procedimentos *inp\_to\_ndc*, *ndc\_to\_user*, *user\_to\_ndc* e *ndc\_to\_dc*, que transformam dados entre os vários sistemas de coordenadas, conforme ilustrado na Figura 1.3. Repita o exercício assumindo que o intervalo de variação do sistema NDC vai de:

- (i) -1 a +1 (coordenadas normalizadas centradas)
- (ii) 0 a 100

## Capítulo 2

# Dispositivos de Visualização

Toda imagem criada através de recursos computacionais deve ser representada em algum dispositivo físico que permita a sua visualização. Diversas tecnologias e diferentes tipos de dispositivos são utilizados para gerar representações visuais, sendo que o desenvolvimento dessas tecnologias teve um papel fundamental na evolução da CG.

Tanto para o usuário como para o implementador de sistemas gráficos é importante conhecer as características de cada uma dessas tecnologias para sua melhor utilização. Vamos discutir alguns aspectos da arquitetura e organização dos tipos mais comuns dos dispositivos de exibição gráfica, sem entrar em detalhes técnicos.

É possível classificar os dispositivos de exibição (traçadores, impressoras e terminais de vídeo) em duas principais categorias, segundo a forma pela qual as imagens são geradas: dispositivos vetoriais e dispositivos matriciais. Os dispositivos gráficos vetoriais conseguem traçar segmentos de reta perfeitos entre dois pontos da malha finita de pontos definida por suas superfícies de exibição. Os dispositivos matriciais, por outro lado, apenas conseguem traçar pontos, também em uma malha finita. Assim, segmentos de reta são traçados como sequências de pontos próximos.

## 2.1 Dispositivos Gráficos Vetoriais

### 2.1.1 Traçadores Digitais

**Traçadores (plotters)** são dispositivos eletromecânicos que produzem o desenho pelo movimento de uma caneta sobre a superfície do papel. A primitiva gráfica básica nesse tipo de dispositivo é o segmento de reta. Arcos, curvas e caracteres são produzidos pelo traçado de uma série de pequenos segmentos.

Nos traçadores de mesa, o papel é fixado sobre uma superfície plana retangular, sobre a qual está localizado um braço mecânico que movimenta-se por translação. Ao longo do braço desloca-se um cabeçote que suporta uma caneta perpendicularmente à mesa, a qual pode ser pressionada contra o papel ou levantada de forma a não tocá-lo.

Nos traçadores de rolo, o braço é fixo, e o papel é movimentado para frente e para trás por ação de um rolo, como em uma máquina de escrever.

Embora distintos em construção, estes dois tipos de traçadores possuem características de programação e controle similares. A posição da caneta sobre o papel é definida pelo posicionamento do braço em relação ao papel (abcissa  $x$ ), e do cabeçote sobre o braço (ordenada  $y$ ). Figuras são traçadas pela variação controlada da posição da caneta (abcissa e ordenada) e pelo controle do estado da caneta (abaixada ou levantada). O traçador é em geral controlado por um processador dedicado que recebe instruções diretamente do computador ou de um arquivo que descreve o desenho.

### 2.1.2 Dispositivos de Vídeo Vetoriais (Vector Refresh Display Tubes)

No início da CG, o principal dispositivo de vídeo não era um monitor parecido com uma TV, e sim um caríssimo CRT (*Cathode Ray Tube*) do tipo usado em osciloscópios. Como o *display* dos osciloscópios, os monitores tinham como entradas duas voltagens,  $x$  e  $y$ , que direcionavam um feixe de elétrons para um ponto específico da tela. O feixe traçava uma linha do último ponto para o corrente, num único movimento vetorial.

Um CRT consiste basicamente de uma superfície de exibição, quase plana, recoberta internamente de material à base de fósforo, um canhão emissor de elétrons e um sistema de deflexão (Figura 2.1). O canhão emite um fino feixe de elétrons que, acelerados, chocam-se contra a superfície fosforecente da tela. Sob a ação dos elétrons, o material fosforecente incandesce, emitindo luz no ponto da tela atingido pelo

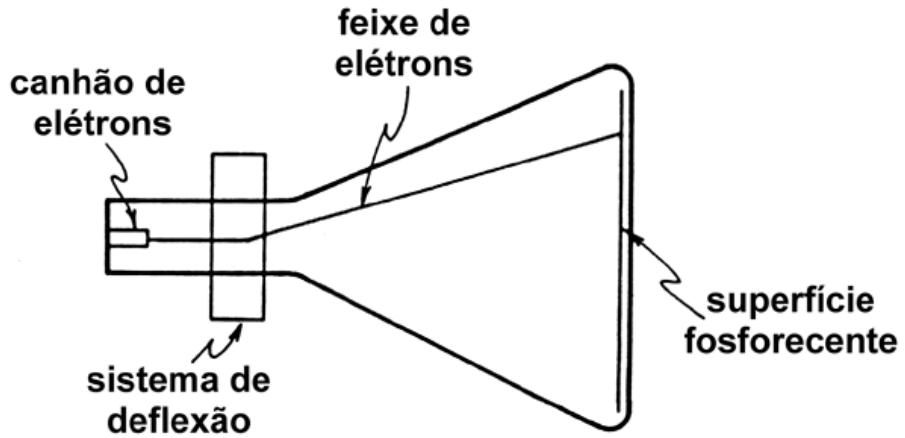


Figura 2.1: Estrutura interna de um CRT.

feixe. A função do sistema de deflexão é dirigir controladamente o feixe de elétrons para um determinado ponto da tela.

O brilho do fósforo dura apenas alguns milisegundos (a emissão de luz pelo fósforo não é estável, e cai a zero logo após a interrupção do bombardeio de elétrons), de forma que toda a figura precisa ser continuamente retraçada para que o gráfico permaneça na tela. Este processo é denominado *refreshing* (daí o nome, *vector refreshing tubes*). Se a imagem sendo mostrada é composta por muitos vetores, vai haver um atraso significativo entre o traçado do primeiro e do último vetores, e alguns dos vetores traçados inicialmente podem desaparecer nesse período. O resultado é que o tubo não consegue retraçar a imagem de modo suficientemente rápido para evitar que um efeito de flickering (“cintilação”) torne-se aparente na tela.

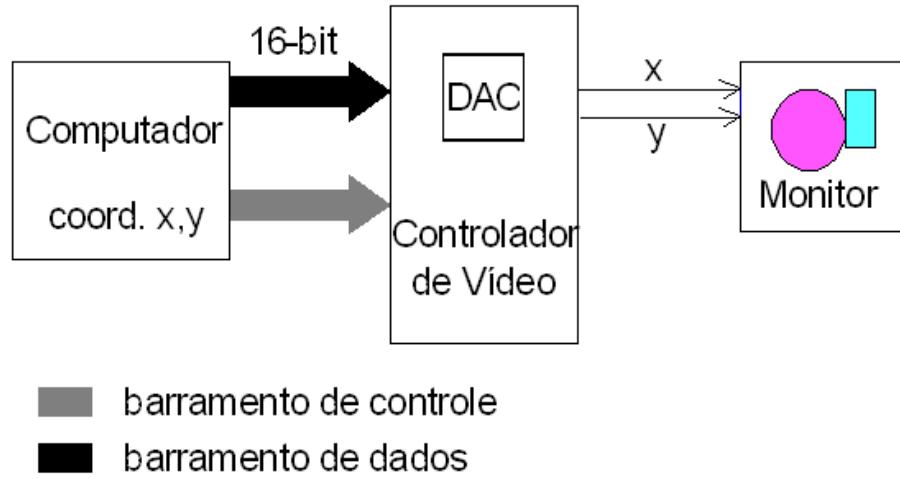


Figura 2.2: Conversão Digital-Analógica para Visualização num CRT.

O tubo não exige muita memória para manter uma imagem complexa construída por segmentos de reta, uma vez que apenas as coordenadas dos extremos dos segmentos, e as dos cantos da tela precisam ser armazenadas. Esta era uma característica importante no início da CG, já que a memória era muito cara.

O computador gera as coordenadas dos pontos que definem a figura a ser mostrada na tela, e um DAC (conversor digital-analógico) é necessário para converter os pontos digitais em voltagens a serem enviadas para o CRT (Figura 2.2).

As desvantagens dos terminais gráficos vetoriais eram: a tecnologia cara, o efeito de flickering, e a memória limitada, que inviabilizava a descrição de imagens com detalhes complexos. As vantagens: dispositivo gráfico de alta resolução (pelo menos 1000x1000), rapidez na geração de imagens simples, o que os torna adequados para testes iniciais em animação.

### 2.1.3 Terminais CRT com memória (Direct View Storage Tubes)

O CRT com memória é um tubo especial que se comporta como se o fósforo fosse de altíssima persistência. Uma imagem traçada nesse tipo de tubo pode manter-se por mais de uma hora sem sofrer desvanecimento. Entretanto, o tubo é construído com fósforo comum, e a longa persistência da imagem é conseguida pela inclusão de dois elementos adicionais ao tubo: um canhão espalhador de elétrons e uma máscara montada sobre a superfície do fósforo, entre a tela e os canhões.

Os elétrons são uniformemente espalhados por toda a superfície da tela pelo canhão espalhador, e a máscara funciona como um filtro que seleciona os pontos ou áreas da superfície a serem atingidas pelos elétrons. Para isso, a máscara é construída de material dielétrico passível de ser carregado de maneira não uniforme. As áreas carregadas positivamente atraem os elétrons, permitindo que passem pela máscara. Onde há carga negativa, os elétrons são repelidos, e não atingem o fósforo.

Um canhão de elétrons é responsável pela emissão de um feixe fino e intenso de elétrons que produz a carga diferenciada na máscara. Dessa forma são criados “buracos” na máscara que deixarão os elétrons do outro canhão passarem, definindo a forma da imagem a ser gerada. O desenho gravado na máscara sustenta a imagem na tela até que uma voltagem positiva seja enviada para a máscara, apagando a imagem.

Vantagens dos sistemas DVST: alta resolução (pelo menos 1024x781); ausência de *flickering*. Desvantagens: não há como apagar seletivamente a imagem. O apagamento da tela é sempre global, resultante do carregamento uniforme da máscara; o apagamento causa um efeito de *flashing* na tela; a tecnologia não é adequada para monitores coloridos (usa apenas fósforo verde); a imagem perde definição com o tempo; os tubos desgastam e precisam ser substituídos.

A tecnologia dos DVST está ultrapassada, tendo sido substituída pelos dispositivos de varredura matriciais.

## 2.2 Primitivas de Software para Dispositivos Vetoriais

Uma primitiva de software é uma rotina ou um comando escrito em alguma linguagem que executa uma função básica de um aplicativo: ou seja, uma primitiva de software implementa uma única função. Um sistema de software complexo é construído através de chamadas hierárquicas para suas funções primitivas.

Praticamente todos os dispositivos de saída vetoriais trabalham com duas primitivas de software básicas: *moveto(dcx,dcy)* e *drawto(dcx,dcy)*. A primeira move a posição corrente (CP - *current position*) do feixe (ou caneta, no caso de um *plotter*) para o ponto definido pelas coordenadas do dispositivo especificadas, (*dcx, dcy*), que passa a definir a nova CP. A segunda desenha um linha reta (ou, em outras palavras, um vetor) entre o ponto definido pela CP e o ponto definido pelas coordenadas (*dcx, dcy*), que passa a ser a nova CP.

Num sistema gráfico com vários dispositivos de saída conectados ao computador hospedeiro, cada qual com seu próprio sistema de coordenadas, é conveniente que o sistema disponha de primitivas genéricas que sejam acessadas de maneira independente do dispositivo - através de NDCs ou de coordenadas físicas. As rotinas genéricas *move\_to* e *draw\_to*, nesse caso, contém estruturas “case” com comandos alternativos para cada dispositivo admitido, e enviam a saída para o dispositivo de saída correntemente ativado, depois de fazer as conversões apropriadas [Rankin, p. 29].

## 2.3 Dispositivos Gráficos Matriciais

### 2.3.1 Impressoras

Impressoras matriciais: nessas impressoras, os caracteres são impressos por intermédio de um conjunto de agulhas - um ponto é impresso quando uma agulha pressiona a fita sobre o papel. As agulhas são montadas sobre um cabeçote móvel, e os diferentes caracteres são obtidos pelo acionamento conveniente das agulhas a medida que o cabeçote se movimenta.

Estas impressoras podem operar no modo texto para a impressão de caracteres, ou num modo gráfico (“*bit image*”) pelo qual é possível controlar cada agulha de modo independente. Em outras palavras, o conjunto de padrões (caracteres) que podem ser impressos é programável. Cada padrão a ser impresso é definido por uma pequena matriz de pontos, onde cada ponto pode ser traçado ou não.

Impressoras gráficas: Diversas técnicas têm sido utilizadas na construção de impressoras gráficas, sendo que as mais comuns são as tecnologias de jato de tinta e laser.

Nas impressoras a jato de tinta, o cabeçote transporta um pequeno bico que expele a tinta num jato curto e fino sobre o papel. É possível variar a intensidade do jato, obtendo-se assim controle sobre a

densidade de impressão, e algumas impressoras dispõem de vários bicos com tintas de cores diferentes, cuja mistura produz diversas tonalidades.

Nas impressoras a laser, o processo de impressão é semelhante ao das copiadoras eletrostáticas. Um feixe de raio laser varre uma chapa numa trajetória semelhante ao de um cabeçote de uma impressora. O bombardeio do feixe deixa a chapa carregada com uma carga eletrostática não uniforme. Por efeito da intensidade da carga, uma tintura (*tonner*) adere à chapa e por pressão é impregnada no papel, formando a imagem.

Apesar de utilizarem tecnologias distintas, estas impressoras produzem imagens de natureza semelhante (mas com qualidade superior) às geradas pelas impressoras matriciais em modo gráfico. As imagens são, em última instância, constituídas de minúsculos pontos regularmente espaçados.

### 2.3.2 Dispositivos de Vídeo de Varredura (Raster Scanning VDUs)

A tecnologia utilizada atualmente na grande maioria dos terminais de vídeo gráficos é a mesma dos aparelhos de TV. Um terminal gráfico simples requer (Figura 2.3):

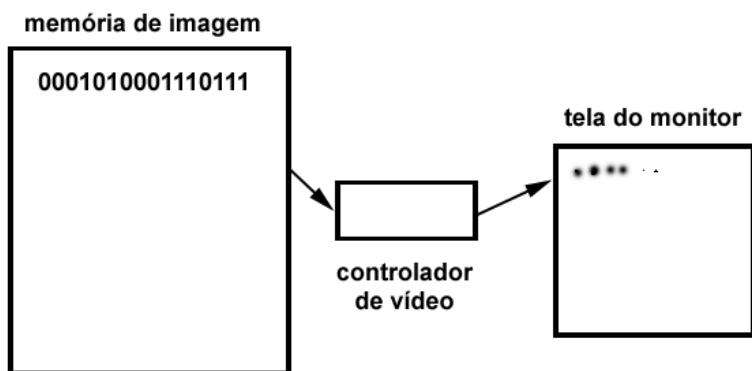


Figura 2.3: Uma seqüência de bits na memória de imagem é convertida para uma seqüência de pixels na tela.

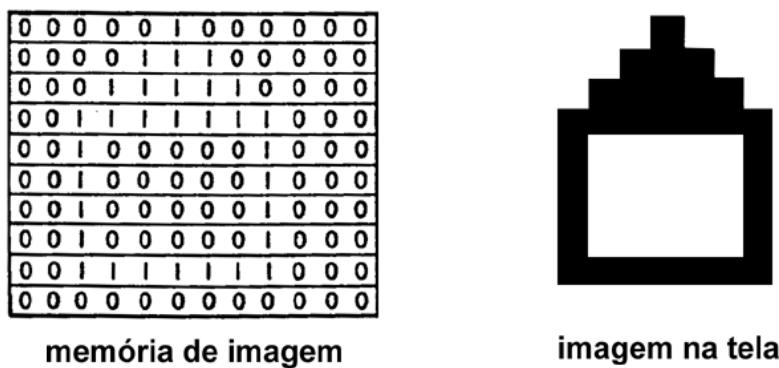


Figura 2.4: Representação esquemática de uma imagem matricial e sua representação num *frame buffer*.

1. uma memória digital (*frame buffer*, ou “memória de imagem”), na qual a imagem a ser visualizada é armazenada como uma matriz de pixels (cada posição na matriz contém a intensidade associada ao pixel correspondente na tela) (Figura 2.4). Os dados da imagem são colocados no *frame buffer* pelo computador hospedeiro.
  2. o monitor
  3. um controlador de vídeo (*display controller*), que consiste de uma interface que transfere o conteúdo do frame buffer para o monitor. Os dados devem ser transferidos repetidamente, pelo menos 15 vezes por segundo, de modo a manter uma imagem estável na tela, reduzindo o *flickering*. Note que processo de transferência implica numa conversão digital-analógica (DAC).

Para gerar a imagem, utiliza-se a técnica conhecida como *raster scanning* (“varredura”, ou “rastreio”), que é a mesma utilizada na geração de imagens de TV. Essa técnica também utiliza um CRT, sendo que

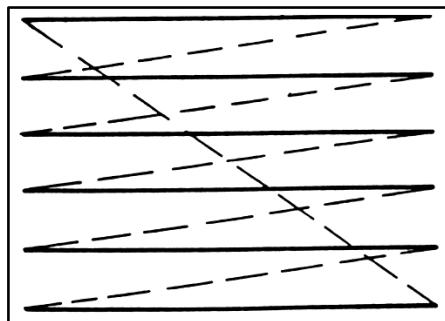


Figura 2.5: Varredura por rastreio fixo.

o feixe de elétrons varre a tela muitas vezes por segundo, de acordo com uma trajetória fixa (Figura 2.5). O feixe movimenta-se da esquerda para a direita, na horizontal. Ao final de uma varredura horizontal, o feixe (com intensidade anulada) é reposicionado no início da linha imediatamente abaixo, para nova varredura.

Para garantir o “refrescamento” da imagem, o feixe é redirecionado ao ponto inicial da primeira linha sempre que atingir o final da tela. Note que a intensidade do feixe num determinado pixel é determinada pelo valor associado ao pixel no *frame buffer*. Se a memória consiste de apenas um bit por pixel, então o pixel pode ter apenas dois estados: *on* ou *off*. Se existem oito bits por pixel, então cada pixel pode variar entre 256 níveis de intensidade, desde completamente *off* (preto), passando por diferentes níveis de cinza, podendo chegar a completamente *on* (branco).

A drástica redução dos custos de memória de semicondutor, e o surgimento de circuitos integrados dedicados ao controle e à geração de sinais para terminais de rastreio fixo tornaram esta tecnologia extremamente competitiva. Na tecnologia de varredura, todos os pontos que compõem uma imagem precisam ser armazenados, e não apenas os pontos extremos dos segmentos de reta. Consequentemente, gráficos de varredura requerem muito mais memória. Como o preço da memória digital está sendo reduzido à metade a cada ano que passa, a tecnologia tornou-se a alternativa mais acessível, utilizada tanto em microcomputadores como em estações de trabalho gráficas. Além do custo, outras vantagens são a capacidade de representar áreas cheias, e não somente linhas, e a possibilidade de utilização de cores ou diferentes níveis de cinza.

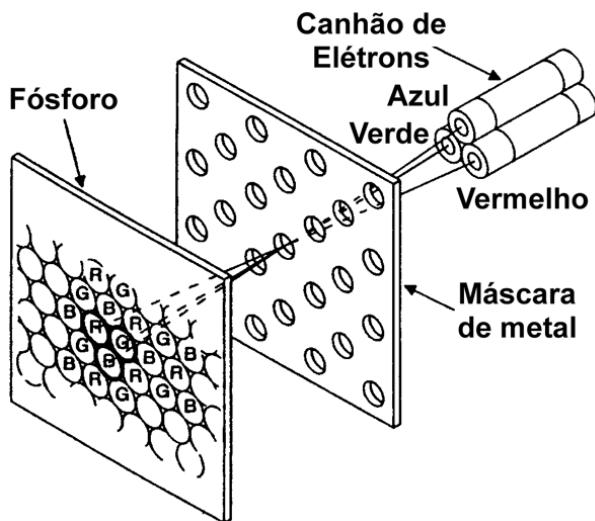


Figura 2.6: Representação esquemática de um CRT por varredura colorido.

Em monitores monocromáticos, toda a superfície da tela é revestida com o mesmo tipo de fósforo, e o feixe de elétrons pode ser direcionado para qualquer ponto da tela através das voltagens  $x$  e  $y$ . Num monitor colorido, cada pixel da tela é recoberto por três tipos de fósforo, que produzem as cores vermelho, verde e azul (*Red*, *Green*, *Blue* - RGB). Ao invés de um único feixe de elétrons existem três, cada qual associado a uma cor de fósforo. Entre a superfície da tela recoberta de fósforo, e os feixes de elétrons, está uma barreira de metal, denominada *shadow mask* ou *metal mask* (Figura 2.6), que, por meio de buracos em posições estratégicas, garante que cada feixe atinge apenas o fósforo ao qual está associado.

R	G	B	Valor Binário	Cor
0	0	0	0	preto
0	0	1	1	azul
0	1	0	2	verde
0	1	1	3	turquesa
1	0	0	4	vermelho
1	0	1	5	magenta
1	1	0	6	amarelo
1	1	1	7	branco

Tabela 2.1: Cores RGB em três bits.

Variando a intensidade de cada feixe, varia-se a intensidade do brilho de cada fósforo, obtendo-se cores diferentes.

Assim como nos monitores monocromáticos o feixe de elétrons está associado a um conjunto de bits na *frame memory* que determina a intensidade dos fósforos vermelho, verde e azul. Se existe apenas um bit associado a cada feixe (ou seja, três bits de memória por pixel), pode-se obter oito cores distintas, conforme mostrado na Tabela 2.1. O número de bits associado a cada pixel é denominado *pixel depth* (bit planes), ou “profundidade” do pixel.

Se cada feixe tem profundidade  $p$ , o pixel tem profundidade  $3p$ , e pode-se gerar  $2^{3p}$  cores. Alguns terminais gráficos têm pixels com profundidade quatro, sendo que os três primeiros bits correspondem aos valores R, G e B, e o quarto especifica o brilho, ou intensidade total da cor mostrada. Isto resulta em  $2^4 = 16$  cores possíveis para cada pixel. Outros sistemas usam valores diferentes de profundidade de pixel, com significados diferentes associados aos respectivos bit planes.

Se a profundidade do pixel é  $D$ , o número real de cores possíveis é  $2^D$ , mas o número total de cores que podem ser mostradas, denominado *palette range*, pode ser bem maior. Ao invés de usar o valor do pixel armazenado na memória para controlar o feixe diretamente (e.g., usá-lo como uma cor), ele é usado como um índice para uma tabela de cores (*color lookup table*, CLUT). Um *palette*, ou “cor lógica”, é um dos  $2^D$  valores de pixel possíveis, e portanto existem  $2^D$  palettes (0 a  $2^D - 1$ ). Um software pode atribuir valores associados a cores físicas às posições da *look-up table* correspondentes a um *palette*. Assim, o valor armazenado numa posição da tabela é usado para controlar a cor do pixel associado no monitor.

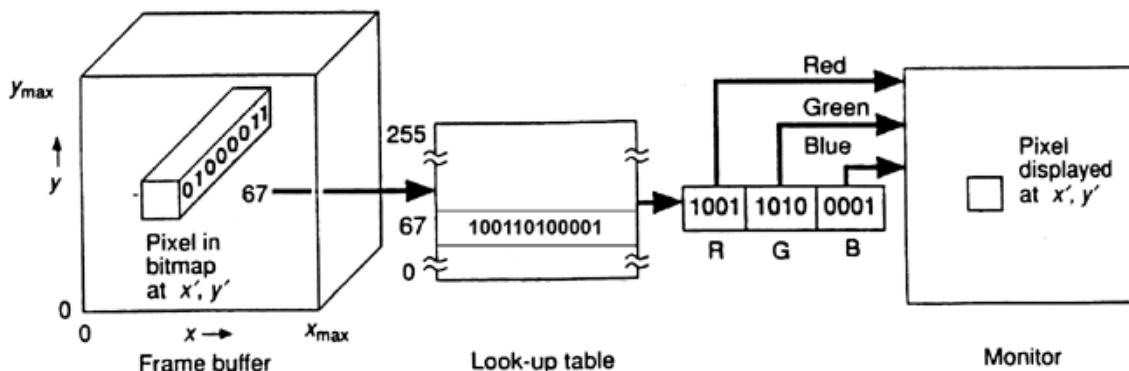


Figura 2.7: Organização de uma vídeo *look-up table*.

Em geral, a tabela de cores é uma área fixa de RAM com  $2^D$  posições de 24 bits cada: a cada feixe (ou cor) são associados 8 bits, e portanto cada feixe pode ter 256 níveis de intensidade. Isto resulta num total de 16 milhões de cores possíveis, sendo que apenas  $2^D$  podem ser mostradas simultaneamente na tela. A Figura 2.7 mostra um *frame buffer* com 8 bits por pixel, e uma LUT com 12 bits por posição.

### 2.3.3 Primitivas de Software

A primitiva básica, no caso de um dispositivo matricial, traça um ponto numa determinada posição do retângulo de visualização:

```
int write_pixel(int dcx, int dcy, int pixel_value)
```

Também é possível ler dados de um dispositivo, através da primitiva inversa:

```
int read_pixel(int dcx, int dcy, int *pixel_value)
```

Em ambos os casos, os dados são lidos ou escritos no *frame buffer*, ou memória de imagem, que armazena todos os pixels de uma imagem a ser visualizada ou lida. Além disso, o programador precisa acessar a *lookup table* - caso ela exista - o que é feito através de duas primitivas:

```
int write_CLUT(int pixel_value, int R, int G, int B)
read_CLUT(int pixel_value, int *R, int *G, int *B)
```

Todos os comandos gráficos num sistema envolvendo dispositivos matriciais podem ser construídos a partir dessas quatro primitivas.

Num sistema básico com um dispositivo de visualização monocromático, as primitivas podem ser reduzidas simplesmente a duas rotinas básicas, que são casos especiais da rotina “*write\_pixel*”. Estas rotinas “acendem” e “apagam” o ponto definido pelas coordenadas (*dcx*, *dcy*).

```
dot_on(int dcx, int dcy);
dot_off(int dcx, int dcy)
```

Apesar destas subrotinas usarem coordenadas de tela, rotinas num nível superior executariam as transformadas de visualização para converter coordenadas do usuário para coordenadas do dispositivo.

## 2.4 Exercícios

1. Escreva um programa (em C) que desenhe um polígono regular (ou seja, um polígono com lados de comprimentos iguais e ângulos internos iguais) com raio igual a um terço da altura da tela, e centro no centro da tela. O programa deve solicitar o número de vértices, *n*.
2. Modifique o programa do exercício anterior para solicitar também o raio *r* do círculo que circunscreve o polígono. Faça com que o seu programa estime o número *n* de vértices necessários para que o polígono pareça uma “boa aproximação” para um círculo, para diferentes raios. Usando os parâmetros *ndh*, *ndv*, *width*, *height* do dispositivo, obtenha uma relação teórica entre *n* e *r* para a aproximação de círculos através de polígonos.
3. Alguns dispositivos vetoriais oferecem um conjunto de três primitivas gráficas:

- **pen\_up**: levanta a caneta do papel, ou desliga o feixe de elétrons;
- **pen\_down**: coloca a caneta sobre o papel, ou liga o feixe;
- **locate(dcx,dcy)**: posiciona a CP num ponto do retângulo de visualização.

- Escreva rotinas de software que simulem estas três primitivas, usando as primitivas **moveto** e **drawto**.
4. Calcule as razões de aspecto (gráfica e física), e as resoluções de área horizontal e vertical de uma tela de TV colorida padrão, onde:

- width = 42cm;
- height = 31cm;
- ndh = 546;
- ndv = 434.

## 2.5 Dispositivos de Entrada

Juntamente com os dispositivos de saída, os dispositivos de entrada gráfica permitem o estabelecimento da comunicação usuário-máquina. É importante que programadores de sistemas e aplicativos gráficos compreendam a natureza dos principais dispositivos de entrada, de forma a selecioná-los adequadamente em função da natureza das informações a serem tratadas, e da maior ou menor facilidade que oferecem aos usuários para exprimir informações.

### **2.5.1 Teclado**

O teclado alfanumérico é o recurso mais utilizado na comunicação entre usuário e máquina, permitindo a entrada de textos e dados numéricos. Um teclado alfanumérico estendido inclui teclas adicionais para a seleção de alternativas em menus ou para o disparo de funções específicas. Teclas de movimento de cursor podem ser usadas para seleção de coordenadas de um ponto em uma tela gráfica.

Em geral, valores de coordenadas especificando posições de um cursor na tela são fornecidas para o computador por meio de dispositivos como a *lighth pen*, o *joystick*, o *mouse* e as mesas digitalizadoras (*tablets*).

### **2.5.2 Lighth Pen**

É um dispositivo que funciona associado a um monitor de vídeo, e que é capaz de detectar luz. A caneta tem em sua ponta uma célula fotoelétrica e um interruptor de pressão. Ao ser pressionado contra a tela, o interruptor habilita a célula a detectar o pulso de luz emitido pelos fósforos que recobrem a tela no ponto sendo apontado.

Quando a *lighth pen* detecta um pulso de luz num terminal de varredura, o conteúdo dos registradores *X* e *Y* do controlador de vídeo é armazenado, e o processamento é interrompido. Através dos valores armazenados, o software gráfico determina as coordenadas do pixel apontado pela caneta.

Quando detecta um pulso de luz num monitor vetorial, a caneta informa ao processador, a execução é interrompida, e o processador é capaz de detectar qual instrução de traçado estava em execução no momento que ocorreu a interrupção, e portanto qual segmento de reta foi apontado pelo operador. A caneta é uma tecnologia um tanto ultrapassada, por ser cansativa para o usuário e sujeita a falhas.

### **2.5.3 Joystick**

É uma alavanca que admite movimentos para frente e para trás, esquerda e direita. Dois potenciômetros são responsáveis pela detecção dos movimentos na horizontal e na vertical. Sua manipulação fornece um par de valores numéricos em uma faixa limitada fixa para cada posição da alavanca. É um dispositivo de baixo custo, bastante utilizado em microcomputadores. Entretanto, o joystick é um dispositivo de baixa resolução, e não é muito adequado para controlar a posição absoluta de um cursor na tela.

### **2.5.4 Mouse**

O *mouse* é uma pequena caixa que pode deslizar sobre uma superfície plana. Na parte inferior, uma pequena esfera incrustada no *mouse* rola livremente a medida que este se movimenta. Os movimentos de rotação da esfera relativos à caixa são transmitidos mecanicamente a dois potenciômetros que medem o deslocamento do *mouse* em relação a dois eixos ortogonais fixos. Como o sistema de eixos é fixo em relação à caixa, o dispositivo é capaz apenas de detectar movimentos relativos à sua própria posição atual. Ele pode ser levantado da superfície, movido e colocado de volta sobre a superfície sem que a sua posição corrente no sistema seja alterada.

### **2.5.5 Mesa Digitalizadora (Tablet)**

É o dispositivo mais adequado para entrada de dados geométricos. Consiste de uma superfície plana e um cursor que pode ser posicionado sobre a superfície. Por indução eletromagnética, a posição corrente do cursor, relativa a um referencial fixo à mesa, pode ser detectada e transmitida ao processador.

### **2.5.6 Data Glove**

É o dispositivo de entrada e saída pelas mãos utilizado em realidade virtual. Consiste de um conjunto de sensores acoplados a uma luva que detectam movimento e força dos dedos. Antenas de transmissão e recepção são responsáveis por fornecer posicionamento das mãos. Com isso, a posição das mãos pode indicar posição de objetos, que são então mapeados para tela ou para um dispositivo de *headtrack*.

### **2.5.7 Outros dispositivos**

Entrada por voz, som via MIDI.

## Capítulo 3

# Traçado de Curvas em Dispositivos Gráficos Matriciais

O traçado de primitivas gráficas elementares, como segmentos de reta ou arcos de circunferência, requer 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. Estes algoritmos recebem o nome de *Algoritmos de Conversão Matricial*, por converterem em representação matricial elementos gráficos expressos em uma representação vetorial.

Um elemento geométrico básico a ser traçado num terminal gráfico é o segmento de reta. Atualmente, é comum que rotinas de traçado de linhas estejam disponíveis em hardware (em chips controladores). O chip recebe as coordenadas dos pixels extremos, e calcula quais pixels intermediários da superfície de exibição devem ser traçados para gerar uma reta (aproximada) na tela.

Gráficos complexos requerem o traçado de uma grande quantidade de segmentos de reta, e a velocidade é importante - daí a opção de traçado por meio de hardware especializado, que libera o processador hospedeiro para outras atividades enquanto o chip controlador calcula os pixels a serem traçados.

Se a função de traçado precisa ser implementada em software, a rotina deve ser escrita em código de máquina otimizado para atingir o máximo de velocidade. Entretanto, é instrutivo estudar alguns algoritmos para traçado de linhas usando uma linguagem de alto nível.

Suponhamos, que a superfície de exibição possua igual densidade de pixels na horizontal e na vertical (razão de aspecto gráfica igual a 1).

A seguir será apresentada uma importante propriedade do espaço onde os diversos elementos gráficos serão traçados.

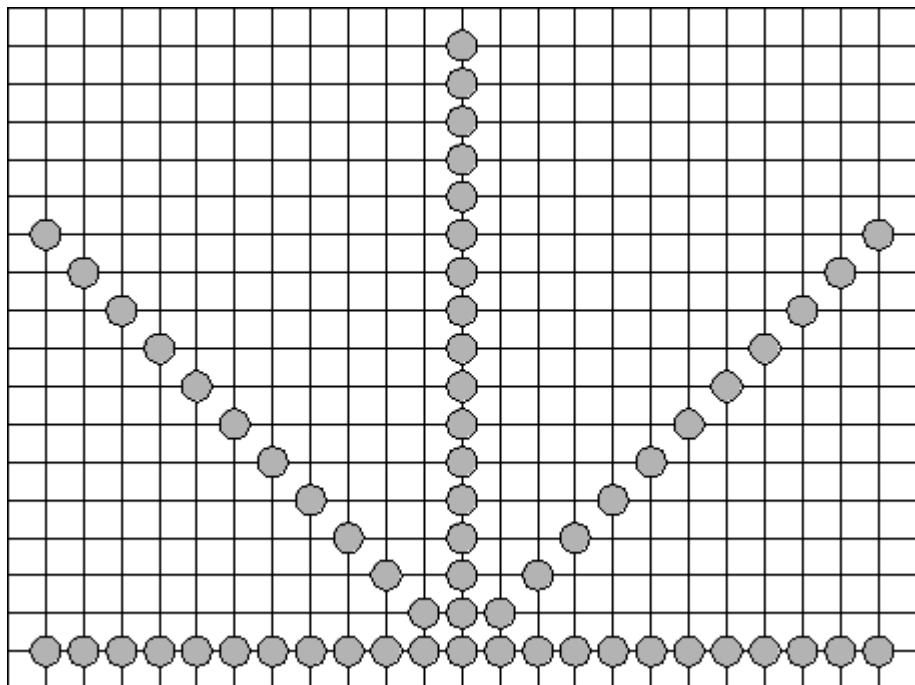


Figura 3.1: Representação de segmentos de reta horizontais, verticais e diagonais.

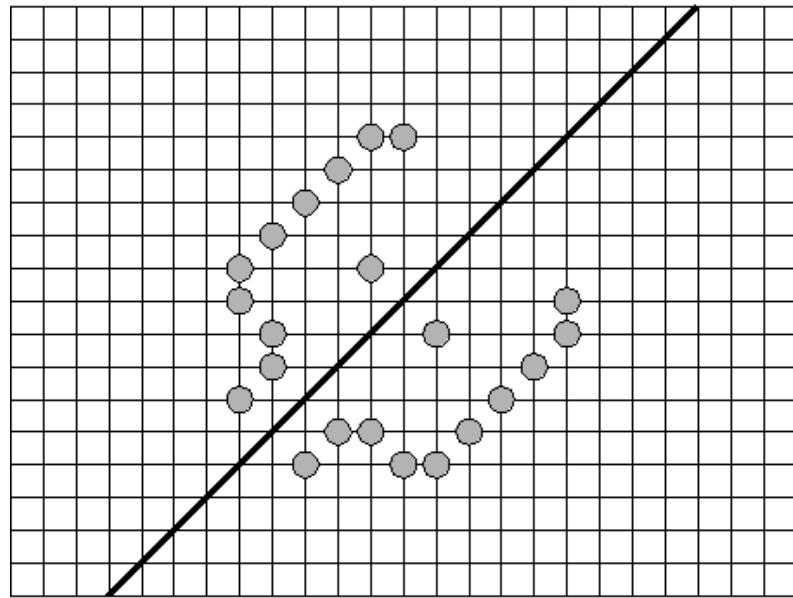


Figura 3.2: Reflexão de uma imagem com relação a um eixo diagonal.

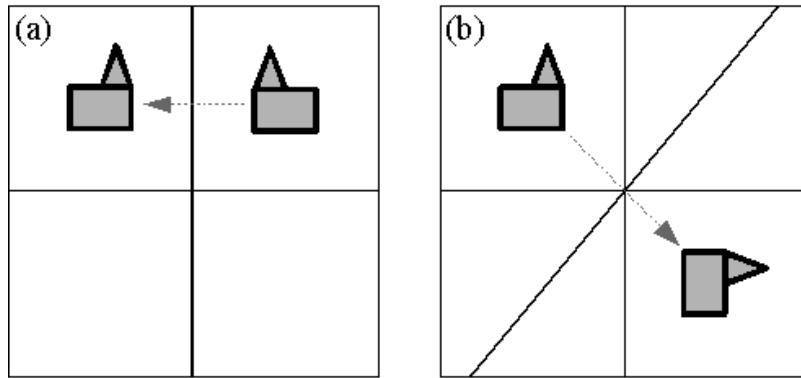


Figura 3.3: Rotação de  $90^\circ$  obtida através de 2 reflexões, uma horizontal (a) e outra na diagonal (b).

### 3.1 Simetria e Reflexão

Tomemos como exemplo o traçado de um segmento de reta. A representação matricial de segmentos de reta horizontais, verticais, e diagonais a  $45$  e  $135$  não apresenta nenhuma dificuldade (ver Figura 3.1). Pode-se dizer que estas direções formam na realidade eixos de simetria no espaço matricial. Logo, qualquer imagem representada no espaço matricial pode ser refletida em relação a qualquer reta horizontal, vertical ou diagonal sem apresentar qualquer deformação (ver Figura 3.2).

A **Simetria** será explorada na conversão matricial de primitivas gráficas que apresentem simetria em relação a qualquer um destes eixos. Na Figura 3.3 pode-se ver como efetuar a rotação de uma imagem utilizando de 2 reflexões.

### 3.2 Conversão Matricial de Segmentos de Reta

Alguns critérios de conversão matricial podem ser vistos através da Figura 3.4.

Em (a) adotou-se o critério de selecionar-se os dois pixels imediatamente acima e abaixo do ponto de intersecção do segmento com cada vertical, a menos quando o segmento passa por um pixel. Restrição: dessa forma obtém-se linhas densas, como se o segmento fosse espesso.

Em (b) o critério foi selecionar todos os pixels cujas coordenadas são obtidas arredondando-se os valores das coordenadas de algum ponto do segmento. Restrição: com segmentos a  $45$  o critério produz resultados semelhantes ao anterior.

A conversão ilustrada em (c) foi obtida selecionando-se em cada vertical o pixel mais próximo do ponto de intersecção do segmento com a reta vertical. Vantagens: aparência leve e continuidade.

Em (d) utilizou-se o mesmo critério, só que para as linhas horizontais. Restrição: descontinuidade.

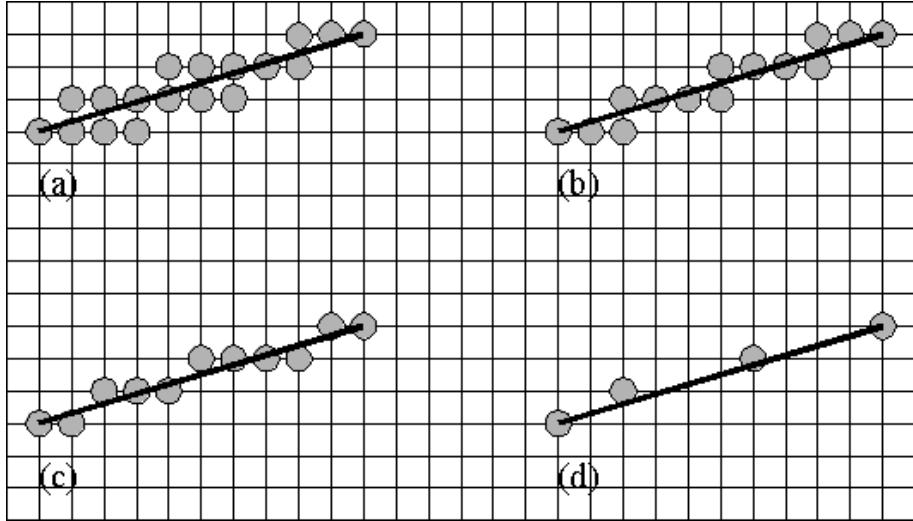


Figura 3.4: Conversões matriciais de um segmento de reta resultante de diferentes critérios.

### 3.2.1 Características Desejáveis para os Algoritmos de conversão Matricial de Segmentos de Retas

1. **Linearidade:** os pixels traçados devem dar a aparência de que estão sobre uma reta. Isto é trivial no caso de segmentos paralelos aos eixos x ou y, ou com inclinação de 45 graus, mas não nos outros casos.
2. **Precisão:** os segmentos devem iniciar e terminar nos pontos especificados. Caso isso não ocorra, pequenos gaps podem surgir entre o final de um segmento e o início de outro.
3. **Espessura (Densidade) uniforme:** a densidade da linha é dada pelo número de pixels traçados dividido pelo comprimento da linha. Para manter a densidade constante, os pixels devem ser igualmente espaçados. A imagem do segmento não varia de intensidade ou espessura ao longo de sua extensão.
4. **Intensidade independente da inclinação:** para segmentos de diferentes inclinações
5. **Continuidade:** a imagem não apresenta interrupções indesejáveis.
6. **Rapidez no Traçado dos segmentos.**

Os algoritmos que vamos estudar usam o método **incremental**, que consiste em executar os cálculos iterativamente, mantendo um registro (ou memória) do progresso da computação a cada passo da iteração. Nesse método, começando por um dos extremos da linha, o próximo ponto é calculado e traçado até que o outro extremo seja atingido e traçado.

O problema da conversão matricial consiste essencialmente em ajustar uma curva, no caso, um segmento de reta, a qual é precisamente definida por coordenadas reais, a uma malha de coordenadas inteiras. Isto pode ser feito calculando as coordenadas reais,  $(x_r, y_r)$ , do próximo ponto na linha, e escolhendo na malha os pixels cujas coordenadas  $(x, y)$  mais se aproximam das coordenadas reais, obtidos por arredondamento ou truncamento.

Qualquer algoritmo de traçado de linhas pode ser testado com relação aos requisitos acima: Para o requisito 1, calcula-se a soma dos quadrados das diferenças entre cada par  $(x_r, y_r)$ ,  $(x, y)$ . O melhor algoritmo é o que resultar na menor diferença. O requisito 2 é problemático apenas quando os pontos extremos dos segmentos são especificados em coordenadas do usuário. Para o requisito 3, conta-se o número de pixels traçados, e divide-se pelo comprimento da linha. Os requisitos 4 e 6 são avaliados através da execução de *benchmarks* em cada algoritmo, para efeito de comparação. O requisito 5 pode ser verificado analisando-se os tamanho dos “buracos” entre um ponto e o seu próximo, o que pode ser feito através de um algoritmo que calcule a distância entre dois pontos.

### 3.2.2 Critério Adotado

Seja o segmento de reta definido por seus extremos  $(x_1, y_1)$  e  $(x_2, y_2)$ .

Pode-se supor que este segmento está no 1º octante (porquê?!). Então estas coordenadas respeitam as relações:

$$0 < x_1 < x_2 \quad (3.1)$$

$$0 < y_1 < y_2 \quad (3.2)$$

$$y_2 - y_1 < x_2 - x_1 \quad (3.3)$$

Nestas condições o segmento de reta corta um número maior de linhas verticais do reticulado do que de horizontais e assim:

*Critério de Conversão: em cada vertical do retilado com abscissa entre  $x_1$  e  $x_2$  apenas o pixel mais próximo da interseção do segmento com a vertical faz parte da sua imagem.*

Este critério aplicado a segmentos do 2ºoctante não produz resultados satisfatórios (tente fazê-lo!), pois nesse caso o número de verticais que corta o segmento será menor que o de horizontais. Solução: para segmentos no 2ºoctante utilizar este critério só que trocando  $x$  com  $y$  e vertical por horizontal. A Figura 3.5 apresenta a conversão de segmentos pelo critério acima.

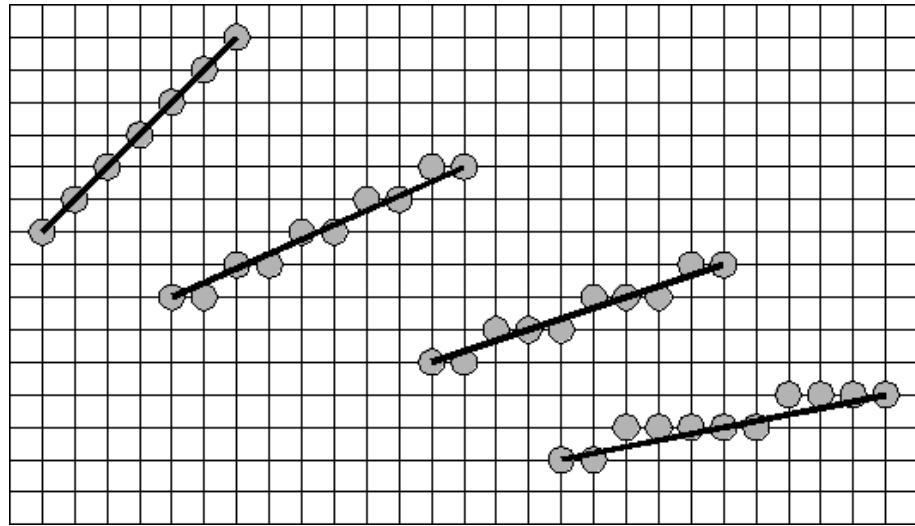


Figura 3.5: Imagens de segmentos de reta convertidos pelo critério explícito acima.

### 3.2.3 Algoritmos

Equação da reta:

$$y = y_1 + a(x - x_1) \quad (3.4)$$

onde  $a = \frac{(y_2 - y_1)}{(x_2 - x_1)}$ .

Dessa forma, basta que para cada vertical da grade entre  $x_1$  e  $x_2$ , determinemos o inteiro mais próximo da ordenada correspondente da reta. O algoritmo 3.1 implementa esta idéia.

### 3.2.4 Algoritmo do “Ponto-Médio”

Um algoritmo equivalente a este mas que dispensa as operações em ponto flutuante foi proposto por Bresenham (em 1965). Bresenham desenvolveu um algoritmo clássico que usa apenas variáveis inteiiras, e que permite que o cálculo de  $(x_i + 1, y_i + 1)$  seja feito incrementalmente, usando os cálculos já feitos para  $(x_i, y_i)$ . Vamos estudar uma versão do Algoritmo de Bresenham denominada **MidPoint Line Algorithm** (“Algoritmo do Ponto-Médio ou Meio-Ponto para Retas”) por Foley.

O algoritmo assume que a inclinação da linha está entre 0 e 1 (outras inclinações podem ser tratadas por simetria). O ponto  $(x_1, y_1)$  é o inferior esquerdo, e  $(x_2, y_2)$  é o superior direito.

Considere a linha na Figura 3.6. Assumindo que o pixel que acabou de ser selecionado é  $P$ , em  $(x_p, y_p)$ , e o próximo deve ser escolhido entre o pixel à direita (pixel E) e o pixel acima à direita (NE). Seja  $Q$  o ponto de intersecção entre a reta e a coluna  $x = x_p + 1$  da malha, e  $M$  o ponto intermediário entre os pixels E e NE. O que se faz é observar de que lado da reta está o ponto  $M$ . É fácil verificar que se  $M$  está acima da reta, o pixel E está mais próximo da reta; se  $M$  está abaixo, NE está mais próximo.

```

void line(int x1, int y1, int x2, int y2, int color){
    int x, y;
    float a;
    int valor;

    a=(y2-y1)/(x2-x1);
    for (x=x1;x<=x2;x++){
        /* arredonda y */
        y = (y1 + a * (x - x1));
        write_pixel(x, y, color);
    }/* end for */
}/*end line */

```

**Algoritmo 3.1:** Algoritmo simples para conversão matricial de retas. É pouco eficiente pois utiliza cálculos de ponto flutuante.

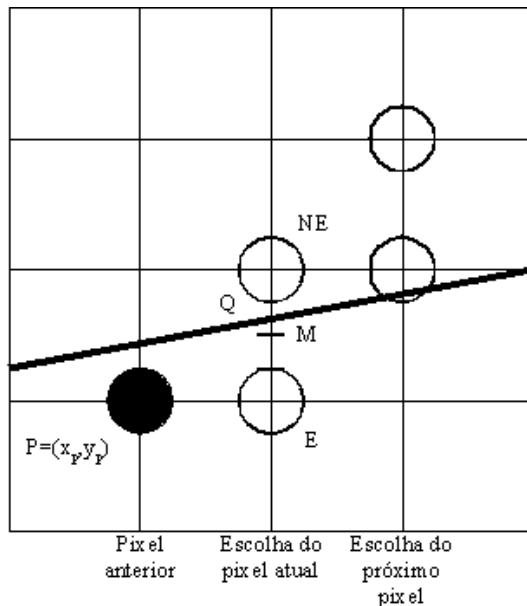


Figura 3.6: Grade de pixels para o Algoritmo o Ponto-Médio (M) e as escolhas E e NE.

Dessa forma, o teste do ponto-médio permite a escolha do pixel mais próximo da reta. Além disso, o erro (a distância vertical entre o pixel escolhido e a linha) é sempre  $\leq \frac{1}{2}$ .

O que precisamos agora é um método para calcular de que lado da reta está o ponto  $M$ . Para isso, considere sua função implícita,  $F(x, y) = ax + by + c = 0$ . Se  $dy = y_2 - y_1$ , e  $dx = x_2 - x_1$ , a equação da reta em termos de sua inclinação pode ser escrita como:

$$y = \frac{dy}{dx}x + B \quad (3.5)$$

e, consequentemente,

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0 \quad (3.6)$$

Aqui, tem-se  $a = dy$ ,  $b = -dx$ , e  $c = B \cdot dx$ .

É fácil verificar que  $F(x, y) = 0$  sobre a linha, positiva para pontos abaixo dela, e negativa para pontos acima dela. Para o teste do ponto-médio, basta calcular  $F(M) = F(x_p + 1, y_p + \frac{1}{2})$  e verificar o seu sinal. Como a decisão será tomada com base no valor da função no ponto  $(x_p + 1, y_p + \frac{1}{2})$ , definimos uma “variável de decisão”  $d = a(x_p + 1) + b(y_p + \frac{1}{2}) + c$ . Se  $d > 0$ , escolhemos o pixel NE, se  $d < 0$ , escolhemos o pixel E; se  $d = 0$  pode-se escolher qualquer um deles, por exemplo E.

É interessante verificar o que acontece com a localização de  $M$  e o valor de  $d$  no próximo ponto da reta. Ambos dependem, obviamente, da escolha de E ou NE. Se E for escolhido,  $M$  é incrementado de 1 na direção  $x$ . Assim,

$$d_{new} = F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c \quad (3.7)$$

mas

$$d_{old} = a(x_p + 1) + b(y_p + \frac{1}{2}) + c \quad (3.8)$$

Subtraindo  $d_{old}$  de  $d_{new}$  para obter a diferença incremental, tem-se  $d_{new} = d_{old} + a$ .

Denominamos  $\Delta E$  o incremento a ser adicionado depois da escolha de E; e  $\Delta E = a = dy$ . Em outras palavras, pode-se derivar o valor da variável de decisão no próximo passo incrementalmente, a partir do seu valor atual, sem necessidade de calcular  $F(M)$  diretamente.

Se NE é escolhido,  $M$  é incrementado de 1 em ambas as direções,  $x$  e  $y$ . Portanto,

Subtraindo  $d_{old}$  de  $d_{new}$ , tem-se  $d_{new} = d_{old} + a + b$ . Denominamos  $\Delta NE$  o incremento a ser adicionado depois da escolha de NE; e  $\Delta NE = a + b = dy - dx$ .

### Resumindo a técnica incremental do ponto-médio

A cada passo, o algoritmo escolhe entre 2 pixels, com base no sinal da variável de decisão calculada na iteração anterior; a seguir a variável de decisão é atualizada adicionando-se  $\Delta E$  ou  $\Delta NE$  ao valor anterior, dependendo do pixel escolhido.

Como o primeiro pixel corresponde ao ponto  $(x_1, y_1)$ , pode-se calcular diretamente o valor inicial de  $d$  para escolher entre E e NE. O primeiro ponto-médio está em  $(x_1 + 1, y_1 + \frac{1}{2})$ , e

$$F(x_1 + 1, y_1 + \frac{1}{2}) = a(x_1 + 1) + b(y_1 + \frac{1}{2}) + c = ax_1 + by_1 + c + a + \frac{b}{2} = F(x_1, y_1) + a + \frac{b}{2} \quad (3.9)$$

Como  $(x_1, y_1)$  é um ponto da reta,  $F(x_1, y_1) = 0$ ; consequentemente  $d_{start}$  é dado por  $a + \frac{b}{2} = dy - \frac{dx}{2}$ . Usando  $d_{start}$ , escolhe-se o segundo pixel, e assim por diante. Para eliminar a fração em  $d_{start}$ ,  $F$  é multiplicada por 2. Isto é:

$F(x, y) = 2(ax + by + c)$ . Isto multiplica cada constante e a variável de decisão por 2, mas não afeta o sinal da variável de decisão, que é o que interessa para o teste do ponto-médio.

A aritmética necessária para calcular  $d_{new}$  a cada passo é adição simples - nenhuma multiplicação é necessária (veja o algoritmo abaixo). Note que esta versão do algoritmo funciona apenas para linhas com inclinação entre 0 e 1, mas a generalização não é difícil. A Figura 3.7 mostra a reta que vai do ponto  $(5,8)$  ao ponto  $(9,11)$ , traçada por este algoritmo. Os valores sucessivos de  $d$  são 2, 0, 6 e 4, resultantes da escolha de NE, E, NE e NE, respectivamente.

### Exercícios

1. Escreva procedimentos para checar os 6 requisitos de um bom algoritmo de conversão matricial, e aplique-os aos algoritmos de conversão estudados.
2. Escreva uma versão melhorada do Algoritmo de Bresenham, que teste se as iterações devem ser feitas ao longo do eixo x ou ao longo do eixo y, e também para checar exceções como linhas paralelas ao eixo x.

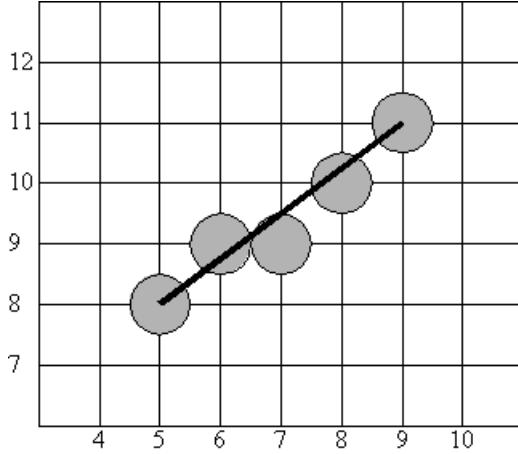


Figura 3.7: Grade de pixels para o Algoritmo o Ponto-Médio (M) e as escolhas E e NE.

### 3.3 Conversão Matricial de Circunferências

A equação de uma circunferência com centro na origem e raio  $R$ , em coordenadas cartesianas, é dada por:

$$x^2 + y^2 = R^2 \quad (3.10)$$

Circunferências não centradas na origem podem ser transladadas para a origem, e os pixels reais podem ser traçados aplicando-se um *offset* aos pixels gerados pelo algoritmo de conversão matricial.

Existem muitas abordagens simples, porém inefficientes, para o traçado de círculos: Em algoritmos não incrementais, um polígono regular de  $n$  lados é usado como aproximação para a circunferência. Para que a aproximação seja razoável, deve-se escolher um valor suficientemente alto para  $n$ . Entretanto, quanto maior o valor de  $n$ , mais lento será o algoritmo, e várias estratégias de aceleração precisam ser usadas. Em geral os algoritmos incrementais de conversão matricial são mais rápidos.

Outra abordagem seria usar a equação explícita da circunferência,  $y = f(x)$ :

$$y = \pm \sqrt{R^2 - x^2} \quad (3.11)$$

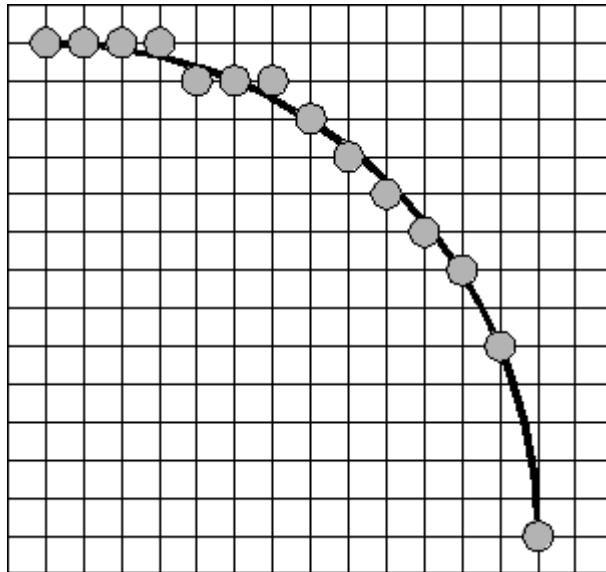


Figura 3.8: Um arco de  $\frac{1}{4}$  de circunferência, obtido variando-se  $x$  em incrementos unitários, e calculando e arredondando  $y$ .

Para desenhar  $\frac{1}{4}$  de circunferência (os outros  $\frac{3}{4}$  são desenhados por simetria), poderíamos variar  $x$  de 0 a  $R$ , em incrementos de uma unidade, calculando  $+y$  a cada passo através da equação acima. Essa estratégia funciona, mas é inefficiente porque requer operações de multiplicação e raiz quadrada. Além

```

void inc_line(int x1, int y1, int x2, int y2, int color){
    int dx, dy, incE, incNE, d, x, y;

    dx = x2 - x1;
    dy = y2 - y1;
    d = 2 * dy - dx; /* Valor inicial de d */
    incE = 2 * dy; /* Incremento de E */
    incNE = 2 * (dy - dx); /* Incremento de NE */
    x = x1;
    y = y1;
    write_pixel(x, y, color);
    while (x < x2){
        if (d <= 0){
            /* Escolhe E */
            d = d + incE;
            x = x + 1;
        }else{
            /* Escolhe NE */
            d = d + incNE;
            x = x + 1;
            y = y + 1;
        }/* end if */
        write_pixel(x, y, color);
    }/* end while */
}/* end inc_line */

```

**Algoritmo 3.2:** Algoritmo do Ponto-Médio incremental.

disso, haverá grandes *gaps* nas regiões onde a tangente à circunferência é infinita (valores de  $x$  próximos a  $R$ , Figura 3.8). Uma maneira de resolver o problema dos gaps é “plotar”  $x = R \cdot \cos \theta$  e  $y = R \cdot \sin \theta$ , para variando de 0 a 90 graus, mas essa solução também é ineficiente, pois precisa de funções caras ( $\sin$  e  $\cos$ ).

### 3.3.1 Simetria de ordem 8

Note que o traçado de uma circunferência pode tirar proveito de sua simetria. Considere uma circunferência centrada na origem. Se o ponto  $(x, y)$  pertence à circunferência, pode-se calcular de maneira trivial sete outros pontos da circunferência (Figura 3.9). Consequentemente, basta computar um arco de circunferência de 45° para obter a circunferência toda. Para uma circunferência com centro na origem, os oito pontos simétricos podem ser traçados usando o procedimento *CirclePoints* (Algoritmo 3.3).

```

void CirclePoints(int x, int y, int color){

    write_pixel( x, y, color);
    write_pixel( x, -y, color);
    write_pixel(-x, y, color);
    write_pixel(-x, -y, color);
    write_pixel( y, x, color);
    write_pixel( y, -x, color);
    write_pixel(-y, x, color);
    write_pixel(-y, -x, color);
}/* end CirclePoints */

```

**Algoritmo 3.3:** Procedimento CirclePoints.

Vamos estudar especificamente um algoritmo derivado do algoritmo de Bresenham para geração de circunferências, chamado “Midpoint Circle Algorithm” em [Foley et. al].

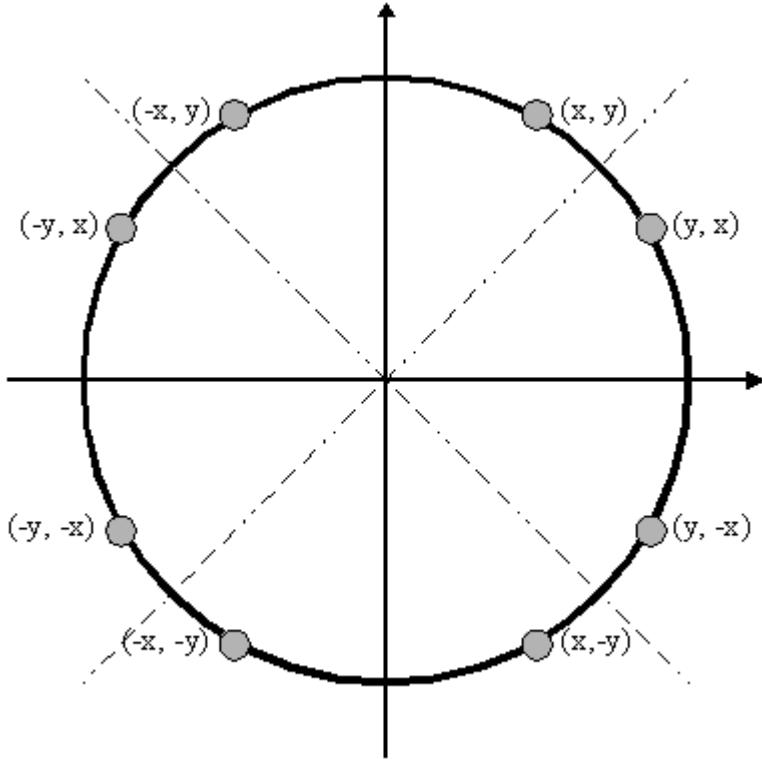


Figura 3.9: Oito pontos simétricos em uma circunferência.

### 3.3.2 Algoritmo do “Ponto-Médio” para Circunferências

Consideraremos apenas um arco de  $45^\circ$  da circunferência, o 2º octante, de  $x = 0$ ,  $y = R$  a  $x = y = \frac{R}{\sqrt{2}}$ , e usaremos o procedimento *CirclePoints* para traçar todos os pontos da circunferência. Assim como o algoritmo gerador de linhas, a estratégia é selecionar entre 2 pixels na malha aquele que está mais próximo da circunferência, avaliando-se uma função no ponto intermediário entre os dois pixels. No segundo octante, se o pixel  $P$  em  $(x_p, y_p)$  foi previamente escolhido como o mais próximo da circunferência, a escolha do próximo pixel será entre os pixels E e SE (Figure 3.10).

Seja a função  $F(x, y) = x^2 + y^2 - R^2$ , cujo valor é 0 sobre a circunferência, positivo fora dela e negativo dentro. Se o ponto intermediário (o “ponto-médio”) entre os pixels E e SE está fora da circunferência, o pixel SE é escolhido, porque está mais próximo dela. Por outro lado, se o pixel intermediário está dentro da circunferência, então o pixel E é escolhido.

Assim como no caso das linhas, a escolha é feita com base na variável de decisão  $d$ , que dá o valor da função no “ponto-médio”:

$$d_{old} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2 \quad (3.12)$$

Se  $d_{old} < 0$ , E é escolhido, e o próximo ponto-médio será incrementado de 1 na direção  $x$ . Assim,

$$d_{new} = F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2 \quad (3.13)$$

e  $d_{new} = d_{old} + (2x_p + 3)$ ; consequentemente  $\Delta E = 2x_p + 3$ .

Se  $d_{old} \geq 0$ , SE é escolhido, e o próximo ponto-médio será incrementado de 1 na direção  $x$  e decrementado de 1 na direção  $y$ . Portanto:

$$d_{new} = F(x_p + 2, y_p - \frac{3}{2}) = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2 \quad (3.14)$$

Como  $d_{new} = d_{old} + (2x_p - 2y_p + 5)$ ,  $\Delta SE = 2x_p - 2y_p + 5$ .

Note que no caso da reta (equação linear),  $\Delta E$  e  $\Delta NE$  eram constantes. No caso da circunferência (equação quadrática), E e SE variam a cada passo, sendo funções do valor específico de  $(x_p, y_p)$ , o pixel escolhido na iteração anterior (P é chamado “ponto de avaliação”). As funções podem ser avaliadas diretamente, a cada passo, dados os valores de  $x$  e  $y$  do pixel escolhido na iteração anterior. Essa avaliação não é computacionalmente cara, uma vez que as funções são lineares.

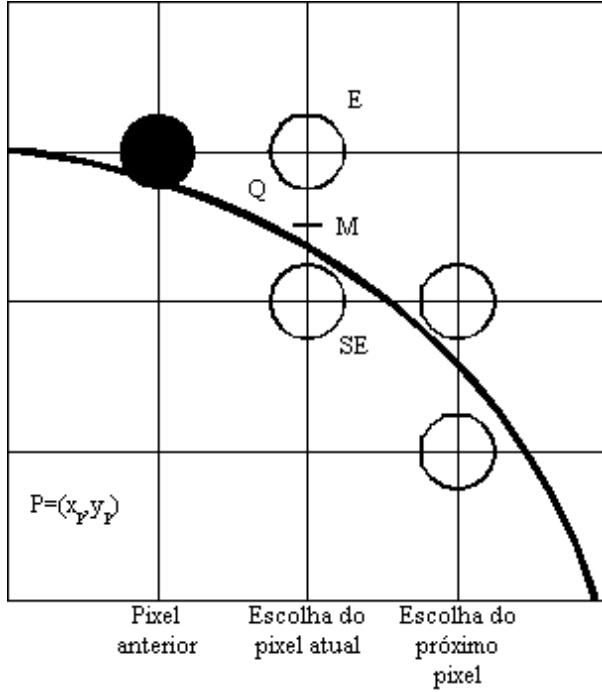


Figura 3.10: Malha de pixels para o Algoritmo do Ponto-Médio para circunferências, ilustrando a escolha entre os pixels E e SE.

**Resumindo**, os mesmos dois passos executados para o algoritmo do traçado de linhas são executados para o algoritmo de circunferências:

1. escolher o pixel com base no sinal da variável  $d$ , calculada na iteração anterior;
2. atualizar a variável  $d$  com o valor correspondente ao pixel escolhido. A diferença é que, na atualização de  $d$ , calculamos uma função linear do ponto de avaliação.

Falta ainda calcular a condição inicial (o 1º valor de  $d$ ). Limitando a utilização do algoritmo a raios inteiros no segundo octante, o pixel inicial é dado por  $(0, R)$ . O próximo “ponto-médio” está em  $(1, R - \frac{1}{2})$ , e portanto  $F(1, R - \frac{1}{2}) = 1 + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R$ . O algoritmo, bastante parecido com o algoritmo para traçado de retas, é mostrado no Algoritmo 3.4.

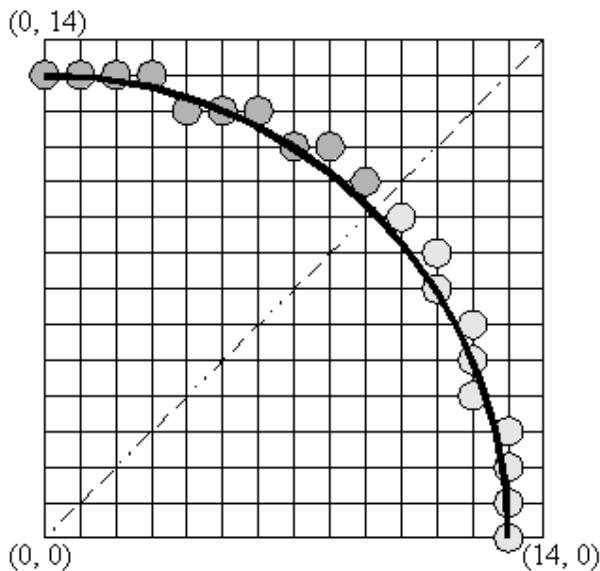


Figura 3.11: Segundo octante da circunferência gerado com o algoritmo do Ponto-Médio e primeiro octante gerado por simetria.

```

void MidPointCircle(int r, int color){
    int x, int y;
    float d;

    /* Valores iniciais */
    x = 0;
    y = r;
    d = 5/4 - r;

    CirclePoints(x, y, color);
    while (y > x){
        if (d < 0){
            /* Selecione E */
            d = d + 2 * x + 3;
            x++;
        }else{
            /* Selecione SE */
            d = d + 2 * (x - y) + 5;
            x++;
            y--;
        }/*end if*/
        CirclePoints(x, y, color);
    }/* end while */

}/* end MidpointCircle */

```

**Algoritmo 3.4:** Algoritmo do Ponto-Médio para circunferências (Aritmética Real).

O algoritmo pode ser facilmente modificado para manipular circunferências centradas fora da origem, ou de raios não inteiros. Entretanto, um problema com esta versão é a necessidade de usar aritmética real, devido ao valor inicial de  $d$ .

Para eliminar as frações, vamos definir uma nova variável de decisão,  $h$ , onde  $h = d - \frac{1}{4}$ , e substituir  $d$  por  $h + \frac{1}{4}$  no código. Agora, a inicialização é  $h = 1 - R$ , e a comparação  $d < 0$  torna-se  $h < -\frac{1}{4}$ . Entretanto, como o valor inicial de  $h$  é inteiro, e a variável é incrementada de valores inteiros ( $\Delta E$  e  $\Delta SE$ ), a comparação pode ser alterada simplesmente para  $h < 0$ . Temos agora um algoritmo que opera em termos de  $h$ , usando apenas aritmética inteira, que é mostrado no Algoritmo 3.5. (Para manter a notação consistente com o algoritmo de linhas, substituímos novamente  $h$  por  $d$ ). A Figura 3.11 mostra o segundo octante de uma circunferência de raio 14 gerada pelo algoritmo.

### Diferenças de Segunda Ordem

É possível melhorar o desempenho do algoritmo usando ainda mais extensivamente a técnica da computação incremental. Vimos que as funções são equações lineares, e no algoritmo, elas estão sendo computadas diretamente. Entretanto, qualquer polinômio pode ser calculado incrementalmente - assim como foi feito com as variáveis de decisão. (Na verdade, estamos calculando diferenças parciais de 1<sup>a</sup> e 2<sup>a</sup> ordens). A estratégia consiste em avaliar as funções diretamente em dois pontos adjacentes, calcular a diferença (que no caso de polinômios é sempre um polinômio de menor grau), e aplicar esta diferença a cada iteração.

Se escolhemos **E** na iteração atual, o ponto de avaliação move de  $(x_p, y_p)$  para  $(x_p + 1, y_p)$ . Como vimos, a diferença de 1<sup>a</sup> ordem é  $\Delta E_{old}$  em  $(x_p, y_p) = 2x_p + 3$ . Conseqüentemente,

$$\Delta E_{new} \text{ em } (x_p + 1, y_p) = 2(x_p + 1) + 3 \quad (3.15)$$

e a diferença de 2<sup>a</sup> ordem é  $\Delta E_{new} - \Delta E_{old} = 2$ .

Analogamente, **SE**  $_{old}$  em  $(x_p, y_p) = 2x_p - 2y_p + 5$ . Conseqüentemente,

$$\Delta SE_{new} \text{ em } (x_p + 1, y_p) = 2(x_p + 1) - 2y_p + 5 \quad (3.16)$$

e a diferença de 2<sup>a</sup> ordem é  $\Delta SE_{new} - \Delta SE_{old} = 2$ .

Se escolhemos **SE** na iteração atual, o ponto de avaliação move de  $(x_p, y_p)$  para  $(x_p + 1, y_p - 1)$ . Conseqüentemente

```

void MidPointCircleInt(int r, int color){
    int x, int y, d;

    /* Valores iniciais */
    x = 0;
    y = r;
    d = 1 - r;

    CirclePoints(x, y, color);
    while (y > x){
        if (d < 0){
            /* Seleccione E */
            d = d + 2 * x + 3;
            x++;
        }else{
            /* Seleccione SE */
            d = d + 2 * (x - y) + 5;
            x++;
            y--;
        }/*end if*/
        CirclePoints(x, y, color);
    }/* end while */

}/* end MidpointCircleInt */

```

**Algoritmo 3.5:** Algoritmo do Ponto-Médio para circunferências utilizando aritmética inteira.

$$\Delta E_{new} \text{ em}(x_p + 1, y_p - 1) = 2(x_p + 1) + 3 \quad (3.17)$$

e a diferença de 2<sup>a</sup> ordem é  $\Delta E_{new} - \Delta E_{old} = 2$ . Além disso,

$$\Delta SE_{new} \text{ em}(x_p + 1, y_p - 1) = 2(x_p + 1) - 2(y_p - 1) + 5 \quad (3.18)$$

e a diferença de 2<sup>a</sup> ordem é  $\Delta SE_{new} - \Delta SE_{old} = 4$ .

Considerando que  $\Delta E$  e  $\Delta SE$  são computados usando o pixel inicial  $(0, R)$ , a versão final do algoritmo dada a seguir e consiste dos seguintes passos:

1. escolher o pixel com base no sinal da variável  $d$  calculada na iteração anterior;
2. atualizar a variável  $d$  usando E ou SE, usando o valor de calculado na iteração anterior;
3. atualizar os  $\Delta$ s para considerar o movimento para o próximo pixel, utilizando as diferenças constantes previamente calculadas; e
4. mover para o próximo pixel.  $\Delta E$  e  $\Delta SE$  são computados usando o pixel inicial  $(0, R)$ .

O algoritmo, bastante utilizando a otimização das diferenças de 2<sup>a</sup>ordem é apresentado em no Algoritmo 3.6.

### 3.4 Conversão Matricial de Elipses

Consideremos a elipse padrão da Figura 3.12, centrada em  $(0, 0)$ . Ele é descrita pela equação:

$$F(x, y) = b^2 x^2 + a^2 y^2 - a^2 b^2 = 0 \quad (3.19)$$

onde  $2a$  é o comprimento do eixo maior (eixo  $x$ ) e  $2b$  é o comprimento do eixo menor (eixo  $y$ ).

Como foi feito para circunferências, tomar-se-á como base elipses centradas na origem (translação!).

As elipses possuem simetria horizontal e vertical, assim a preocupação será de traçar apenas o primeiro quadrante da elipse e depois traçar os demais por reflexão.

Primeiramente dividiremos o primeiro quadrante em duas regiões: o limite entre as duas regiões é o ponto da curva cuja tangente tem inclinação igual a  $-1$ . A determinação deste ponto não é tão simples.

```

void MidPointCircleInt(int r, int color){
    // Função de utiliza diferenças parciais de 2a ordem para incrementar variavel de
    // decisão d. Circunferência centrada na origem

    /* Valores iniciais */
    int x = 0;
    int y = r;
    int d = 1 - r;
    int deltaE=3
    int deltaSE= -2*r+5

    CirclePoints(x, y, color);
    while (y > x){
        if (d < 0){ /* Selecione E */
            d +=deltaE;
            deltaE += 2;
            deltaSE += 2;
        }else{ /* Selecione SE */
            d +=deltaSE;
            deltaE += 2;
            deltaSE += 4;
            y--;
        }/*end if*/
        x++;
        CirclePoints(x, y, color);
    }/* end while */

}/* end MidpointCircleInt */

```

**Algoritmo 3.6:** Algoritmo MidPoint para conversão matricial de circunferências utilizando diferenças de 2<sup>a</sup>ordem

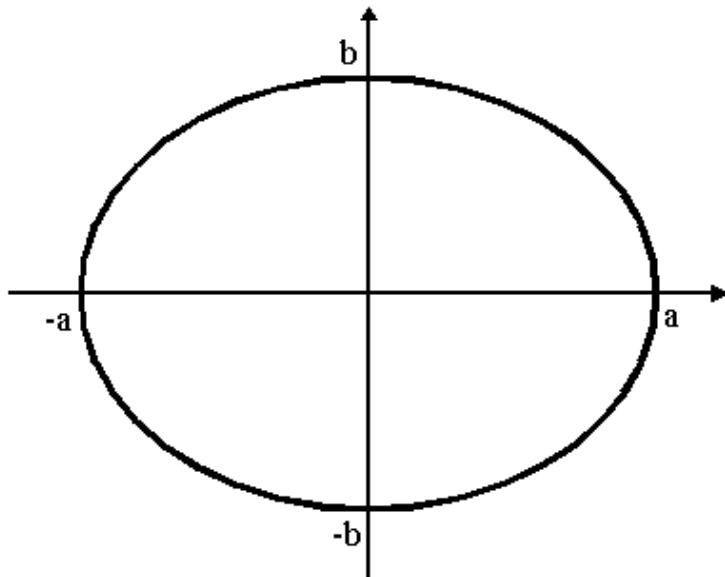


Figura 3.12: Elipse padrão centrada na origem.

Para tanto utilizaremos o vetor gradiente que é perpendicular à tangente à curva no ponto  $P$ , definido como:

$$\text{Gradiente } F(x, y) = \frac{\partial F}{\partial x} \vec{i} + \frac{\partial F}{\partial y} \vec{j} = 2b^2 x \vec{i} + 2a^2 y \vec{j} \quad (3.20)$$

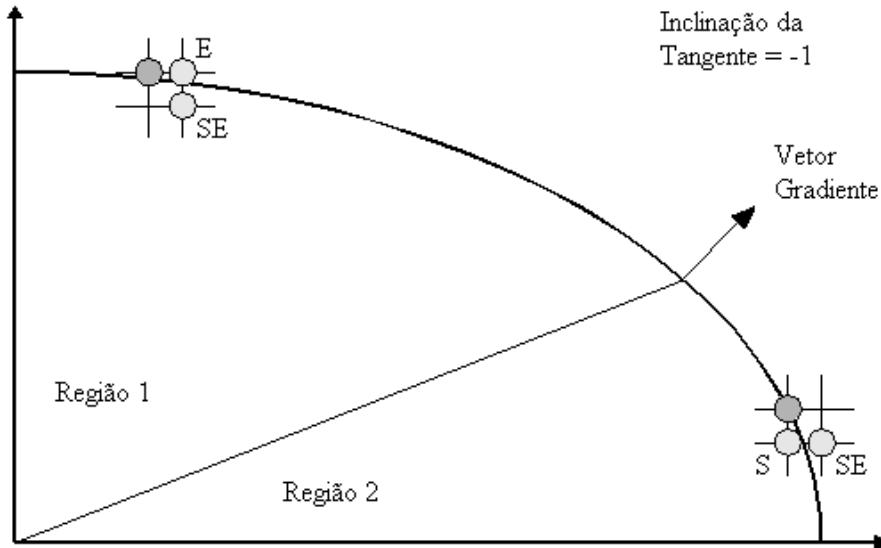


Figura 3.13: As duas regiões adotadas, definidas pela tangente a  $45^\circ$ .

O limite entre as duas regiões é o ponto cuja inclinação da curva é  $-1$ , e este ponto ocorre quando o vetor gradiente tem inclinação igual a  $1$ , isto é, quando os componentes nas direções  $\vec{i}$  e  $\vec{j}$  são de magnitudes iguais. A componente  $\vec{j}$  do gradiente é maior do que a  $\vec{i}$  na região 1, e vice-versa na região 2. Assim, se o próximo “ponto-médio” é  $a^2(y_p - \frac{1}{2}) \leq b^2(x_p + 1)$ , nós mudamos da região 1 para a região 2 (ver Figura 3.13).

Como nos outros algoritmos de “ponto-médio” anteriores, o sinal da variável de decisão  $d$  (é o valor da função no “ponto-médio”) será usado para verificar que pixels fazem ou não parte da elipse.

Assim, na **região 1**, se o pixel corrente está em  $(x_p, y_p)$ , então a variável de decisão para a região 1,  $d1$ , é  $F(x_p + 1, y_p - \frac{1}{2})$ , o ponto médio entre E e SE. Repetindo o processo de derivação para a função de incremento (como foi feito para circunferência), tem-se que  $\Delta E = b^2(2x_p + 3)$  e  $\Delta SE = b^2(2x_p + 3) + a^2(-2y_p - 1)$ .

Para a **região 2**, se o pixel corrente está em  $(x_p, y_p)$ , então a variável de decisão para a região 2,  $d2$ , é  $F(x_p + \frac{1}{2}, y_p - 1)$ , o ponto médio entre S e SE. Pode-se fazer cálculos idênticos aos da região 1.

A condição inicial deve ser então computada. Assumindo valores inteiros para  $a$  e  $b$ , a elipse começa em  $(0, b)$ , e o primeiro “ponto-médio” é calculado em  $(1, b - \frac{1}{2})$ . Assim,

$$F(1, b - \frac{1}{2}) = b^2 + a^2(b - \frac{1}{2})^2 - a^2b^2 = b^2 + a^2(-b + \frac{1}{4}) \quad (3.21)$$

```

void MidpointEllipse(int a, int b, int color){
    int x, int y;
    float d1, d2;

    /* Valores iniciais */
    x = 0;
    y = b;
    d1 = b * b - a * a * b + a * a / 4.0;

    EllipsePoints(x, y, color); /* Simetria de ordem 4 */
    while(a * a * (y - 0.5) > b * b * (x + 1)){
        /* Região 1 */
        if (d1 < 0)
            d1 = d1 + b * b * (2 * x + 3);
        x++;
        }else{
            d1 = d1 + b * b * (2 * x + 3) + a * a * (-2 * y + 2);
            x++;
            y--;
        }/*end if*/
        EllipsePoints(x, y, color);
    }/* end while */

    d2 = b * b * (x + 0.5) * (x + 0.5) + a * a * (y - 1) * (y - 1) - a * a * b * b;
    while(y > 0){
        /* Região 2 */
        if (d2 < 0){
            d2 = d2 + b * b * (2 * x + 2) + a * a * (-2 * y + 3);
            x++;
            y--;
        }else{
            d2 = d2 + a * a * (-2 * y + 3);
            y--;
        }/*end if*/
        EllipsePoints(x, y, color);
    }/* end while */
}/*end MidpointEllipse*/

```

**Algoritmo 3.7:** Algoritmo do Ponto-Médio para conversão matricial de elipses.

### 3.5 Correção no Traçado

Para a maioria dos dispositivos gráficos, o retângulo de visualização não é quadrado, e as densidades de pixels na horizontal e na vertical são diferentes. Ou seja, a razão de aspecto física raramente é igual a 1, e a consequência é uma distorção visível no traçado. Por exemplo, um quadrado (especificado em coordenadas do usuário) pode ser deformado num retângulo ao ser convertido em coordenadas do dispositivo, ou uma circunferência pode ser distorcida numa elipse.

A forma mais simples de corrigir esta deformação é através de uma transformação de escala dos dados que definem a curva a ser traçada. Ou seja, no caso da circunferência, bastaria converter os pontos que definem a circunferência em pontos que na verdade definem uma elipse, mas que, como existe a deformação, aparecem na tela como uma circunferência.

As diferenças de densidades representam de fato uma operação de escala não homogênea realizada pelo dispositivo. Para corrigí-la basta aplicar-se aos parâmetros das curvas a serem traçadas uma transformação de escala inversa, que compense a do dispositivo. Por exemplo, se a densidade horizontal é o dobro da densidade vertical, então as abscissas dos pontos extremos de cada segmento a traçar devem ser dobradas para que mantenham a mesma proporção em relação às ordenadas. Essa manipulação dos pontos das curvas deve ser feita pelo programa que efetua a interface entre os programas do usuário e os dispositivos gráficos de saída - os chamados **Pacotes gráficos de interfaceamento**, ou **Ambientes Gráficos**.

Outra questão complexa é a eliminação do efeito de “serrilhado” (*aliasing*) observável nas imagens geradas por métodos de conversão matricial. Este efeito é mais pronunciado nos trechos de arcos com inclinações próximas da horizontal ou vertical. As técnicas usualmente aplicadas para a correção desse efeito são “caras” (do ponto de vista computacional), e exigem que se faça um controle de intensidade do traço.

As técnicas de anti-serrilhado (*antialiasing*) traçam não só os pixels calculados pelo procedimento de conversão matricial, mas também os pixels vizinhos, acima e abaixo, com intensidades variáveis, que serão tanto maiores quanto mais próximos os pixels estiverem do centro da reta. O cálculo das intensidades dos pixels e a determinação das suas coordenadas são processos não triviais, e requerem um número bem maior de operações do que o efetuado pelos algoritmos de conversão estudados.

## 3.6 Antialiasing

As primitivas geradas apresentam o problema da aparência “serrilhada”, ou “efeito escada”. Este efeito indesejável é o resultado da abordagem “tudo ou nada” adotada no processo de conversão matricial, no qual cada pixel recebe a cor associada à primitiva ou mantém a sua cor original. Esse efeito é apenas uma das várias manifestações visíveis do fenômeno conhecido como aliasing. A aplicação de técnicas que reduzem (ou eliminam) o efeito de *aliasing* é denominado *antialiasing*, e primitivas ou imagens geradas utilizando tais técnicas são ditas *antialiased*.

Considere o algoritmo do ponto médio para traçado de segmentos de reta sendo usado para traçar uma linha preta de um pixel de espessura, com inclinação entre 0 e 1, em um fundo branco. Em cada coluna de pixels interceptada pela linha o algoritmo seta a cor preta ao pixel que está mais perto do ponto ideal pertencente à linha. A cada vez que se passa para a próxima coluna e o pixel mais próximo da linha ideal é o que está a nordeste (NE), e não à leste (E), ocorre um “efeito escada”. O mesmo acontece em outros tipos de primitivas. Uma forma de reduzir o problema (em termos visuais) é aumentar a resolução do dispositivo (usar mais pixels) (ver Figura 3.14). Esta é uma solução cara em termos de memória e tempo gasto no processo de conversão matricial, e que não resolve o problema. Alternativas menos custosas são descritas a seguir.

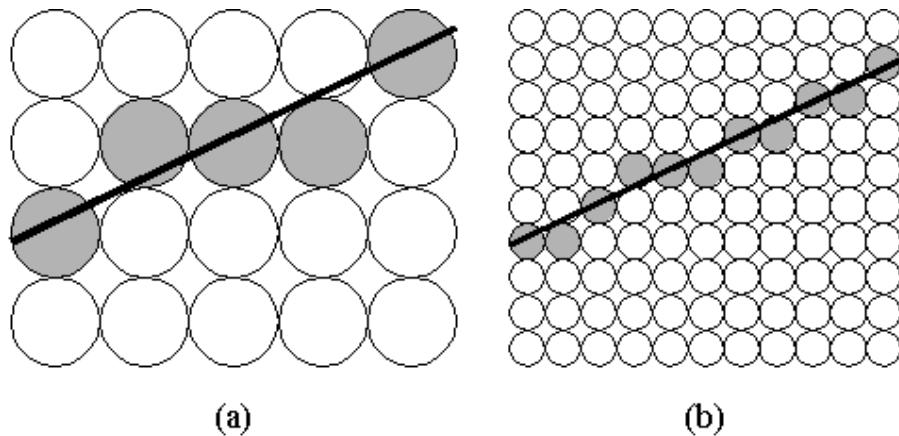


Figura 3.14: Segmento de reta renderizado com o algoritmo do ponto médio em diferentes escalas. (a) é uma ampliação da região central de (b).

### 3.6.1 Amostragem de Áreas não Ponderada

O fato é que, apesar da primitiva ideal, como um segmento de reta, ter espessura zero, a primitiva sendo traçada tem espessura não nula, pois ocupa uma área finita da tela. Assim, podemos pensar em um segmento de reta como um retângulo com uma certa espessura que cobre uma região da malha de

pixels, como ilustrado na Figura 3.15. Conseqüentemente, o melhor procedimento não é atribuir o valor correspondente a preto a um único pixel em uma coluna, mas atribuir diferentes intensidades da cor a cada pixel interceptado pela área coberta pelo retângulo que aproxima a linha, na coluna em questão. Assim, linhas horizontais ou verticais, cuja espessura é 1 pixel, afetam exatamente 1 pixel em uma coluna ou linha. Para linhas com outras inclinações, mais do que 1 pixel é traçado, cada qual com uma intensidade apropriada.

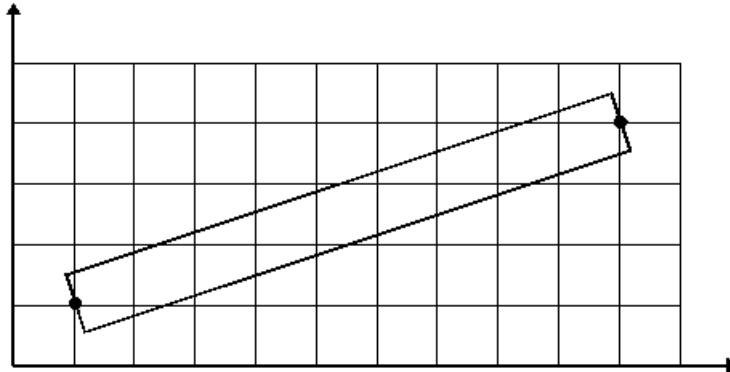


Figura 3.15: Segmento de reta definido com uma espessura diferente de zero.

Mas, qual é a geometria de um pixel? Qual o seu tamanho? Qual intensidade a atribuir a um pixel interceptado pela linha? É computacionalmente simples assumir que os pixels definem uma malha regular de “quadrados” não sobrepostos que recobre a tela, sendo que as intersecções da malha definem o posicionamento dos pixels, i.e., pixels são tratados como “quadrados” com centro nas interseções da malha. Uma primitiva pode sobrepor toda ou parte da área ocupada por um pixel. Assumimos também que a linha contribui para a intensidade do pixel segundo um valor proporcional à porcentagem da área do pixel coberta por ela. Se a primitiva cobre toda a área do pixel, ele recebe a intensidade máxima (preto, no caso), se cobre parte o pixel recebe um tom de cinza cuja intensidade é proporcional à área coberta (Figura 3.16). O efeito é “suavizar” a definição da reta ou das arestas que definem a primitiva, melhorando sua aparência visual quando observada à distância. Essa estratégia é denominada amostragem por área não ponderada (*unweighted area sampling*). Uma estratégia ainda mais eficiente é a amostragem por área ponderada (*weighted area sampling*).

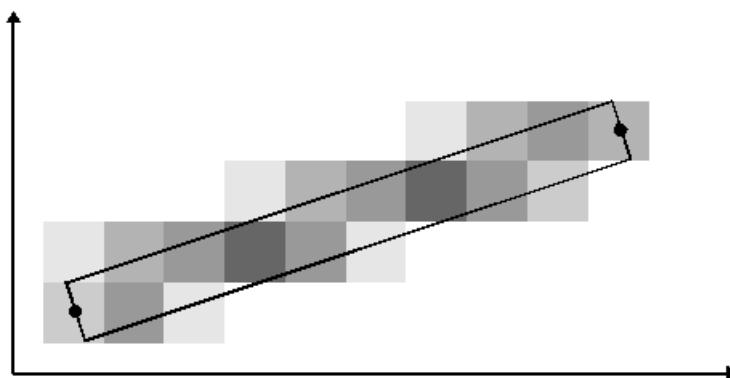


Figura 3.16: A intensidade do pixel é proporcional à área coberta.

### Amostragem de Áreas Ponderada

Nessa outra estratégia, assim como na anterior, valem duas propriedades: 1) a intensidade atribuída ao pixel é diretamente proporcional à área do mesmo coberta pela primitiva; 2) se a primitiva não intercepta a área do pixel, então ela não contribui nada para a cor do mesmo. A diferença é que, na estratégia ponderada, ao determinar a contribuição de uma área considera-se, também, a sua proximidade em relação ao centro do pixel. Assim, áreas iguais podem contribuir de forma desigual: uma área pequena próxima do centro do pixel tem maior influência do que uma área grande a uma distância maior. Entretanto, para garantir que a primitiva contribui para um pixel apenas se ela sobrepõe à área ocupada pelo mesmo, precisamos mudar a geometria do pixel da seguinte forma: vamos considerar que o pixel ocupa uma área

circular maior do que a área quadrangular considerada na situação anterior. Se a primitiva sobrepõe essa nova área, ela contribui para a intensidade do pixel.

Vamos definir uma função peso que determina a influência, na intensidade do pixel, de uma área pequena da primitiva, denominada  $dA$ , em função da distância de  $dA$  ao centro do pixel. No caso da amostragem não ponderada, essa função é constante, e no caso da amostragem ponderada, ela diminui de forma inversamente proporcional à distância em relação ao centro do pixel. Podemos pensar nessa função como uma função  $W(x, y)$ , no plano, cuja altura em relação ao plano  $xy$  dá o peso associado à área  $dA$  no ponto  $(x, y)$ . Para a amostragem por área não ponderada, essa função é representada por uma 'caixa', como indicado na Figura 3.17. A figura mostra os pixels como quadrados cujos centros (marcados) estão nas intersecções da malha regular. A contribuição de cada pequena área é proporcional à área multiplicada pelo peso. Portanto, a intensidade total é dada pela integral da função peso sobre a área de sobreposição. O volume representado por essa integral,  $Ws$ , é sempre uma fração entre 0 e 1, e a intensidade  $I$  do pixel é dada por  $Imax \cdot Ws$ . No caso da amostragem não ponderada a altura da 'caixa' é normalizada para 1, de forma que o volume da caixa é 1. No caso da área da linha cobrir todo o pixel, tem-se  $I = Imax \cdot 1 = Imax$ .

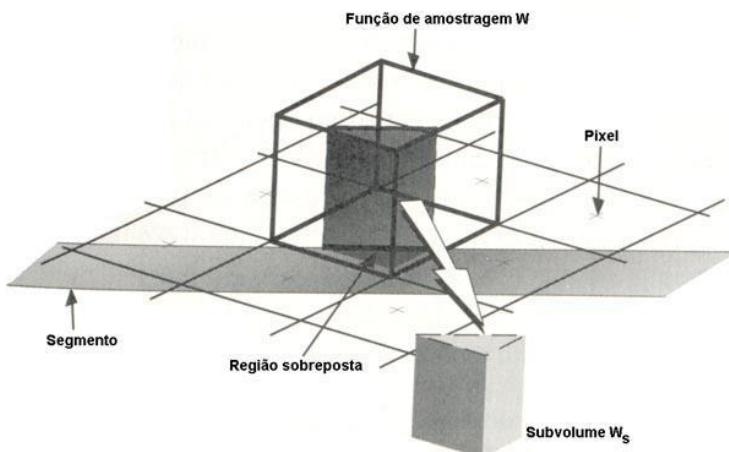


Figura 3.17: Filtro definido por um cubo para um pixel definido por um quadrado.

Vamos agora construir uma função peso para a amostragem ponderada: a função adotada tem valor máximo no centro do pixel e diminui linearmente a medida que a distância ao centro aumenta, e seu gráfico define um cone, como indicado na Figura 3.18. A base do cone é uma circunferência cujo centro coincide com o centro do pixel, e cujo raio corresponde a uma unidade da distância entre os pixels na malha. Note que as bases das funções peso em diferentes pixels se sobrepõem, e portanto uma única pequena região da primitiva pode contribuir para a intensidade de diferentes pixels. Essa sobreposição também garante que não existem áreas da grade que não são cobertas por algum pixel (esse não seria o caso se o raio da base fosse de meia unidade, por exemplo).

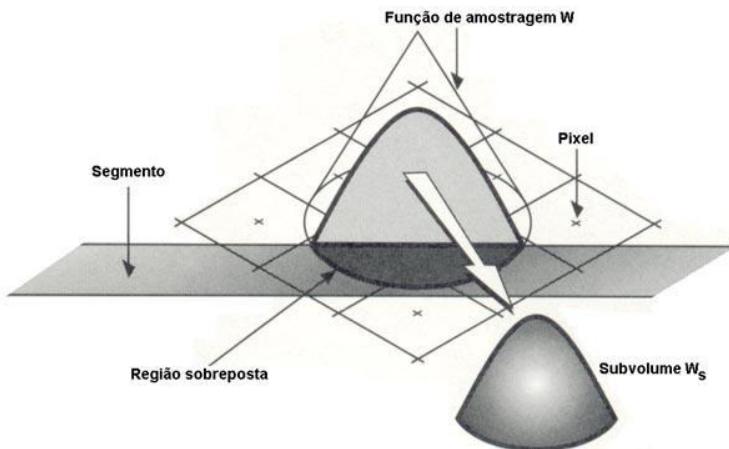


Figura 3.18: Filtro cônico com diâmetro igual ao dobro da largura de um pixel.

Assim como na função 'caixa', a soma das contribuições de intensidade para o cone é dada pelo

volume sob o cone que está na região de interseção da primitiva com a base do cone. Esse volume,  $W_s$ , é uma seção vertical do cone, como mostrado na Figura 3.18. A altura do cone é normalizada para que o volume do mesmo seja igual a 1, garantindo que um pixel que está inteiramente sob a base do cone seja traçado com intensidade máxima. Contribuições de áreas da primitiva que interceptam a base mas estão distantes do centro do pixel são pequenas, mas áreas muito pequenas, porém próximas do centro do pixel contribuem mais para a cor do mesmo. O resultado dessa abordagem é reduzir o contraste entre pixels adjacentes, de forma a gerar transições visuais mais suaves. Particularmente, linhas horizontais ou verticais com espessura de um pixel agora resultam em mais do que uma única linha ou coluna de pixels acesos, o que não ocorre na abordagem não ponderada. Essa abordagem pode ser embutida nos algoritmos de conversão matricial, como descrito em Foley et al. [Fol90, seção 3.17.4]. Outros tipos de funções podem ser usadas, sendo que existem funções ótimas melhores do que a função cônica (v. Foley, seção 4.10).

## Capítulo 4

# Preenchimento de Polígonos

A tarefa de preencher um polígono pode ser dividida em duas:

1. Decidir que pixels pintar para preencher o polígono;
2. Decidir com que valor pintar o pixel.

Geralmente decidir que pixels devem ser pintados, consiste em se varrer (*scan*) sucessivas linhas que interceptam a primitiva, preenchendo os pixels em “blocos” (*span*) que estão dentro da primitiva (que define o polígono) da esquerda para a direita. Vamos considerar inicialmente o preenchimento de primitivas gráficas não “recortadas” (*unclipped*) com uma única cor. Em geral, a decisão sobre quais pixels preencher é feita “varrendo-se” (*scan*) linhas sucessivas que interceptam a primitiva, e preenchendo, da esquerda para a direita, blocos de pixels adjacentes (*spans*) que estão dentro da primitiva que define o polígono.

### 4.1 Retângulos

Para preencher um retângulo com uma única cor, pinta-se cada pixel dentro de uma linha de varredura da esquerda para a direita com o mesmo valor de pixel, isto é, pode-se preencher cada “bloco” de  $x_{min}$  a  $x_{max}$ . As primitivas para blocos exploram a Coerência Espacial: o fato que freqüentemente não há alteração nas primitivas de um pixel para outro (pixels adjacentes) dentro de um bloco (*span*) ou de uma linha de varredura para a próxima linha de varredura. Assim, podemos explorar a coerência para buscar apenas os pixels em que ocorrem mudanças. Para um retângulo preenchido com a mesma cor, todos os pixels num bloco receberão um mesmo valor, o que garante coerência de bloco (*span coherence*). O retângulo também exibe coerência de linha de varredura (*scan-line coherence*), no sentido de que linhas de varredura consecutivas que interceptam o retângulo são idênticas. Posteriormente consideraremos também coerência de arestas (*edge coherence*) para lados de polígonos em geral. Coerência é uma propriedade bastante explorada em CG, não apenas para conversão matricial de primitivas 2D, mas também para visualização de primitivas 3D.

A capacidade de tratar vários pixels identicamente em um bloco (*span*) tem especial importância para a diminuição de tempo gasto para gravar os pixels no *Frame Buffer* (memória onde é armazenada a imagem). Assim, se em menos acessos a memória pudermos escrever mais pixels estaremos ganhando tempo. Abaixo colocaremos um algoritmo para preencher um retângulo, sendo que nele nós não nos preocuparemos em escrever eficientemente na memória. Deixaremos isso para discussão posterior.

```
Para y = ymin até ymax do retângulo { Por linha de varredura }
    Para x = xmin até xmax { Cada pixel dentro do bloco }
        write_pixel (x,y,valor)
```

**Algoritmo 4.1:** Trecho de Algoritmo para Preenchimento de Retângulo.

Consideremos agora dois retângulos que compartilham um mesmo lado. Se preenchermos cada retângulo isoladamente, o lado (aresta) compartilhado pelos dois será gerado duas vezes o que não é desejável (perco tempo!). Aqui surge o problema de definição de área pertencente a cada primitiva, isto é, quais os pixels que pertencem a cada primitiva e quais os pixels que não pertencem. De forma natural, podemos definir os pixels que matematicamente se encontram no interior de uma área definida pela primitiva, como pertencente a ela. Mas o que podemos falar acerca dos pixels que estão no limite da primitiva?

Se voltamos ao nosso problema do retângulo podemos adotar como uma regra que: os pixels que constituem os limites da primitiva, isto é, os que estão sobre os seus lados, não são considerados parte da primitiva se o meio-plano definido pelo lado e que contém a primitiva está abaixo ou a esquerda do lado. Ou, mais claramente, os pixels que estão sobre os lados (aresta) esquerdo e inferior pertencem a primitiva e serão desenhados, porém os lados superior e direito não pertencem a primitiva e portanto não serão desenhados. Assim, uma aresta vertical compartilhada por dois retângulos pertence ao lado que está mais a direita. Sendo que na realidade, blocos dentro de um retângulo representam uma intervalo que é fechado a esquerda e embaixo e aberto em cima e a direita.

Algumas considerações devem ser feitas sobre esta regra:

1. A regra se aplica da mesma forma a polígonos arbitrários e não somente a retângulos.
2. O vértice do canto inferior esquerdo ainda continua sendo desenhado duas vezes.
3. A regra pode também ser aplicada para retângulos e polígonos não preenchidos.
4. A regra faz com que em cada bloco esteja faltando o seu pixel mais a direita, e em cada retângulo fique faltando o seu lado (aresta) superior.

Os problemas apresentados nas considerações acima demonstram que não há solução perfeita para o problema de não escrever mais de uma vez linhas que sejam potencialmente compartilhadas por mais de um polígono.

## 4.2 Polígonos de Forma Arbitrária

O algoritmo que vamos discutir a seguir contempla tanto polígonos côncavos quanto polígonos convexos, mesmo que eles tenham auto-intersecção e buracos em seu interior. Ele opera computando blocos de pixels que estão entre os lados esquerdo e direito do polígono. O extremo do bloco é calculado por um algoritmo incremental que calcula a linha de varredura/lado de intersecção a partir da intersecção com a linha de varredura anterior.

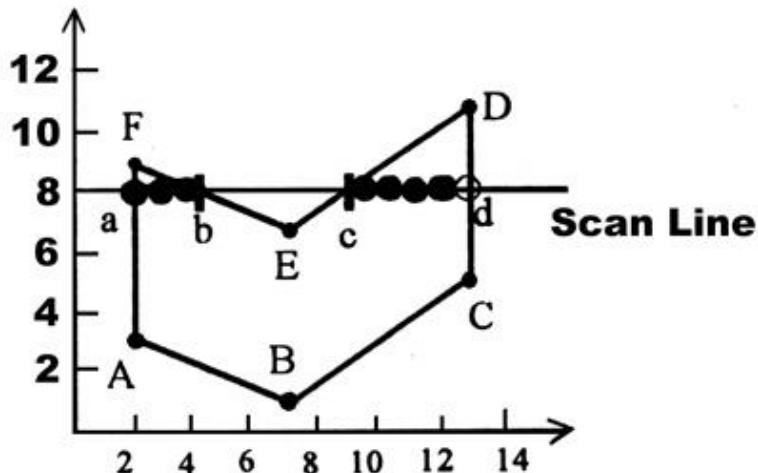


Figura 4.1: Esta figura ilustra o processo de linha de varredura para um polígono arbitrário. As intersecções da linha de varredura 8 com os lados FA e CD possuem coordenadas inteiros, enquanto as intersecções com os lados EF e DE possuem coordenadas reais.

É preciso determinar que pixels da linha de varredura estão dentro do polígono, e estabelecer os pixels correspondentes (no exemplo da Figura 4.1, blocos para  $x = 2$  até 4 e 9 até 13) com seu valor apropriado. Repetindo este processo para cada linha de varredura que intercepta o polígono, o polígono inteiro é convertido. A Figura 4.2 ilustra este processo.

Devemos nos preocupar com a definição dos pontos extremos do polígono. Uma forma de obtê-los é usar o algoritmo de conversão de linhas do “Ponto-Médio” para cada lado do polígono. Note que esta estratégia inclui alguns pixels no extremo do polígono que na realidade não se encontram no polígono. Eles foram escolhidos pelo algoritmo de conversão matricial de linhas, e são colocados sobre o lado porque estão mais próximas a ele.

Não podemos desenhar pixels que não pertençam verdadeiramente ao interior do polígono, pois podemos estar invadindo o domínio de outro polígono. Isto é, se a linha em questão define um extremo

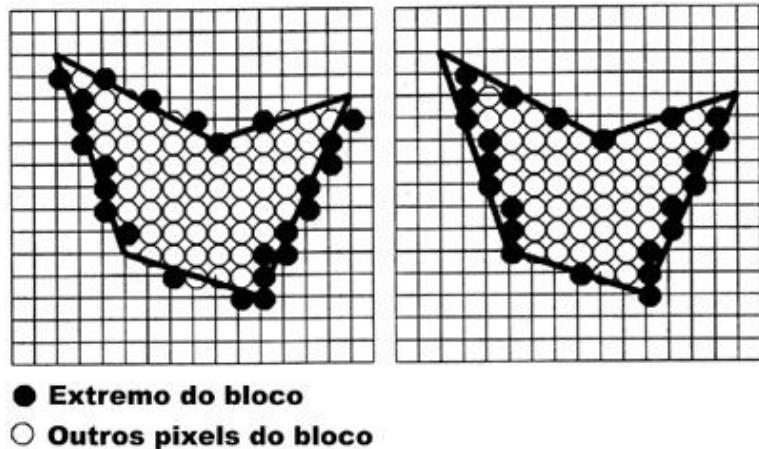


Figura 4.2: Linhas de varredura em um polígono. Os extremos em preto, e os pixels no interior em cinza.  
(a) Extremo calculado pelo algoritmo do "Meio-Ponto". (b) Extremo interior ao polígono.

que é compartilhado por outro polígono, estaremos entrando na região definida por ele. Assim, é óbvio que é preferível traçar apenas os pixels que estejam estritamente na região definida pelo polígono, mesmo que um pixel exterior esteja mais próximo a aresta real (pelo algoritmo de conversão ("Ponto-Médio")). Logo, o algoritmo de conversão deve ser ajustado de acordo. Compare a Figura 4.2(a) com 2(b), e note que os pixels fora da primitiva não são desenhados na parte (b).

Com esta técnica, um polígono não invade a região definida por outro - a mesma técnica pode ser aplicada para polígonos não preenchidos, por uma questão de consistência. Poderíamos, alternativamente, converter retângulos e polígonos não cheios convertendo independentemente cada segmento de reta que os compõe, mas nesse caso polígonos cheios e não cheios não teriam os mesmos pixels na fronteira!

Como no algoritmo original do "Ponto-Médio", podemos usar um algoritmo incremental para calcular o bloco (*span*) extremo sobre uma linha de varredura a partir da linha de varredura anterior, sem ter que calcular analiticamente os pontos de intersecção da linha de varredura com os lados do polígono.

O processo de preencher os polígonos pode ser dividido em três passos:

1. Obter a intersecção da linha de varredura com todos os lados do polígono.
2. Ordenar os pontos de intersecção (em  $x$  crescente).
3. Preencher os pixels entre pares de pontos de intersecção do polígono que são internos a ele. Para determinar quais os pares que são internos ao polígono, podemos usar a regra de Paridade: A paridade inicialmente é par, e a cada intersecção encontrada o bit de paridade é invertido, o pixel é pintado quando a paridade é ímpar, e não é pintado quando é par.

Vamos tratar dos dois primeiros passos do processo mais adiante, e consideremos agora a estratégia usada para o preenchimento dos blocos (passo 3). Na Figura 4.1, a lista ordenada de coordenadas  $x$  é (2, 4.5, 8.5, 13) para a linha de varredura 8. Há 4 pontos a serem discutidos sobre este passo:

### I Se a intersecção é um valor fracionário, como determinar qual o pixel que deverá ser tomado para que fique interior ao polígono?

O valor deverá ser arredondado de forma a que o ponto fique dentro do polígono. Se estamos dentro do polígono (paridade ímpar) e atingimos uma intersecção fracionária pela direita, arredondamos a coordenada  $x$  da intersecção para baixo; se estamos fora do polígono arredondamos para cima. Isso garante que sempre teremos um ponto dentro do polígono.

### II Como tratar o caso de intersecção com coordenadas inteiras?

Pode-se utilizar o critério visto anteriormente, para evitar conflitos entre lados compartilhados em retângulos. Se a coordenada  $x$  de um pixel mais a esquerda de um bloco (*span*) é inteira ele é definido como interno; se a coordenada  $x$  do pixel mais a direita de um bloco é inteira, ele é definido como externo ao polígono.

### III Como tratar o caso II para vértices que são compartilhados por mais de uma aresta do polígono?

Usamos a técnica de paridade. Ou seja, contamos o vértice de  $y_{min}$  de um lado para alterar a paridade, mas não contamos o vértice de  $y_{max}$ , dessa forma o vértice de  $y_{max}$  é desenhado somente

se ele é o vértice de  $ymin$  do lado adjacente. Por exemplo, na Figura 4.1, o vértice A é contado uma vez no cálculo de paridade porque é o vértice de  $ymin$  para o lado FA, mas é também o vértice de  $ymax$  para o lado AB. Assim ambos os lados e blocos são tratados como intervalos que são fechados em seu valor mínimo e abertos em seu valor máximo.

#### IV Como tratar o caso especial de II em que os vértices definem uma linha horizontal?

Assim como no caso dos retângulos, arestas horizontais inferiores são traçadas, e arestas horizontais superiores não são. Como veremos, isso acontece automaticamente se não contarmos os vértices dessas arestas no cálculo da paridade, o que é natural, já que eles não correspondem a vértices  $ymin$  ou  $ymax$ .

Ilustremos este processo, aplicando essas regras à linha de varredura 8, na Figura 4.1. Os pixels serão preenchidos a partir do ponto a (coordenadas (2, 8)), até o primeiro pixel a esquerda do ponto b (coordenadas (4, 8)); e do primeiro pixel a direita do ponto c (coordenadas (9, 8)) até 1 pixel a esquerda do ponto d (coordenadas (12, 8)). Para a linha de varredura 3, o vértice A conta uma vez porque é o vértice de  $ymin$  do lado FA (e é também o vértice  $ymax$  do lado AB), isto causa paridade ímpar, assim o bloco é desenhado a partir de A até um pixel a esquerda da intersecção com o lado CB, onde a paridade é estabelecida como par e o bloco é terminado. A linha de varredura 1 passa apenas pelo vértice B, e os lados AB e BC tem como vértice de  $ymin$  B, o qual é dessa forma contado como paridade par. Este vértice atua como um bloco nulo, pois a linha de varredura entra pelo vértice, desenha o pixel e sai por ele mesmo. Assim, um pixel mínimo local é pintado, porém pixel de máximo local não o é. Isso acontece com a intersecção da linha de varredura 9 com o vértice F, compartilhado pelos lados FA e EF. Para ambos os lados, B é vértice  $ymax$ , e dessa forma não afeta a paridade, que continua par.

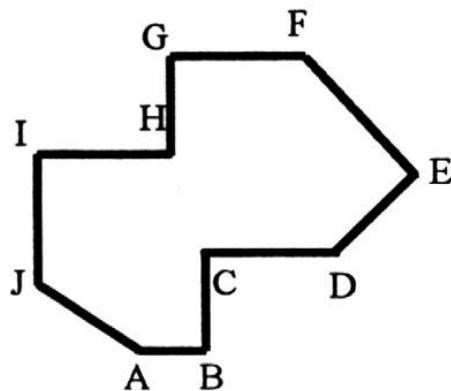


Figura 4.3: Tratamento dos lados horizontais de um polígono.

##### 4.2.1 Arestas Horizontais

Examinemos os casos apresentados na Figura 4.3. Consideremos o lado AB (lado inferior). O vértice A é vértice  $ymin$  para o lado JA, e AB por ser horizontal não possui mínimo. Assim, a paridade é ímpar (pela regra da paridade!) e o bloco (*span*) é desenhado. O lado vertical BC tem  $ymin$  em B, e pela mesma razão anterior o lado AB não contribui para alterar a paridade, assim a paridade torna-se par e o bloco termina. O lado IJ tem vértice  $ymin$  em J (e em JA ele é máximo) assim a paridade torna-se ímpar e o bloco é desenhado até o lado BC. O bloco que começa em IJ e encontra C, não termina aí, porque C é  $ymax$  para BC, assim o bloco continua sobre o lado CD (até D), no entanto DE tem  $ymin$  em D e a paridade torna-se par e o bloco termina. O lado IJ tem  $ymax$  em I e o lado HI não contribui para a cálculo da paridade, assim a paridade continua e o bloco sobre o lado HI não é desenhado. Entretanto, o lado GH tem vértice  $ymin$  em H, e a paridade torna-se ímpar, e o bloco é desenhado a partir de H até o pixel a direita da intersecção da linha de varredura com o lado EF. Finalmente não há vértice  $ymin$  em G nem em F, e dessa forma o lado (que é o lado superior - topo) FG não é desenhado.

##### OBSERVAÇÕES:

- O algoritmo acima contempla polígonos que possuem auto-intersecção. Mas não desenha os pixels das arestas superiores nem os das arestas à direita, mesmo se a primitiva for isolada.
- Não desenha os pixels sobre o topo interno de um polígono côncavo em formato de U.

- Não desenha os pixels que são pontos máximos locais.
- Neste algoritmo, em vez de escrever pixels em lugares errados, pixels não são escritos (mesmo em seus lugares certos!).

#### 4.2.2 Slivers

Existe um outro problema com o nosso algoritmo de conversão matricial: polígonos com lados muito próximos criam um “sliver” - uma área poligonal tão estreita que seu interior não contém um bloco de pixels para cada linha de varredura (Figura 4.4). Devido à regra de que são traçados apenas os pixels que estejam no interior ou sobre arestas inferiores ou à esquerda, podem existir muitas linhas de varredura com um único pixel, ou sem nenhum. A ausência de pixels é um outro exemplo do problema de *aliasing*, ou seja, da representação de um sinal contínuo através de uma aproximação discreta (digital). Para melhorar a aparência nesses casos, pode-se usar técnicas de *antialiasing* (se tivermos múltiplos bits por pixel). *Antialiasing* implicaria em “suavizar” nossa regra de “traçar apenas pixels que estejam no interior ou numa aresta inferior ou à esquerda”, de forma a permitir que pixels na fronteira ou mesmo no exterior do polígono assumam intensidades variando como uma função da distância entre o centro do pixel e a primitiva. Nesse caso, múltiplas primitivas podem contribuir para o valor de um pixel.

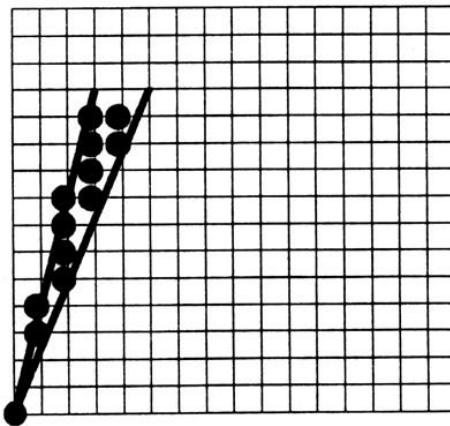


Figura 4.4: Exemplo de uma conversão matricial de um *Sliver*

#### 4.2.3 Algoritmo para Conversão Matricial de Segmento de Reta que Utiliza “Coerência de Arestas” de um Polígono

No método anterior usávamos “força bruta” para calcular a intersecção da linha de varredura com os lados de um polígono. Mas, freqüentemente, somente poucos lados de um polígono são de interesse para uma dada linha de varredura. Além disso, notemos que muitos lados interceptados por uma linha de varredura  $i$  também são pela linha de varredura  $i + 1$ . Esta “Coerência de Arestas” ocorre para muitas linhas de varreduras que interceptam um lado. Ao movermos de uma linha de varredura para a próxima, podemos calcular o próximo  $x$  da intersecção da linha de varredura com o lado, a partir do  $x$  da linha de varredura anterior que interceptou o lado. Usando a mesma técnica do algoritmo do “Ponto-Médio”, temos:

$$x_{i+1} = x_i + \frac{1}{m} \quad (4.1)$$

onde  $m = \frac{(y_{max}-y_{min})}{(x_{max}-x_{min})}$  é a inclinação da aresta.

No algoritmo do ponto-médio para conversão matricial de segmentos de reta, evitamos usar aritmética fracionária calculando uma variável de decisão inteira, e verificando o seu sinal para escolher o pixel mais próximo à reta “verdadeira”. Aqui, também gostaríamos de usar aritmética inteira para calcular o pixel interior mais próximo da verdadeira intersecção. Consideraremos arestas com inclinação maior que 1 que são arestas a esquerda. Arestas a direita e outras inclinações são tratadas com argumentos similares aos que serão vistos para esse caso, e arestas verticais são casos especiais. (Arestas horizontais são tratadas implicitamente pelas regras de traçado de blocos, como vimos anteriormente.)

Precisamos traçar um pixel no extremo  $(x_{min}, y_{min})$ . A medida que  $y$  é incrementado, a coordenada  $x$  do ponto na linha ideal será aumentada de  $\frac{1}{m}$ . Este aumento resultará num valor de  $x$  com uma parte inteira e uma parte fracionária, a qual pode ser expressa como uma fração com denominador igual a

$y_{max} - y_{min}$ . Com a repetição do processo, a parte fracionária vai se tornar maior do que 1 (*overflow*), e a parte inteira terá de ser incrementada. Por exemplo, se a inclinação é  $\frac{5}{2}$ , e  $x_{min}$  é 3, a seqüência de valores de  $x$  será  $3, 3\frac{2}{5}, 3\frac{4}{5}, 3\frac{6}{5} = 4\frac{1}{5}$

e assim por diante. Quando a parte fracionária de  $x$  é zero, podemos traçar diretamente o pixel  $(x, y)$  que está sobre a linha, mas quando ela é diferente de zero precisamos arredondar (para cima!) de forma a tomar um pixel que esteja estritamente dentro da linha. Quando a parte fracionária de  $x$  torna-se maior do que 1, incrementamos  $x$ , subtraímos 1 da parte fracionária, e movemos um pixel a direita. Se com o incremento obtemos um valor inteiro para  $x$ , traçamos o pixel correspondente e decrementamos a fração de 1, de forma a mantê-la menor do que 1.

Podemos evitar o uso de aritmética fracionária mantendo apenas o numerador da fração, e observando que a parte fracionária será maior do que 1 quando o numerador tornar-se maior que o denominador. O algoritmo abaixo implementa esta técnica, usando a variável incremento para armazenar as sucessivas adições feitas no numerador até que ocorra um *overflow* (ultrapasse o denominador), quando o numerador é decrementado pelo denominador, e  $x$  é incrementado.

```
void LeftEdgeScan (int xmin, int ymin, int xmax, int ymax, int valor){
    int x, y;

    x = xmin;
    y = ymin;
    numerador = xmax - xmin;
    denominador = ymax - ymin;
    incremento = denominador;

    for (y = ymin; y <= ymax; y++){
        PintaPixel (x,y,valor);
        incremento+=numerador;

        if (incremento > denominador){
            /* Overflow, Arredonda o próximo pixel e decremente o incremento */
            x++;
            incremento-=denominador;
        }/* end if */
    }/* end for */
}/* end LeftEdgeScan */
```

**Algoritmo 4.2:** Conversão matricial da aresta esquerda de um polígono.

Vamos desenvolver um algoritmo de conversão matricial de linhas que se aproveita da coerência de arestas e, para cada linha de varredura, armazena o conjunto de arestas por ela interceptadas numa estrutura de dados denominada AET - *active-edge table* (tabela das arestas ativas). As arestas na AET são ordenadas de acordo com as coordenadas  $x$  dos pontos de intersecção, de forma que possamos preencher os blocos definidos por pares de valores de intersecção (devidamente arredondados) - que definem as extremidades dos blocos.

A medida que movemos para a próxima linha de varredura,  $y+1$ , a AET é atualizada. Primeiramente, arestas que estão na AET mas não são interceptadas por esta próxima linha de varredura (ou seja, aquelas para as quais  $y_{max} = y$ ) são removidas. Depois, quaisquer novas arestas interceptadas por essa linha (ou seja, aquelas para as quais  $y_{min} = y + 1$ ) são incluídas na AET. Finalmente, para as arestas que ainda estão na AET, os novos valores de  $x$  das intersecções são calculados, usando o algoritmo incremental para conversão matricial de arestas já visto.

Para que a inclusão de novas arestas na AET seja eficiente, criamos uma tabela de arestas global (ET - *Edge Table*), contendo todas as arestas do polígono em ordem crescente de sua menor coordenada  $y$  ( $y_{min}$ ). A ET é uma tabela *hashing* aberta (v. Aho, Data Structures and Algorithms), com  $p$  posições (*buckets*, ou "cestos"), uma para cada linha de varredura. A cada "cesto"  $y$  é associada uma lista encadeada com as arestas com  $y_{min} = y$ . As arestas na lista estão ordenadas em ordem crescente da coordenada  $x$  de sua extremidade inferior ( $x_{min}$ ). Cada nó da lista armazena ainda a coordenada  $y_{max}$  da aresta, e o incremento em  $x$  utilizado para passar para a próxima linha de varredura,  $\frac{1}{m}$ . A Figura 4.5 mostra como estariam ordenadas as seis arestas do polígono da Figura 4.1, e a Figura 4.6 mostra a

AET para as linhas de varredura 9 e 10 daquele polígono. Uma vez construída a ET, os passos para o algoritmo de conversão de arestas são dados no Algoritmo 4.3.

Este algoritmo explora **coerência de arestas** para calcular o valor  $x$  das intersecções e **coerência de linhas de varredura** (juntamente com ordenação) para calcular blocos. Uma vez que a ordenação trabalha com um número pequeno de arestas, e a reordenação do passo 3.6 é feita numa lista quase que totalmente ordenada, pode-se usar um método de ordenação por inserção, ou o método *bubblesort*, que é  $O(N)$ .

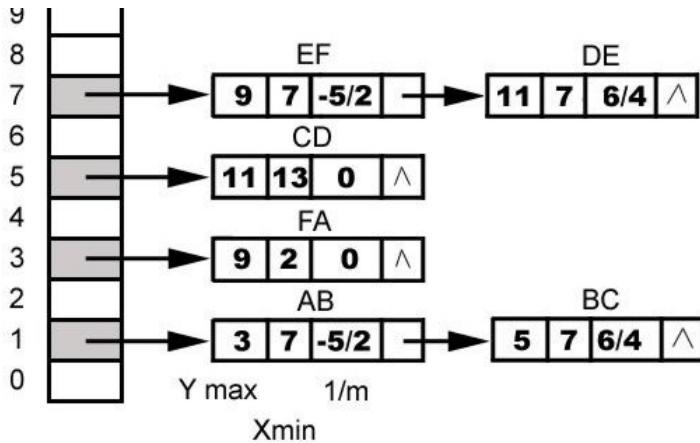


Figura 4.5: ET para o polígono de Figura 4.1.

Para efeito de conversão matricial, triângulos e trapezóides podem ser tratados como casos especiais de polígonos, uma vez que possuem apenas dois lados para cada linha de varredura (lembre-se que lados horizontais não são convertidos explicitamente). Na verdade, como um polígono arbitrário sempre pode ser decomposto numa malha de triângulos que compartilham vértices e arestas, poderíamos converter um polígono decompondo-o numa malha de triângulos, convertendo os triângulos a seguir. Este é um problema clássico da geometria computacional, o da **triangulação**. O processo de decomposição é simples para polígonos convexos.

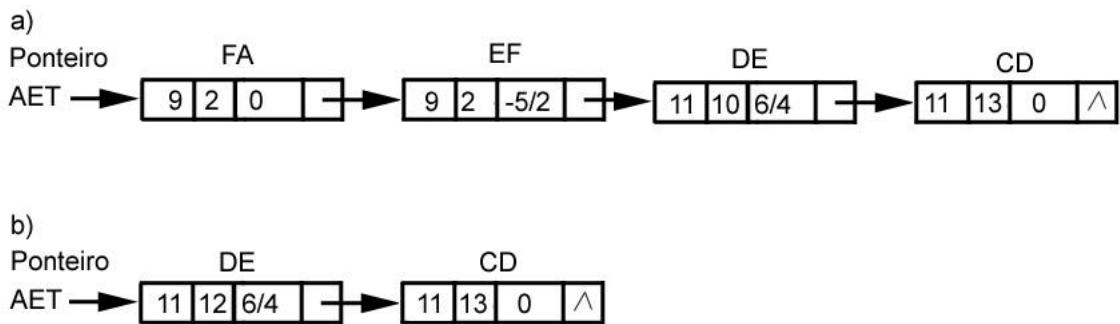


Figura 4.6: AET para o polígono da Figura 4.1 a) linha de varredura 9 b) linha de varredura 10. Note que a coordenada x da aresta DE em (b) foi arredondada para cima..

Note que o cálculo dos blocos é cumulativo. Ou seja, quando a iteração corrente do algoritmo de conversão de arestas gera um ou mais pixels na mesma linha de varredura que os pixels gerados anteriormente, os blocos devem ser atualizados se um novo pixel tem um valor de  $x$  máximo ou mínimo mais extremo. Tratar casos de cálculo de blocos para arestas que se cruzam, ou para *slivers* requer um tratamento especial. Podemos calcular os blocos num passo, e preenchê-los num segundo passo, ou calcular um bloco e preenchê-lo num único passo. Outra vantagem de se utilizar blocos é que o processo de *clipping* pode ser feito simultaneamente ao cálculo dos blocos: os blocos podem ser recortados (*clipped*) individualmente nas coordenadas esquerda e direita do retângulo de corte.

1. Obtém a menor coordenada  $y$  armazenada na ET; ou seja, o valor de  $y$  do primeiro “cesto” não vazio.
2. Inicializa a AET como vazia.
3. Repita até que a ET e a AET estejam vazias:
  - 3.1. Transfere do cesto  $y$  na ET para a AET as arestas cujo  $y_{min} = y$  (lados que estão começando a serem varridos), mantendo a AET ordenada em  $x$ ,
  - 3.2. Retira os lados que possuem  $y = y_{max}$  (lados não mais envolvidos nesta linha de varredura,
  - 3.3. Desenhe os pixels do bloco na linha de varredura  $y$  usando pares de coordenadas  $x$  da AET.
  - 3.4. Incremente  $y$  de 1 (coordenada da próxima linha de varredura).
  - 3.5. Para cada aresta não vertical que permanece na AET, atualiza  $x$  para o novo  $y$ .
  - 3.6. Como o passo anterior pode ter desordenado a AET, reordena a AET.

**Algoritmo 4.3:** Algoritmo de conversão de arestas baseado na coerência de arestas.

# Capítulo 5

## Transformações 2D e 3D

Transformações Geométricas (TG) são a base de inúmeras aplicações gráficas, podendo estar desde em simples programas para representar *layouts* de circuitos eletrônicos; em programas de planejamento de cidades, onde pode-se usar movimentos de translação para colocar os símbolos que definem edifícios e árvores em seus devidos lugares, rotações para orientar corretamente esses símbolos, e alteração de escala para adequar o tamanho desses símbolos; ou mesmo em sistemas de software sofisticados que permitem a construção de cenas realistas.

### 5.1 Transformações em 2D

Primeiramente estudaremos as transformações Geométricas em duas dimensões. Pode-se efetuar a Translação de pontos no plano  $(x, y)$  adicionando-se quantidades inteiras às suas coordenadas. Assim, cada ponto  $P(x, y)$  pode ser movido por  $d_x$  unidades em relação ao eixo  $x$ , e por  $d_y$  unidades em relação ao eixo  $y$ . Logo, o ponto  $P'(x', y')$ , pode ser escrito como:

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \end{aligned} \tag{5.1}$$

E se definimos os vetores colunas:

$$\begin{aligned} P &= \begin{bmatrix} x \\ y \end{bmatrix} \\ P' &= \begin{bmatrix} x' \\ y' \end{bmatrix} \\ T &= \begin{bmatrix} d_x \\ d_y \end{bmatrix} \end{aligned} \tag{5.2}$$

então a translação, expressa pela Equações , pode ser definida como:

$$P' = P + T \tag{5.3}$$

Podemos transladar um objeto, fazendo-o a todos os seus pontos (o que não é muito eficiente). Para transladar uma linha podemos fazê-lo apenas para seus pontos limites e sobre estes pontos redesenhar a linha. Isso também é verdade para alterações de Escala e Rotação. Na Figura 5.1 é mostrado o efeito de uma translação de um objeto por  $(3, -4)$ .

Pode-se efetuar Mudanças de Escala (ou apenas Escala) de um ponto pelo eixo x ( $s_x$ ), ou pelo eixo y ( $s_y$ ), através das multiplicações:

$$\begin{aligned} x' &= s_x \cdot x \\ y' &= s_y \cdot y \end{aligned} \tag{5.4}$$

Ou em forma matricial:

$$P' = S \cdot P \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{5.5}$$

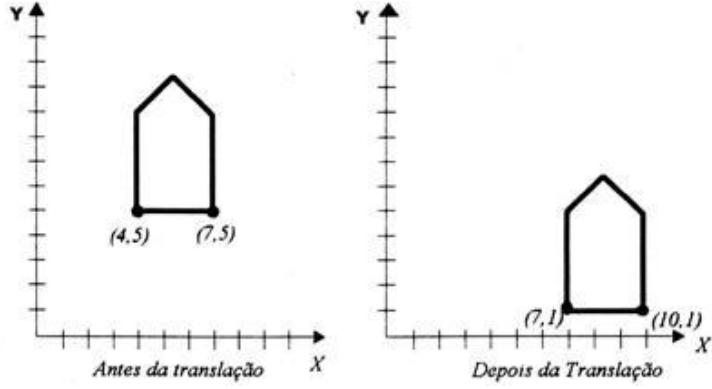


Figura 5.1: Translação de uma casa.

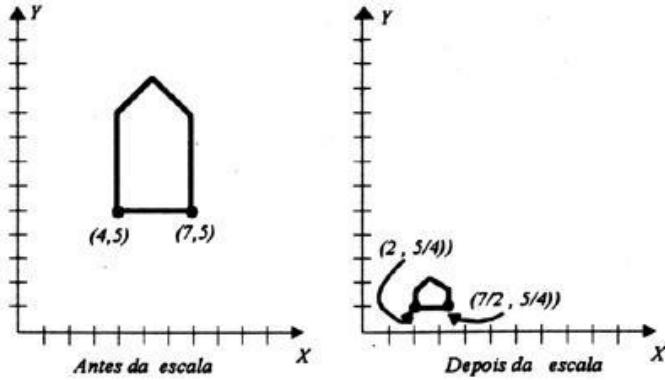


Figura 5.2: Mudança de escala de uma casa. Como a escala é não uniforme, sua proporção é alterada.

Na figura 5.2 a casa sofre uma escala de  $\frac{1}{2}$  em  $x$  e de  $\frac{1}{4}$  em  $y$ .

Observe que a escala é feita em relação a origem. Assim a casa fica menor e mais próxima da origem. Também as proporções da casa são alteradas, isto é, uma escala em que  $s_x$  é diferente de  $s_y$ . Porém ao utilizar-se escalas uniformes ( $s_x = s_y$ ) as proporções não são afetadas.

A Rotação de pontos através de um ângulo  $\theta$  qualquer também é feita a partir da origem. A rotação é definida matematicamente por:

$$\begin{aligned} x' &= x \cdot \cos(\theta) - y \cdot \sin(\theta) \\ y' &= x \cdot \sin(\theta) + y \cdot \cos(\theta) \end{aligned} \quad (5.6)$$

e matricialmente:

$$P' = R \cdot P \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad (5.7)$$

Escala e Rotação a partir de qualquer ponto serão tratadas posteriormente.

Os ângulos positivos são definidos quando a rotação é feita no sentido contrário aos do ponteiro do relógio, e ângulos negativos quando a rotação é feita no sentido dos ponteiros do relógio. Lembremos que  $\sin(-\theta) = -\sin(\theta)$  e que  $\cos(-\theta) = \cos(\theta)$ .

Observando a Figura 5.4 vemos que a rotação por  $\theta$  transforma  $P(x, y)$  em  $P'(x', y')$ . Como a rotação é em relação a origem, as distâncias de  $P$  e  $P'$  a origem,  $r$  na figura, são iguais. Assim, temos que:

$$\begin{aligned} x &= r \cdot \cos(\phi) \\ y &= r \cdot \sin(\phi) \end{aligned} \quad (5.8)$$

$$\begin{aligned} x' &= r \cdot \cos(\theta + \phi) = r \cdot \cos(\phi) \cdot \cos(\theta) - r \cdot \sin(\phi) \sin(\theta) \\ y' &= r \cdot \sin(\theta + \phi) = r \cos(\phi) \sin(\theta) + r \cdot \sin(\phi) \cdot \cos(\theta) \end{aligned} \quad (5.9)$$

Podemos obter a Equação 5.6, substituindo a Equação 5.8 na Equação 5.9.

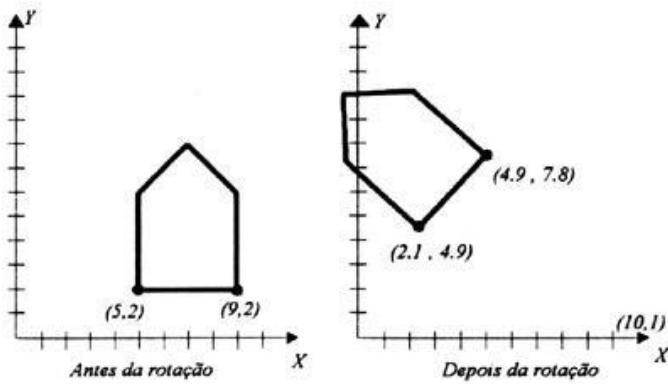


Figura 5.3: Esta figura mostra a rotação da casa por  $45^\circ$ . Da mesma forma que para a escala, a rotação também é feita em relação a origem.

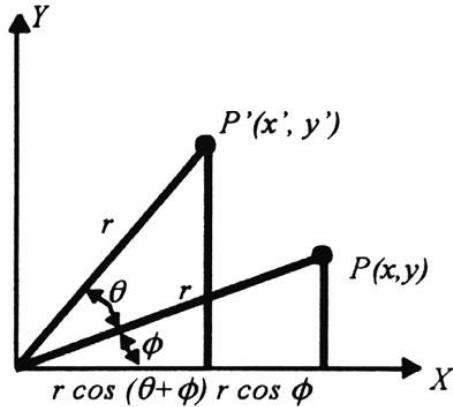


Figura 5.4: Derivando a equação de rotação.

## 5.2 Coordenadas Homogêneas e Matrizes de Transformação

Para a Translação, Escala e Rotação, as matrizes de transformação são respectivamente:

$$P' = T + P \quad (5.10)$$

$$P' = S \cdot P \quad (5.11)$$

$$P' = R \cdot P \quad (5.12)$$

Infelizmente, a translação é tratada de forma diferente (como uma soma) das outras - Rotação e Escala - que são tratadas através de multiplicações. Para que possamos combinar facilmente essas transformações, devemos poder tratar do mesmo modo todas as 3 transformações de uma forma consistente.

Se os pontos são expressos em Coordenadas Homogêneas, todas as 3 transformações podem ser tratadas como multiplicações.

Em coordenadas homogêneas, adicionamos uma terceira coordenada ao ponto. Logo, em vez de um ponto ser representado por um par de valores  $(x, y)$ , ele é representado por uma tripla  $(x, y, W)$ . Dizemos que 2 conjuntos de coordenadas homogêneas  $(x, y, W)$  e  $(x', y', W')$  representam o mesmo ponto se e somente se um é múltiplo do outro. Assim,  $(2, 3, 6)$  e  $(4, 6, 12)$  é o mesmo ponto representado por diferentes triplas. Isto é, cada ponto tem muitas diferentes representações homogêneas. Também, pelo menos uma das coordenadas homogêneas precisa ser diferente de zero, assim  $(0, 0, 0)$  não é permitido.

Se  $W$  é a coordenada não zero, podemos dividir  $(x, y, W)$  por ela, obtendo o mesmo ponto  $(\frac{x}{W}, \frac{y}{W}, 1)$ . Os números  $\frac{x}{W}$  e  $\frac{y}{W}$  são chamados de Coordenadas Cartesianas do ponto homogêneo. Usualmente, triplas representam pontos em um espaço 3D, mas agora estão sendo usadas para representar pontos em 2D. Observemos que, se tomarmos todas as triplas que representam o mesmo ponto, isto é, todas as triplas da forma  $(tx, ty, tW)$  com  $t$  diferente de 0, obtemos uma linha no espaço 3D. Se homogeneizamos o ponto (dividimos por  $W$ ), obtemos um ponto da forma  $(x, y, 1)$ . Logo, os pontos homogeneizados formam o plano definido pela equação  $W = 1$  no espaço  $(x, y, W)$ . A Figura 5.5, mostra esse relacionamento. Como agora os pontos são vetores de 3 elementos, as matrizes de transformações que multiplicam um ponto por outro também precisam ser de 3x3. A equação de Translação 5.1 para coordenadas homogêneas fica:

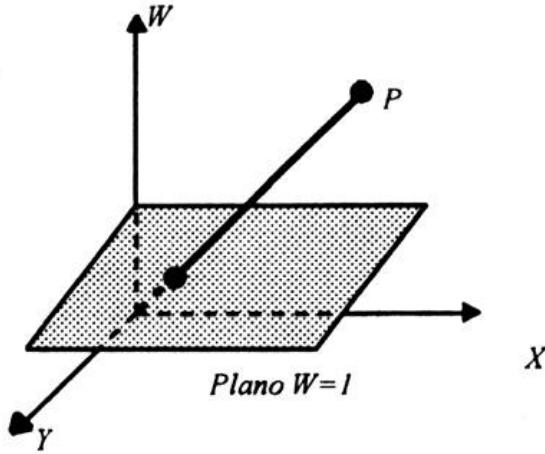


Figura 5.5: O espaço de Coordenadas Homogêneas  $XYW$ , com o plano  $W = 1$  e o ponto  $P(X, Y, W)$  projetado sobre o plano  $W = 1$ .

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (5.13)$$

Assim, a equação 5.13 pode ser representada como:

$$P' = T(d_x, d_y) \cdot P \quad (5.14)$$

onde:

$$T(d_x, d_y) = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \quad (5.15)$$

O que acontece se um ponto  $P$  é transladado por  $T(d_{x1}, d_{y1})$  para  $P'$  e então transladado por  $T(d_{x2}, d_{y2})$  para  $P''$ ?

Intuitivamente esperamos que essas translações sejam equivalentes a  $T(d_{x1} + d_{x2}, d_{y1} + d_{y2})$ . Ou seja, se:

$$P' = T(d_{x1}, d_{y1}) \cdot P \quad (5.16)$$

$$P'' = T(d_{x2}, d_{y2}) \cdot P' \quad (5.17)$$

$$(5.18)$$

Substituindo 5.17 em 5.18, temos:

$$P'' = T(d_{x2}, d_{y2}) \cdot [T(d_{x1}, d_{y1}) \cdot P] = [T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})] \cdot P \quad (5.19)$$

e a matriz produto das matrizes  $T(d_{x2}, d_{y2}) \cdot T(d_{x1}, d_{y1})$ , é:

$$\begin{bmatrix} 1 & 0 & d_{x2} \\ 0 & 1 & d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & d_{x1} \\ 0 & 1 & d_{y1} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_{x1} + d_{x2} \\ 0 & 1 & d_{y1} + d_{y2} \\ 0 & 0 & 1 \end{bmatrix} \quad (5.20)$$

e vemos que a Translação é aditiva.

Essa transformação expressa pelas duas transformações, é chamada de Transformação de Composição. Podemos mostrar similarmente que podemos realizar Composições também com a Escala e a Rotação. Assim, as equações de Escala 5.4, são representadas matricialmente:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (5.21)$$

e definindo,

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.22)$$

temos

$$P' = S(s_{x1}, s_{y1}) \cdot P \quad (5.23)$$

$$P'' = S(s_{x2}, s_{y2}) \cdot P' \quad (5.24)$$

e substituindo:

$$P'' = S(s_{x2}, s_{y2}) \cdot [S(s_{x1}, s_{y1}) \cdot P] = [S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})] \cdot P \quad (5.25)$$

A matriz produto  $S(s_{x2}, s_{y2}) \cdot S(s_{x1}, s_{y1})$  é:

$$\begin{bmatrix} s_{x2} & 0 & 0 \\ 0 & s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{x1} \cdot s_{x2} & 0 & 0 \\ 0 & s_{y1} \cdot s_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.26)$$

e temos que a Escala é multiplicativa.

Finalmente, verifiquemos como ficam as equações de rotação:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (5.27)$$

Definindo:

$$R(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.28)$$

temos que:

$$P' = R(\theta) \cdot P \quad (5.29)$$

Como exercício, mostre que duas rotações são aditivas!

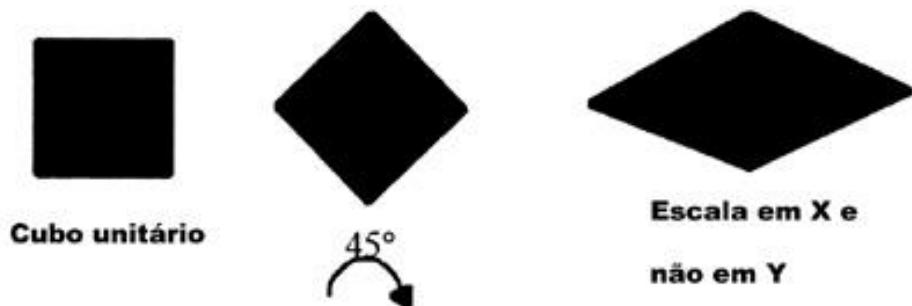


Figura 5.6: Um cubo unitário é rodados 45 graus, posteriormente escalado não uniformemente. Obtem-se dessa forma uma transformação afim.

Uma sequência arbitrária de rotações, translações e escalas, são chamadas de transformações afins. As Transformações Afins preservam paralelismo de linhas, mas não comprimentos e ângulos. A figura 5.6, mostra o resultado de se aplicar uma rotação de  $45^\circ$  e depois uma escala não uniforme a um cubo unitário.

Uma matriz de transformação da forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (5.30)$$

onde a submatriz  $2 \times 2$  do canto superior esquerdo é ortogonal, preserva ângulos e comprimentos. Estas transformações são chamadas de Corpo Rígido, porque o corpo do objeto não é distorcido de forma alguma.

## 5.3 Transformações 2D Adicionais: Espelhamento e Shearing

Duas transformações adicionais bastante usadas em CG são o Espelhamento (Reflexão) e Distorção (Shearing), discutidas a seguir.

### 5.3.1 Espelhamento (Mirror)

A transformação de reflexão, ou espelhamento, aplicada a um objeto, produz um objeto que é o espelho do original. No caso de uma reflexão 2D, o espelho é gerado relativamente a um eixo de reflexão rotacionando o objeto de  $180^\circ$  em torno do eixo de reflexão. Por exemplo, pode-se aplicar uma reflexão em torno da linha  $y = 0$  (o eixo  $x$ ) usando a seguinte matriz de transformação:

$$Mirror_v = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.31)$$

Esta transformação mantém as coordenadas  $x$  do objeto inalteradas, mas “inverte” os valores das coordenadas  $y$ , alterando a orientação espacial do objeto.

Analogamente, poderíamos definir uma reflexão em torno do eixo  $y$ , que “inverteia” as coordenadas  $x$  do objeto.

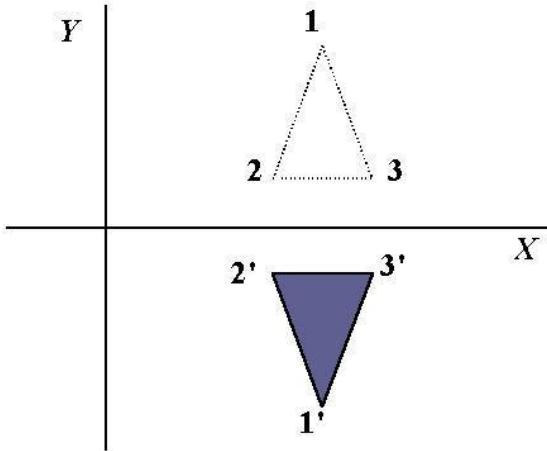


Figura 5.7: Reflexão de um objeto em torno do eixo  $x$ .

Podemos também definir uma reflexão em torno de um eixo perpendicular ao plano  $xy$  e passando (por exemplo) pela origem do sistema de coordenadas, “invertendo” nesse caso ambas as coordenadas  $x$  e  $y$ . A matriz de transformação é dada por:

$$Mirror_{xy} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.32)$$

A operação é ilustrada na figura 5.8. Observe que a matriz de reflexão acima é idêntica à matriz de rotação  $R(\theta)$  com  $\theta = 180^\circ$ , ou seja, estamos tão somente rotacionando o objeto de  $180^\circ$  em torno da origem. A operação pode ser generalizada para adotar qualquer ponto de reflexão localizado no plano  $xy$ , e o efeito é o mesmo que aplicar uma rotação de  $180^\circ$  em torno do ponto pivô da reflexão.

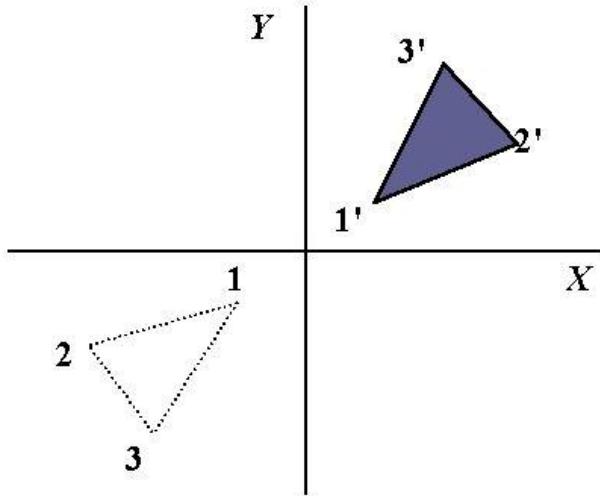


Figura 5.8: Reflexão de um objeto em torno de um eixo perpendicular ao plano  $xy$ , passando pela origem.

Podemos também adotar um eixo de reflexão arbitrário. A matriz de reflexão pode ser derivada pela concatenação de uma seqüência de matrizes de reflexão e de rotação. Por exemplo, se a reflexão for em torno da linha diagonal definida por  $y = x$ , a matriz de reflexão pode ser derivada da combinação da seguinte seqüência de transformações:

1. rotação de  $45^\circ$  na direção horária para fazer a linha  $y = x$  coincidir com o eixo  $x$ .
2. reflexão em torno do eixo  $x$ .
3. rotação de  $45^\circ$  na direção anti-horária para retomar a orientação original da linha  $y = x$ .

### 5.3.2 Shearing

*Shearing* (cisalhamento) é uma transformação que distorce o formato de um objeto - em geral, é aplicado um deslocamento aos valores das coordenadas  $x$  ou das coordenadas  $y$  do objeto. Uma distorção na direção  $x$  é produzida com a seguinte matriz de transformação:

$$Shear_x = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.33)$$

As coordenadas do objeto são transformadas da seguinte maneira:

$$\begin{aligned} x' &= x + sh_xy \\ y' &= y \end{aligned} \quad (5.34)$$

Qualquer número real pode ser usado como parâmetro. O resultado é que as coordenadas  $(x, y)$  são deslocadas horizontalmente segundo um valor proporcional à sua distância em do eixo  $x$ . Por exemplo, se  $sh_x$  é 2, um quadrado será transformado em um paralelogramo. Valores negativos deslocam as coordenadas para a esquerda.

Pode-se gerar distorções na direção relativamente a outros eixos de referência  $y = y_{ref}$  com a matriz:

$$Shear_{y_{ref}} = \begin{bmatrix} 1 & sh_x & -sh_x \cdot y_{ref} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.35)$$

que produz as seguintes transformações sobre as coordenadas:

$$\begin{aligned} x' &= x + sh_x(y - y_{ref}) \\ y' &= y \end{aligned} \quad (5.36)$$

Analogamente, pode-se aplicar uma distorção na direção y, relativa a uma linha  $x = x_{ref}$ , usando

$$Shear_{x_{ref}} = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & -sh_y \cdot x_{ref} \\ 0 & 0 & 1 \end{bmatrix} \quad (5.37)$$

que gera as seguintes transformações nas posições das coordenadas:

$$\begin{aligned} x' &= x \\ y' &= y + sh_y \cdot (x - x_{ref}). \end{aligned} \quad (5.38)$$

Esta transformação desloca as coordenadas de uma posição verticalmente segundo um fator proporcional à sua distância da linha de referência  $x = x_{ref}$ . Qualquer operação de distorção pode ser descrita como uma composição de transformações envolvendo uma seqüência de matrizes de rotação e escala.

## 5.4 Transformações entre sistemas de coordenadas

Já vimos que aplicações gráficas freqüentemente requerem a transformação de descrições de objetos de um sistema de coordenadas para outro. Muitas vezes, o objeto é descrito em um sistema de coordenadas não Cartesiano (como coordenadas polares ou elípticas), e precisa ser convertido para o sistema de coordenadas Cartesiano do dispositivo. Em aplicações de design e modelagem, objetos individuais são definidos em seu próprio sistema de coordenadas, e as coordenadas locais devem ser transformadas para posicionar os objetos no sistema de coordenadas global da cena. Neste texto vamos considerar especificamente transformações entre dois sistemas de coordenadas Cartesianos.

Para transformar descrições de um objeto dadas em um sistema de coordenadas  $xy$  para um sistema  $x'y'$  com origens em  $(0, 0)$  e  $(x_0, y_0)$ , com um ângulo de orientação  $\theta$  entre os eixos  $x$  e  $x'$ , precisamos determinar a transformação que superpõe os eixos  $xy$  aos eixos  $x'y'$ . Isso pode ser feito em 2 passos:

1. transladar o sistema  $x'y'$  de modo que sua origem coincida com a origem do sistema  $xy$ :  $T(-x_0, -y_0)$
2. Rotacionar o eixo  $x'$  de forma que ele coincida com o eixo  $x$ :  $R(-\theta)$

Concatenando as duas matrizes de transformação obtém-se a matriz de composição que descreve a transformação necessária:

$$M_{xy,x'y'} = R(-\theta) \cdot T(-x_0, -y_0) \quad (5.39)$$

Um método alternativo para estabelecer a orientação do segundo sistema de coordenadas consiste em especificar um vetor  $V$  na direção positiva do eixo  $y'$ , passando pela origem do sistema  $xy$ . O vetor  $V$  é especificado como um ponto no sistema de referência  $xy$  em relação à origem do sistema  $xy$ . Um vetor unitário que dá a direção de  $y'$  pode então ser obtido:

$$v = \frac{V}{|V|} = (v_x, v_y) \quad (5.40)$$

E obtemos o vetor unitário  $\vec{u}$  ao longo do eixo  $x'$  rotacionando  $v$  de  $90^\circ$  no sentido horário:

$$u = (v_y, -v_x) = (u_x, u_y) \quad (5.41)$$

Já vimos que os elementos de uma matriz de rotação podem ser expressos como elementos de um conjunto de vetores ortogonais. Portanto, a matriz de rotação a ser usada para rotacionar o sistema  $x'y'$  para que este coincida com o sistema  $xy$  é dada por:

$$R = \begin{bmatrix} u_x & u_y & 0 \\ v_x & v_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.42)$$

Por exemplo, se escolhermos que a nova orientação deve ser dada pelos vetores  $U = (0, 1)$  (eixo  $y$  positivo no sistema atual) e  $V = (-1, 0)$  (eixo  $x$  negativo no sistema atual), a matriz de rotação é dada por:

$$R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (5.43)$$

Essa matriz é equivalente à matriz de rotação para  $\theta = 90^\circ$

## 5.5 Composição de Transformações

Usaremos composição como uma combinação de matrizes de transformação  $R$ ,  $S$  e  $T$ . O propósito de compor-se transformações, é o ganho de eficiência que se obtém ao aplicar-se uma transformação composta a um ponto em vez de aplicar-lhe uma série de transformações, uma após a outra.

Considerando a rotação de um objeto em torno de um ponto arbitrário  $P_1$ . Mas sabemos apenas rotacionar um ponto em relação a origem. Assim, podemos dividir este problema de rotação em 3 problemas simples, ou seja: para rotacionar em relação a  $P_1$ , pode-se usar a seguinte sequência de transformações fundamentais:

1. Efetuar uma translação, levando  $P_1$  à origem.
2. Efetuar a Rotação desejada.
3. Efetuar uma Translação oposta à realizada em (1), levando  $P_1$  a posição anterior.

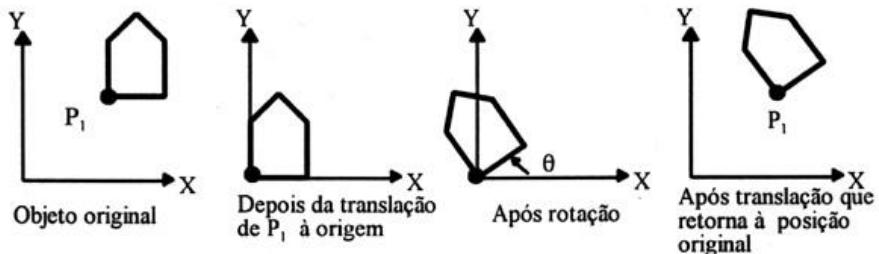


Figura 5.9: Rotação em relação ao Ponto  $P_1$ , por um ângulo  $\theta$ .

Esta sequência é ilustrada na figura 5.9, onde a casa é rotacionada em relação a  $P_1(x_1, y_1)$ . A primeira translação é por  $(-x_1, -y_1)$ , e a última translação (oposta a primeira) é por  $(x_1, y_1)$ . A transformação em sequência é:

$$T(x_1, y_1) \cdot R(\theta) \cdot T(x_1, y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & x_1(1 - \cos \theta) + y_1 \cdot \sin \theta \\ \sin \theta & \cos \theta & y_1(1 - \cos \theta) + x_1 \cdot \sin \theta \\ 0 & 0 & 1 \end{bmatrix} \quad (5.44)$$

Esse procedimento pode ser usado de forma similar para se efetuar a Escala de um objeto em relação a um ponto arbitrário  $P_1$ . Primeiramente o ponto  $P_1$  é transladado para a origem, então é feita a escala desejada, e então o ponto  $P_1$  é transladado de volta. Dessa forma, a transformação em seqüência é:

$$T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(x_1, y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & x_1(1 - s_x) \\ 0 & s_y & y_1(1 - s_y) \\ 0 & 0 & 1 \end{bmatrix} \quad (5.45)$$

Suponhamos que desejamos escalar, rotacionar e posicionar a casa mostrada na figura 5.10, com o ponto  $P_1(x_1, y_1)$  como o centro da rotação e da escala. A seqüência de transformações fica a seguinte:

1. Transladar  $P_1(x_1, y_1)$  para a origem;
  2. Efetuar a escala e a rotação desejadas;
  3. Efetuar a translação da origem para a nova posição  $P_2(x_2, y_2)$ , onde a casa deve ser posicionada.
- $T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1)$  é a matriz da Transformação composta.

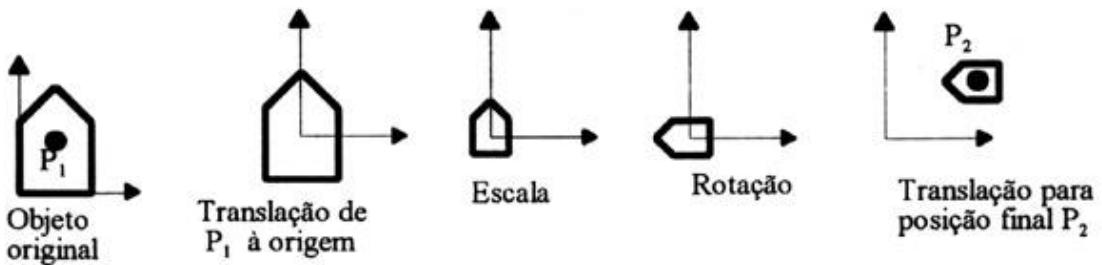


Figura 5.10: Escala e rotação de uma casa em relação ao ponto  $P_1$ .

Se  $M_1$  e  $M_2$  representam duas transformações fundamentais (translação, rotação ou escala), em que casos  $M_1 \cdot M_2 = M_2 \cdot M_1$ ? Isto é, quando suas matrizes de transformação comutam? Sabemos que geralmente a multiplicação de matrizes não é comutativa. Entretanto é fácil mostrar que nos seguintes casos especiais esta comutatividade existe (Tabela 5.1).

m1	M2
Translação	Translação
Escala	Escala
Rotação	Rotação
Escala(s, s)	Rotação

Tabela 5.1: Transformações comutativas.

Nestes casos não precisamos estar atentos a ordem de construção da matriz de transformação.

## 5.6 Transformação Janela - Porta de Visão (“Window-to-Viewport”)

Alguns sistemas gráficos permitem ao programador especificar primitivas de saída em um sistema de coordenadas em ponto-flutuante, chamado de sistema de coordenadas do Mundo Real, usando a unidade de medida que melhor se adeque ao seu programa de aplicação (*angstroms*, *microns*, metros, anos-luz, etc.). O termo Mundo (*World*) é usado para representar o ambiente interativamente criado ou apresentado para o usuário. Como as primitivas são especificadas em coordenadas do mundo, o pacote gráfico precisa mapear as coordenadas do mundo em coordenadas de tela.

Uma forma de se efetuar esta transformação é especificando uma região retangular em coordenadas do mundo, chamada de janela de coordenadas do mundo, e uma região retangular correspondente em coordenadas de tela, chamada de Porta de Visão (*Viewport*), em que a janela em coordenadas do mundo real é mapeada. A transformação que mapeia a janela na porta de visão é aplicada a todas as primitivas de saída em coordenadas do mundo, mapeando-as na tela. Este procedimento é ilustrado na Figura 5.11. Como pode ser visto pela figura, se a janela e a porta de visão não possuem a mesma razão largura-altura, uma escala não uniforme é realizada. Se o programa de aplicação altera a janela ou a porta de visão,

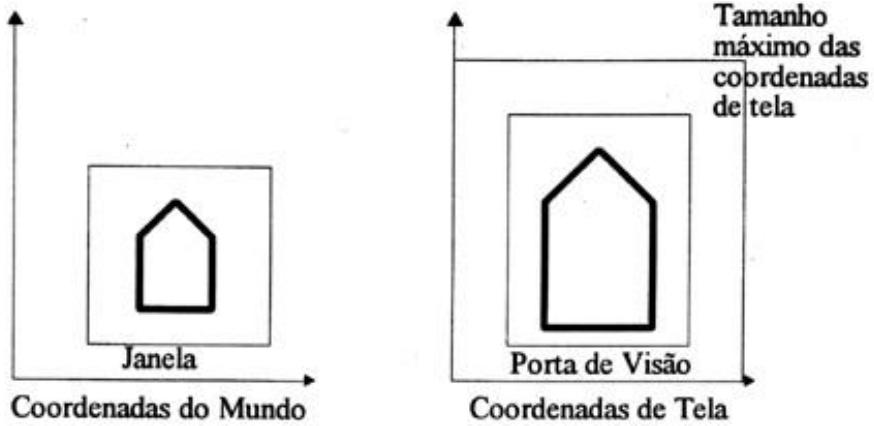


Figura 5.11: Janela em Coordenadas do mundo e porta de visão em coordenadas de tela.

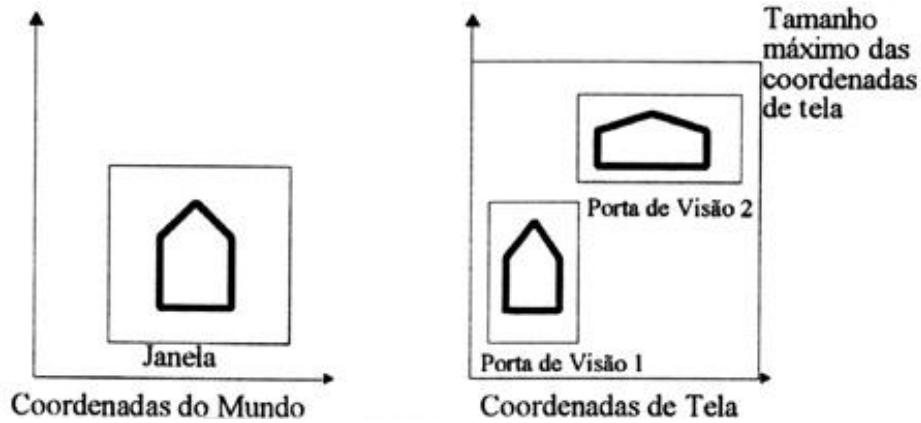


Figura 5.12: Duas portas de visão associadas a mesma janela.

então as novas primitivas de saída desenhadas sobre a tela precisam ser também atualizadas. Outro ponto é que pode-se ter múltiplas portas de visão de uma mesma janela, o que é mostrado na Figura 5.12.

Dada uma janela e uma porta de visão, qual é a matriz de transformação que mapeia a janela em coordenadas do mundo real, na porta de visão em coordenadas de tela? Esta matriz pode ser desenvolvida como uma transformação composta por 3 passos, como é sugerido na Figura 5.11. A janela especificada pelo seu canto inferior esquerdo e canto superior direito, é primeiramente transladada para a origem do mundo de coordenadas. A seguir o tamanho da janela é escalonado para ser igual ao tamanho da porta de visão. Finalmente, a translação é usada para posicionar a porta de visão. A matriz global  $M_{jp}$  é:

$$M_{jp} = T(u_{min}, v_{min}) \cdot S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \cdot T(-x_{min}, -y_{min}) = \\ \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix} = \\ \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & -x_{min} \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & -y_{min} \cdot \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix} \quad (5.46)$$

Multiplicando  $P = M_{jp} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ , temos:

$$P = \begin{bmatrix} (x - x_{min}) \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ (y - y_{min}) \cdot \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 1 \end{bmatrix}. \quad (5.47)$$

Muitos pacotes gráficos combinam a transformação janela - porta de visão com recorte (*clipping*) de primitivas gráficas de saída. A figura 10.14 ilustra esta técnica de recorte.

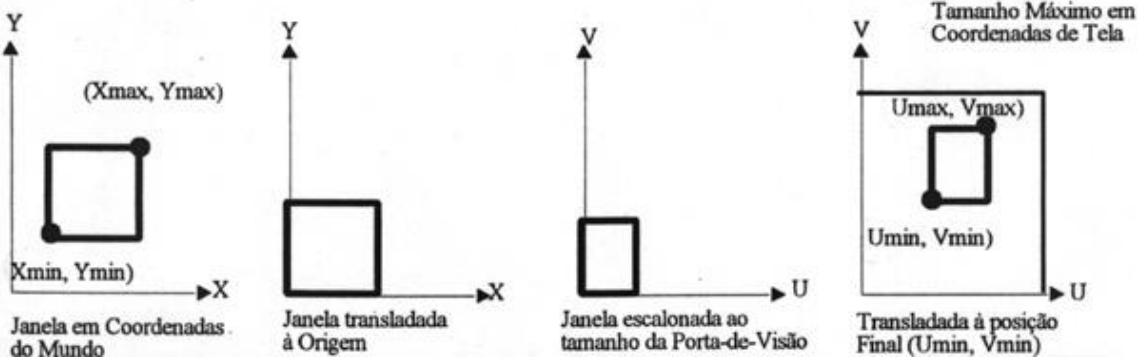


Figura 5.13: Os passos da transformação janela - porta de visão.

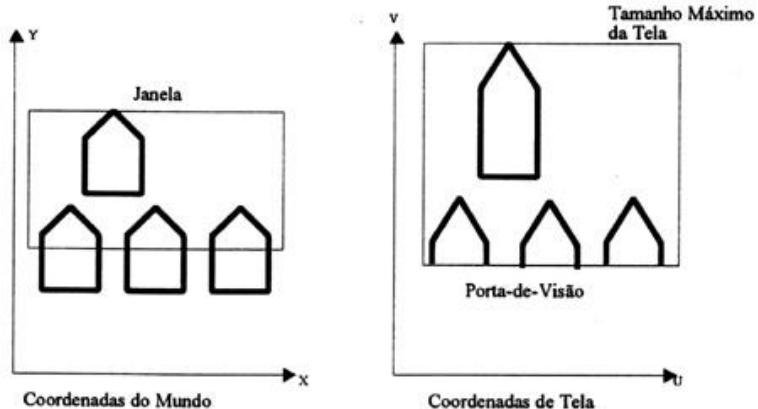


Figura 5.14: Primitivas gráficas de saída em coordenadas do mundo são recortadas pela janela. O seu interior é apresentado na tela (viewport).

## 5.7 Eficiência

Uma composição genérica de transformações  $R$ ,  $S$  e  $T$ , produz uma matriz da forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad (5.48)$$

A submatriz  $2 \times 2$  superior esquerda, é uma composição das matrizes de rotação e escala, enquanto  $t_x$  e  $t_y$  são obtidos por influência da translação. Ao calcularmos  $M \cdot P$  (um vetor de 3 elementos multiplicado por uma matriz  $3 \times 3$ ), verificamos que são necessárias 9 multiplicações e 6 adições. Mas, como a última linha da matriz é fixa, os cálculos efetivamente necessários são:

$$\begin{aligned} x' &= x.r_{11} + y.r_{12} + t_x \\ y' &= x.r_{21} + y.r_{22} + t_y \end{aligned} \quad (5.49)$$

o que reduz o processo a quatro multiplicações e duas adições, o que é um ganho significante em termos de eficiência. Especialmente se considerarmos que esta operação deve ser aplicada a centenas ou mesmo milhares de pontos por cena ou figura.

Outra área onde a eficiência é muito importante é na criação de sucessivas visões de um objeto, tal como por exemplo visões consecutivas de sistemas de moléculas, aviões, etc., onde cada visão é geralmente rotacionada a poucos graus da visão anterior. Se cada visão pode ser gerada e apresentada de forma bastante rápida (30 a 100 milisegundos cada uma), então parecerá ao observador que o objeto está sendo rotacionado continuamente (efeito de animação!). Para conseguirmos isto, devemos efetuar as transformações sobre os pontos do objeto o mais rapidamente possível. As equações de rotação (Equação 5.8) requerem 4 multiplicações e 2 adições. Mas se  $\theta$  é pequeno (poucos graus),  $\cos \theta$  é quase 1. Dessa aproximação, temos que:

$$x' = x - y \cdot \sin \theta$$

$$y' = x' \cdot \sin \theta + y \quad (5.50)$$

$$\sin -\theta = -\sin \theta \quad (5.51)$$

que requer 2 multiplicações e 2 adições (livrando de 2 multiplicações pode ser muito importante em computadores sem hardware específico para isto!).

A Equação 5.50, é somente uma aproximação para os valores corretos de  $x'$  e  $y'$  (existe um erro embutido!). Cada vez que a fórmula é aplicada a novos valores de  $x$  e  $y$  o erro torna-se um pouquinho maior. Uma aproximação melhor é usar  $x'$  em vez de  $x$  na segunda equação:

$$\begin{aligned} x' &= x = y \cdot \sin \theta \\ y' &= x' \cdot \sin \theta + y = (x - y \cdot \sin \theta) \cdot \sin \theta + y = x \cdot \sin \theta + y \cdot (1 - \sin^2 \theta) \end{aligned} \quad (5.52)$$

Esta é uma melhor aproximação que a equação 5.50, porque o determinante da matriz 2x2 correspondente é 1, o que significa que as áreas transformadas pela equação 5.52 não são alteradas.

Eixo de Rotação	Direção da Rotação Positiva
x	y para z
y	z para x
z	x para y

Tabela 5.2: Sentido de rotação.

## 5.8 Transformações em 3D

A capacidade para representar e visualizar um objeto em três dimensões é fundamental para a percepção de sua forma. Porém, em muitas situações é necessário mais do que isto, ou seja, poder “manusear” o objeto, movimentando-o através de rotações, translações e mesmo escala.

Assim, generalizando o que foi visto para TG em 2D, onde estas transformações foram representadas por matrizes 3x3 (utilizando coordenadas homogêneas), as TGs em 3D serão representadas por matrizes 4x4 também em coordenadas homogêneas.

Dessa forma, um ponto  $P$  de coodenadas  $(x, y, z)$  será representado por  $(x, y, z, W)$ . Padronizando (ou homogeneizando) o ponto, teremos se  $w$  for diferente de 0,  $(\frac{x}{W}, \frac{y}{W}, \frac{z}{W}, 1)$ .

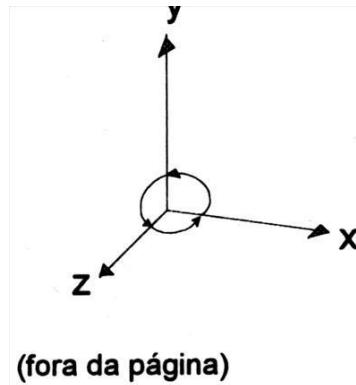


Figura 5.15: Sistema de Coordenadas dado pela Regra da Mão Direita.

O sistema de coordenadas para 3D utilizado será o da Regra da Mão Direita, com o eixo  $Z$  perpendicular ao papel e saindo em direção ao observador, como pode ser visto na figura 5.15.

O sentido positivo de uma rotação, é dado quando observando-se sobre um eixo positivo em direção à origem, uma rotação de  $90^\circ$  irá levar um eixo positivo em outro positivo. Ou conforme a Tabela 5.2.

A Regra da Mão Direita foi escolhida porque este é o padrão utilizado na matemática (lembre-se da definição do produto vetorial!).

A TRANSLAÇÃO em 3D pode ser vista como simplesmente uma extenção a partir da translação 2D, ou seja:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.53)$$

Assim, a Equação 5.53 pode ser representada também como:

$$P' = T(d_x, d_y, d_z) \cdot P \quad (5.54)$$

Similarmente a ESCALA em 3D, fica:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.55)$$

$$P' = S(s_x, s_y, s_z) \cdot P \quad (5.56)$$

Finalmente, verifiquemos como ficam as equações de ROTAÇÃO em 3D, pois as rotações podem ser efetuadas em relação a qualquer um dos três eixos.

A equação de rotação em 2D é justamente uma rotação em torno do eixo  $z$  em 3D, a qual é:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.57)$$

A matriz de rotação em relação ao eixo  $x$  é:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.58)$$

A matriz de rotação em relação ao eixo  $y$  é:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (5.59)$$

Para a rotação também temos que:

$$P' = R(\theta) \cdot P \quad (5.60)$$

OBS: Os vetores compostos pelas linhas e colunas da submatriz 3x3 do canto superior esquerdo de  $R_{x,y,z}(\theta)$  são mutuamente perpendiculares, e a submatriz possui determinante igual a 1, o que significa que as três matrizes são chamadas de **Ortogonais Especiais**. Além disso, uma sequência arbitrária de rotações é também ortogonal especial. Deve-se lembrar que as transformações ortogonais preservam distâncias e ângulos.

Todas estas matrizes de transformação ( $T, SeR$ ) possuem inversas. A inversa de  $T$  é obtida simplesmente negando-se  $d_x$ ,  $d_y$  e  $d_z$ . Para a matriz  $S$ , basta substituir  $s_x$ ,  $s_y$  e  $s_z$  por seus valores inversos. Para cada uma das matrizes de rotação  $R$ , basta negar o ângulo de rotação. Além disso, para uma matriz ortogonal  $B$ , sua inversa ( $B^{-1}$ ) é a sua matriz transposta, ou seja  $B^{-1} = B^T$ .

Uma sequência arbitrária de transformações de translação, escala e rotação podem ser multiplicadas juntas, resultando uma matriz da seguinte forma:

$$M = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.61)$$

Como no caso de 2D, a submatriz 3x3 do canto superior esquerdo  $R$ , agrega as transformações de escala e rotação, enquanto a última coluna à direita  $T$  agrega as translações. Para obter-se um pouco mais de eficiência (ganho computacional) pode-se executar a transformação explicitamente como:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + T \quad (5.62)$$

### 5.8.1 Composição de Transformações em 3D

A composição de transformações em 3D pode ser entendida mais facilmente através do exemplo indicado na figura 5.16. O objetivo é transformar os segmentos de reta  $P_1P_2$  e  $P_1P_3$  da posição inicial em (a) para a posição final em (b). Assim o ponto  $P_1$  deve ser transladado para a origem,  $P_1P_2$  deverá ficar sobre o eixo  $z$  positivo, e  $P_1P_3$  deverá ficar no plano positivo de  $yz$ . Além disso, os comprimentos das linhas não devem ser alterados.

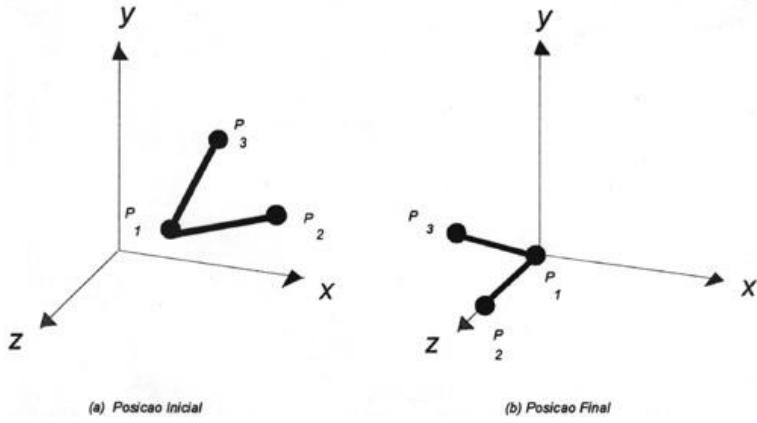


Figura 5.16: Transformando  $P_1$ ,  $P_2$  e  $P_3$  da posição inicial em (a) para a posição final em (b).

Uma primeira maneira de se obter a transformação desejada é através da composição das primitivas de transformação  $T$ ,  $R_x$ ,  $R_y$  e  $R_z$ .

Subdividindo o problema, teremos os seguintes quatro passos:

1. Transladar  $P_1$  para a origem.
2. Rotacionar o segmento  $P_1P_2$  em relação ao eixo  $y$ , de forma que ele ( $P_1P_2$ ) fique no plano  $yz$ .
3. Rotacionar o segmento  $P_1P_2$  em relação ao eixo  $x$ , de forma que ele ( $P_1P_2$ ) fique sobre o eixo  $z$ .
4. Rotacionar o segmento  $P_1P_3$  em relação ao eixo  $z$ , de forma que ele ( $P_1P_3$ ) fique no plano  $yz$ .

#### Primeiro Passo: Transladar $P_1$ para a Origem

A equação de translação é:

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5.63)$$

Aplicando  $T$  a  $P_1$ ,  $P_2$  e  $P_3$ , temos:

$$\begin{aligned} P'_1 &= T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\ P'_2 &= T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix} \\ P'_3 &= T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix} \end{aligned} \quad (5.64)$$

#### Segundo Passo: Rotacionar em Relação ao eixo Y

A Figura 5.17 (o ângulo  $\theta$  indica a direção positiva de rotação em relação a  $y$ . O ângulo utilizado na realidade é  $-(90^\circ - \theta)$ ) mostra  $P_1P_2$  após o primeiro passo, bem como a projeção de  $P_1P_2$  no plano  $xz$ . O ângulo de rotação é  $-(90^\circ - \theta) = \theta - 90^\circ$ . Então:

$$\begin{aligned} \sin(\theta - 90^\circ) &= -\cos \theta = -\frac{x'_2}{D_1} = -\frac{x_2 - x_1}{D_1} \\ \cos(\theta - 90^\circ) &= \sin \theta = -\frac{z'_2}{D_1} = -\frac{z_2 - z_1}{D_1} \end{aligned} \quad (5.65)$$

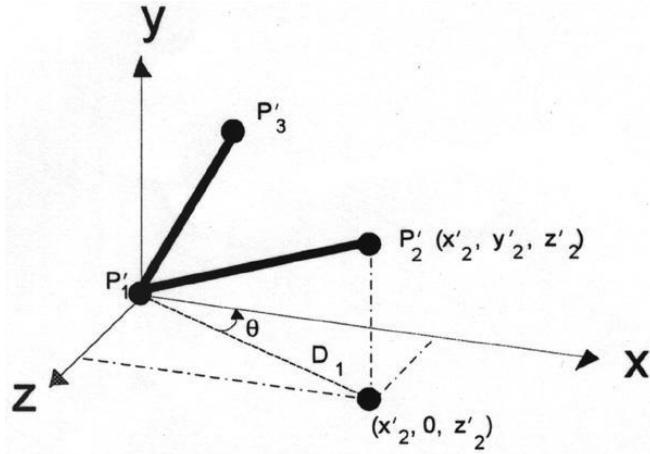


Figura 5.17: Rotação dos pontos  $P_1$ ,  $P_2$  e  $P_3$ .

onde

$$D_1 = \sqrt{(z'_2)^2 + (x'_2)^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2} \quad (5.66)$$

Então substituindo estes valores na Equação 5.59, temos:

$$P''_2 = R_y(\theta - 90^\circ) \cdot P'_2 = \begin{bmatrix} 0 \\ y_2 - y_1 \\ D_1 \\ 1 \end{bmatrix} \quad (5.67)$$

Como era esperado, a componente  $x$  de  $P''_2$  é zero, e  $z$  possui comprimento  $D_1$ .

#### Terceiro Passo: Rotacionar em Relação ao eixo X

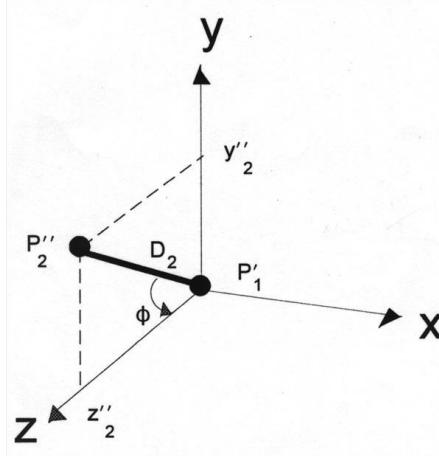


Figura 5.18: Rotação em relação ao eixo x.  $P_1$  e  $P_2$  de comprimento  $D_2$  é rotacionado em direção ao eixo z, pelo ângulo positivo  $\phi$ .

A Figura 5.18 mostra  $P_1P_2$  após o segundo passo (o segmento de reta  $P_1P_3$  não é mostrado porque não é utilizado para determinar os ângulos de rotação. Ambos os segmentos são rotacionados por  $R_x(\phi)$ .) O ângulo de rotação é  $\phi$ , e

$$\begin{aligned} \cos \phi &= \frac{z''_2}{D_2} \\ \sin \phi &= \frac{y''_2}{D_2} \end{aligned} \quad (5.68)$$

onde  $D_2 = |P''_1 P''_2|$  é o comprimento do segmento de reta  $P''_1 P''_2$ . Como as translações e rotações preservam o comprimento, o comprimento de  $P''_1 P''_2$  é o mesmo de  $P_1 P_2$ , dessa forma

$$D_2 = |P''_1 P''_2| = |P_1 P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \quad (5.69)$$

O resultado da rotação no terceiro passo é:

$$P'''_2 = R_x(\phi) \cdot P''_2 = R_x(\phi) \cdot R_y(\theta - 90^\circ) \cdot P'_2 = R_x(\phi) \cdot R_y(\theta - 90^\circ) \cdot T \cdot P_2 = \begin{bmatrix} 0 \\ 0 \\ |P_1 P_2| \\ 1 \end{bmatrix} \quad (5.70)$$

Dessa forma, agora  $P_1 P_2$  agora está sobre (coincidindo) o eixo  $z$  positivo.

#### Quarto Passo: Rotacionar em Relação ao eixo Z

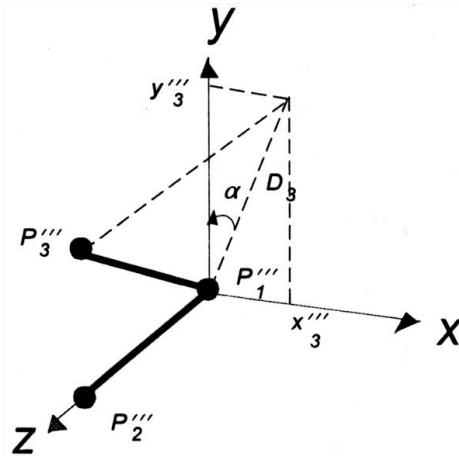


Figura 5.19: Rotação em relação ao eixo  $z$ .

A figura 5.19 mostra  $P_1 P_2$  e  $P_1 P_3$  após o terceiro passo, com  $P'''_2$  sobre o eixo  $z$  e  $P'''_3$  em

$$P'''_3 = \begin{bmatrix} x'''_3 \\ y'''_3 \\ z'''_3 \\ 1 \end{bmatrix} = R_x(\phi) \cdot R_y(\theta - 90^\circ) \cdot T(-x_1, -y_1, -z_1) \cdot P_3 \quad (5.71)$$

A rotação é dada pelo ângulo positivo  $\alpha$ , e

$$\begin{aligned} \cos \alpha &= \frac{y'''_3}{D_3} \\ \sin \alpha &= \frac{x'''_3}{D_3} \\ D_3 &= \sqrt{x'''_3^2 + y'''_3^2} \end{aligned} \quad (5.72)$$

$P'_1 P'_3$  é trazido para o plano  $yz$ .

Dessa forma, após o quarto passo o resultado é alcançado como visto na Figura 5.16(b).

A matriz de composição M é a transformação necessária à composição solicitada na Figura 5.16.

$$M = R_x(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90^\circ) \cdot T(-x_1, -y_1, -z_1) = R \cdot T \quad (5.73)$$

**Exercício:** aplique a matriz de composição a cada um dos pontos  $P_1$ ,  $P_2$  e  $P_3$  para verificar que  $P_1$  é transformado para a origem,  $P_2$  está sobre o eixo  $z$  positivo e  $P_3$  no plano  $yz$  positivo.

Utilizando as propriedades de matrizes ortogonais (ver Equações 5.61, 5.62 e a observação da seção 5.8), podemos encontrar uma segunda forma de encontrar a matriz de transformações. Relembremos que os vetores unitários (de comprimento igual a 1) de  $R$  rotacionam em relação aos eixos principais.

Assim,

$$R = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} \\ r_{1y} & r_{2y} & r_{3y} \\ r_{1z} & r_{2z} & r_{3z} \end{bmatrix} \quad (5.74)$$

Colocando  $R_z$  como o vetor unitário ao longo de  $P_1P_2$  que irá rotacionar em direção ao eixo  $z$  positivo,

$$R_z = [r_{1z} \ r_{2z} \ r_{3z}]^T = \frac{P_1P_2}{|P_1P_2|} \quad (5.75)$$

O vetor unitário  $R_x$  é perpendicular ao plano de  $P_1$ ,  $P_2$  e  $P_3$  e irá rotacionar em direção ao eixo  $x$  positivo, dessa forma  $R_x$  deveria ser o produto vetorial de dois vetores do plano (relembre-se das propriedades de Produto Vetorial vistos em Geometria Analítica !):

$$R_x = [r_{1x} \ r_{2x} \ r_{3x}]^T = \frac{P_1P_3 \times P_1P_2}{|P_1P_3 \times P_1P_2|} \quad (5.76)$$

e finalmente,

$$R_y = [r_{1y} \ r_{2y} \ r_{3y}]^T = R_z \times R_x \quad (5.77)$$

irá rotacionar em direção ao eixo  $y$  positivo. A matriz de composição é dada por

$$\begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot T(-x_1, -y_1, -z_1) = R \cdot T \quad (5.78)$$

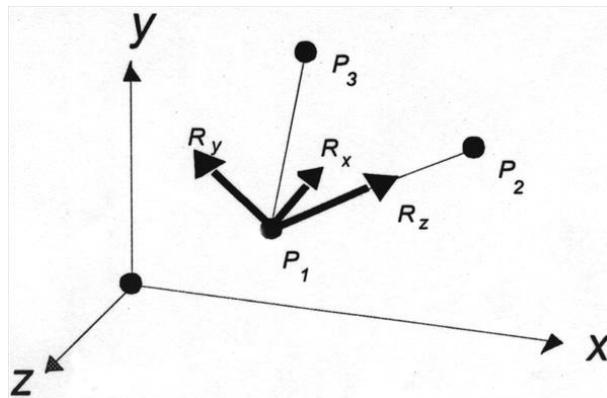


Figura 5.20: Os vetores unitários  $R_x$ ,  $R_y$  e  $R_z$ , os quais são transformados nos eixos principais.

e  $R$  e  $T$  são como em 5.78. A Figura 5.20 mostra os vetores individuais  $R_x$ ,  $R_y$  e  $R_z$ .

# Capítulo 6

## Observação de Cenas 3D

### 6.1 Pipeline de observação (“viewing pipeline”)

Após a criação de cenas e objetos tridimensionais o próximo passo é efetuar a sua apresentação. Ao gerar imagens de cenas 3D em computação gráfica, fazemos uma analogia com uma câmera fotográfica: imaginamos um observador que, posicionado em um ponto de observação, vê a cena através das lentes de uma câmera virtual que pode ser posicionada de forma a obter a imagem desejada da cena (o “fotógrafo” pode definir a posição da câmera, sua orientação e ponto focal). A fotografia que se obtém com uma câmera real é uma projeção da cena em um plano de imagem 2D (o filme na câmera). Da mesma forma que no mundo real, a imagem que se obtém da cena sintética depende de vários fatores que determinam como esta é projetada em um plano para formar a imagem 2D exibida (por exemplo) no monitor. Estes parâmetros incluem a **posição** da câmera, sua **orientação** e **ponto focal**, o **tipo de projeção** efetuada e a posição dos “**planos de recorte**” (*clipping planes*).

A posição e o ponto focal da câmera definem, respectivamente, aonde a câmera está e para onde ela está apontando. O vetor que vai da posição da câmera ao ponto focal é denominado **direção de projeção**. O plano de imagem, que é o plano no qual a cena será projetada, está posicionado no ponto focal e, tipicamente, é perpendicular ao vetor direção de projeção (Figura 6.1). A orientação da câmera é controlada pela sua posição, seu ponto focal e por um outro vetor denominado *view up*. Estes três parâmetros definem completamente a câmera.

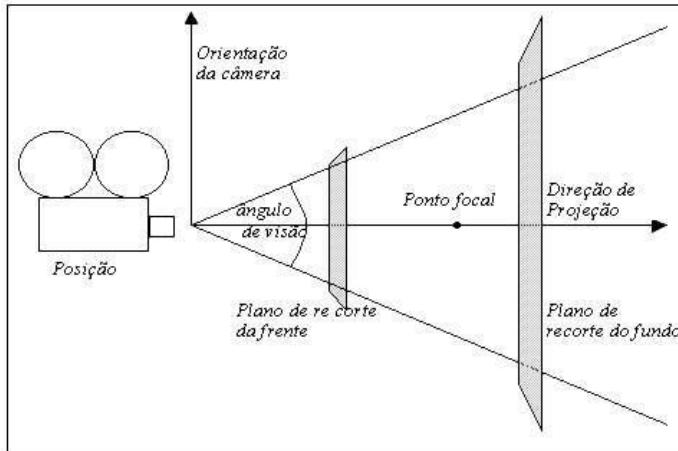


Figura 6.1: Atributos da câmera [Schröeder, 1998].

O método de projeção controla como os atores da cena são mapeados no plano de imagem: na *projeção ortográfica*, ou projeção paralela, o processo de mapeamento é paralelo: assume-se que todos os raios de luz que atingem a câmera são paralelos ao vetor de projeção. Na *projeção perspectiva*, todos os raios convergem para um ponto comum, denominado ponto de observação, ou centro da projeção. Nesse caso, deve-se determinar o **ângulo de visão** da câmera.

Os planos de recorte anterior e posterior interceptam o vetor de projeção e são, geralmente, perpendiculares a ele. Os planos de recorte são usados para eliminar atores que estão muito próximos ou muito distantes da câmera, de forma que apenas os atores que estão na área englobada pelos planos de recorte são potencialmente visíveis. Em geral, os planos de recorte são perpendiculares à direção

de projeção. Suas posições podem ser especificadas usando o chamado intervalo de recorte (“*clipping range*”) da câmera: a localização dos planos é dada a partir da posição da câmera, ao longo da direção de projeção. A posição do plano anterior dá o valor mínimo do intervalo, e a posição do plano posterior dá o valor máximo.

A câmera pode ser manipulada diretamente através dos parâmetros acima descritos, mas existem algumas operações comuns que facilitam a tarefa de quem gera a imagem, como ilustradas nas figuras 6.2 e 6.3:

- **Azimuth** - rotaciona a posição da câmera ao redor do seu vetor view up, com centro no ponto focal
- **Elevation** - rotaciona a posição da câmera ao redor do vetor dado pelo produto vetorial entre o vetor view up e o vetor direção de projeção, com centro no ponto focal.
- **Roll (Twist)** - rotaciona o vetor view up em torno do vetor normal ao plano de projeção.
- **Yaw** - rotaciona o ponto focal da câmera em torno do vetor view up, com centro na posição da câmera.
- **Pitch** - rotaciona o ponto focal ao redor do vetor dado pelo produto vetorial entre o vetor view up e o vetor direção de projeção, com centro na posição da câmera.
- **Dolly** - (in, out) move a posição da câmera ao longo da direção de projeção (mais próximo ou mais distante do ponto focal).
- **Zoom** - altera o ângulo de visão da câmera, de modo que uma região maior ou menor da cena fique na região que é potencialmente visível.

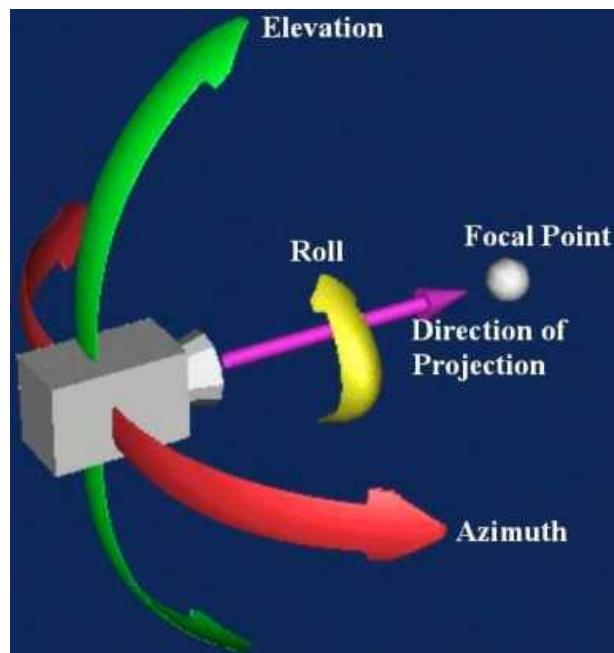


Figura 6.2: Movimentos de câmera [Schröeder, 1998].

Em resumo, gerar uma vista de um objeto em três dimensões é similar a fotografar o objeto. Podemos caminhar na cena e fotografar de qualquer ângulo, de várias distâncias e com diferentes orientações da câmera. A região da cena que aparece no visor da máquina é o que vai ser projetado na superfície do filme. O tipo e o tamanho da lente da câmera determinam quais partes da cena aparecem na foto final. Estas idéias estão incorporadas nas APIs gráficas 3D, de forma que vistas de uma cena podem ser geradas a partir da definição de parâmetros como posição espacial, orientação e ângulo de abertura da “câmera virtual”.

## 6.2 Coordenadas de Observação

### 6.2.1 Especificação do sistema de coordenadas de observação

Escolhemos uma “vista” específica para uma cena estabelecemos primeiramente um sistema de coordenadas de observação (*Viewing Coordinate System* - VCS, ou *View reference Coordinate System*). Um plano

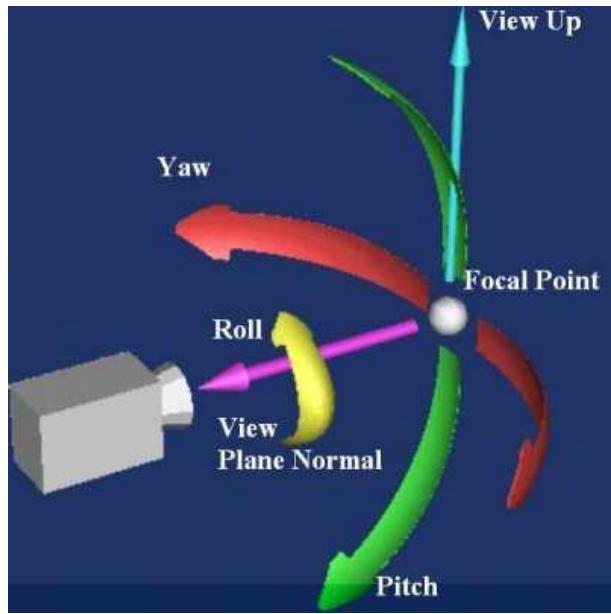


Figura 6.3: Movimentos de câmera [Schröeder, 1998].

de observação, ou plano de projeção, é definido, que é perpendicular ao eixo  $z$  do sistema de coordenadas de visualização VCS (esse plano faz o papel do plano do filme na câmera virtual). Em seguida, é preciso transformar a descrição da cena dada no sistema de coordenadas do mundo (WC - *World Coordinates*) para uma descrição no sistema de coordenadas de observação VCS.

Mas, como determinamos o VCS? Inicialmente, estabelecemos a origem do sistema. Para isso, usamos um ponto inicialmente definido no WC, o *View Reference Point* - VRP. Podemos pensar nesse ponto como sendo a posição da câmera. Em seguida, selecionamos a direção positiva para o eixo  $z$  do VCS (vamos denotar esse eixo por  $z_v$ ), bem como a orientação do plano de observação, especificando o vetor normal ao plano de observação,  $N$ . Finalmente, definimos a direção do eixo  $y$  do VCS ( $y_v$ ) especificando-se um vetor  $V$  (*View Up*). Obviamente,  $V$  deve ser perpendicular a  $N$ . Pode ser difícil para o programador determinar  $V$  exatamente de forma a satisfazer essa condição. Tipicamente, as transformações de observação implementadas nos pacotes gráficos ajustam  $V$  projetando-o em um plano que é perpendicular ao vetor  $N$ : dessa forma, o programador pode escolher qualquer direção conveniente para o vetor  $V$ , desde que não seja paralela à  $N$ . A partir dos vetores  $N$  e  $V$ , o pacote gráfico pode computar um terceiro vetor  $U$ , perpendicular a ambos, que define a direção para o eixo  $x_v$  do VCS (veja Figuras 6.4 e 6.5).

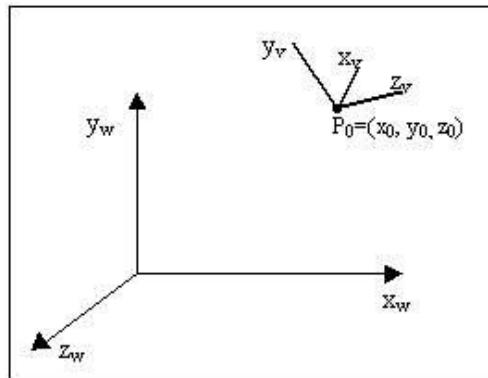


Figura 6.4: Um VCS baseado na regra da mão direita com  $x_v$ ,  $y_v$  e  $z_v$ .relativos ao sistema de coordenadas do mundo [Hearn and Baker, 1994].

### 6.2.2 Transformação do sistema de coordenadas do mundo para o sistema de coordenadas de observação

Antes que as descrições do objeto (em WC) possam ser projetadas no plano de observação, elas precisam ser transformadas para as coordenadas do VCS. A conversão das coordenadas de WC para VCS é

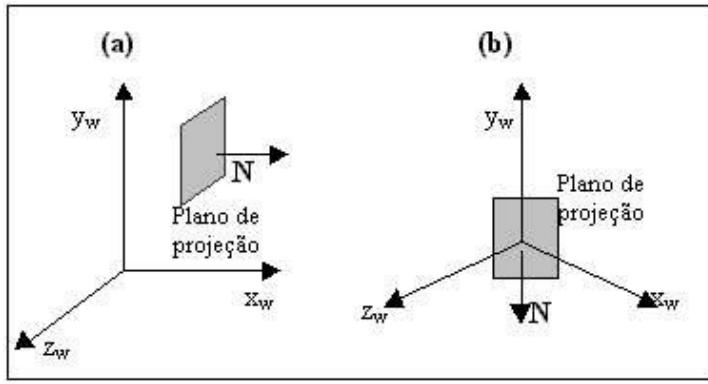


Figura 6.5: Orientações do plano de observação conforme o vetor normal. O vetor  $(1, 0, 0)$  dá a orientação em (a) e  $(1, 0, 1)$ , a orientação em (b) [Hearn and Baker, 1994].

equivalente a uma transformação que superpõe o VCS ao WC, usando as transformações geométricas já vistas. A seqüência de transformações é dada por:

1. translada o VRP para a origem do WC .
2. Aplica rotações para alinhar os eixos do VCS  $(x_v, y_v, z_v)$  com os eixos do WC  $(x_w, y_w, z_w)$ , respectivamente.

Se as coordenadas do ponto VRP em WC são dadas por  $(x_0, y_0, z_0)$ , a matriz de translação é dada por  $T(-x_0, -y_0, -z_0)$ . A seqüência de rotações pode exigir até três rotações em torno dos eixos coordenados, dependendo da direção escolhida para  $N$ . Em geral, se  $N$  não está alinhado com qualquer um dos eixos, a superposição de ambos os sistemas vai exigir uma seqüência  $R_z \cdot R_y \cdot R_x$ . Isto é, primeiro rotacionamos em torno do eixo  $x_w$  para trazer o eixo  $z_v$  para o plano  $x_w z_w$ . Em seguida, rotacionamos em torno do eixo  $y_w$  para alinhar os eixos  $z_w$  e  $z_v$ . A rotação final em torno do eixo  $z_w$  é para alinhar os eixos  $y_w$  e  $y_v$ . A matriz de transformação composta é então aplicada a todas as descrições de objetos no WC para transformá-las para VCS.

Uma forma alternativa de gerar a matriz que descreve as transformações de rotação necessárias é calcular os vetores unitários  $\vec{u}$ ,  $\vec{v}$  e  $\vec{n}$  do sistema de coordenadas de observação, e formar a matriz de composição diretamente, uma vez que esses vetores são ortogonais entre si e definem uma matriz ortogonal. Dados os vetores  $N$  e  $V$ , os vetores unitários podem ser calculados da seguinte forma:

$$\vec{n} = \frac{N}{|N|} = (n_1, n_2, n_3) \quad (6.1)$$

$$\vec{u} = \frac{V \times N}{|V \times N|} = (u_1, u_2, u_3) \quad (6.2)$$

$$\vec{v} = n \times u = (v_1, v_2, v_3) \quad (6.3)$$

Este método também ajusta automaticamente a direção de  $V$  de forma que  $\vec{v}$  seja perpendicular a  $\vec{n}$ . A matriz de rotação composta para a transformação de visualização é então dada por:

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.4)$$

A matriz de transformação completa a ser aplicada aos objetos da cena para transferi-los para o sistema de coordenadas de observação é dada por:

$$M_{wc,vc} = R \cdot T = \begin{bmatrix} u_1 & u_2 & u_3 & -x_0 \\ v_1 & v_2 & v_3 & -y_0 \\ n_1 & n_2 & n_3 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.5)$$

## 6.3 Projeções

Tendo os objetos descritos no sistema de coordenadas de observação, pode-se projetar os objetos tridimensional no plano de observação. No processo de geração da imagem de uma cena deparamos com o problema de apresentar uma entidade tridimensional num meio bidimensional (2D), que é a tela do monitor de vídeo. Esse processo, denominado de Projeção, tem sido tratado exaustivamente por desenhistas, artistas e arquitetos que buscaram técnicas e artifícios para sistematizá-lo e solucioná-lo. A figura 6.6 apresenta uma taxonomia dos métodos de projeção.

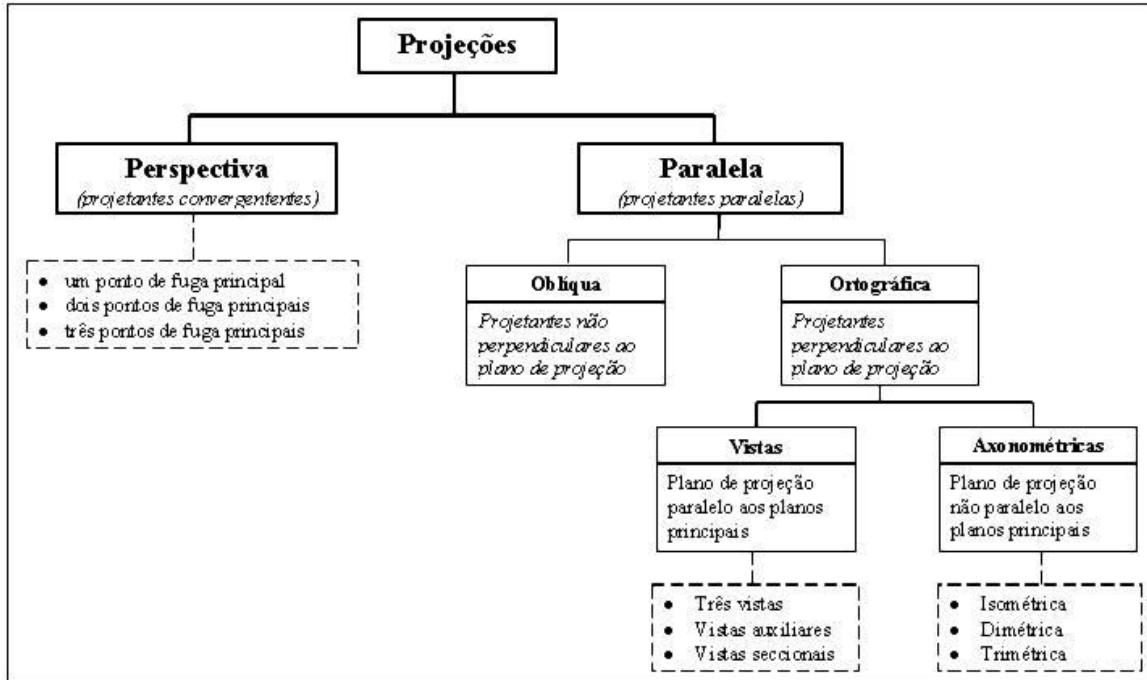


Figura 6.6: Taxonomia de projeções [Plastock and Kalley, 1999].

### 6.3.1 Projeção Perspectiva

As técnicas utilizadas em projeção perspectiva são derivadas daquelas utilizadas pelos artistas e desenhistas profissionais. Pode-se dizer que o olho do observador coloca-se no centro de projeção, e o plano que deve conter o objeto ou cena projetada transforma-se no plano de projeção. Dois segmentos de reta que saem do centro de projeção e atingem o objeto projetado no plano de projeção, são chamadas de projetantes (veja a Figura 6.7)

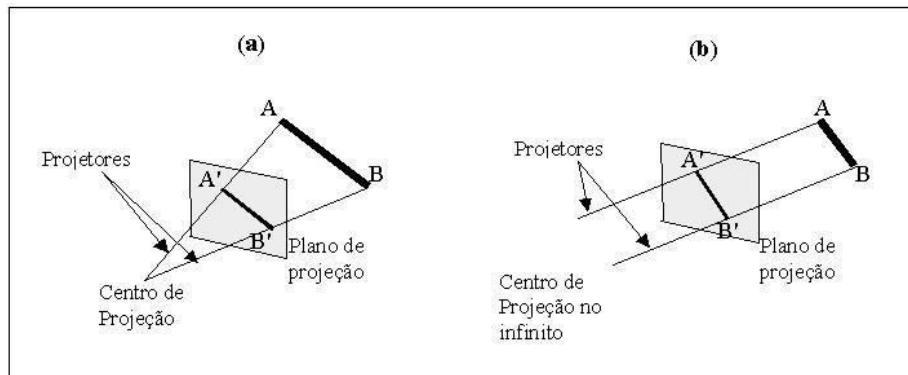


Figura 6.7: Linha AB e sua projeção A'B': (a) perspectiva; (b) ortogonal.

Os desenhos em perspectiva são caracterizados pelo encurtamento perspectivo e pelos pontos de fuga. O encurtamento perspectivo, é a ilusão de que os objetos e comprimentos são cada vez menores à medida que sua distância ao centro de projeção aumenta. Tem-se também a ilusão de que conjuntos de linhas paralelas que não são paralelas ao plano de projeção, convergem para um ponto de fuga. Denominam-se

pontos de fuga principais, quando dá-se a aparência de haver uma intersecção entre um conjunto de retas paralelas com um dos eixos principais  $O_x$ ,  $O_y$  ou  $O_z$ . O número de pontos de fuga principais é determinado pelo número de eixos principais interceptados pelo plano de projeção. Por exemplo: se o plano de projeção intercepta apenas o eixo  $z$  (então é perpendicular ao eixo  $z$ ), somente o eixo  $z$  possui um ponto de fuga principal, pois linhas paralelas aos eixos  $x$  e  $y$ , são também paralelas ao plano de projeção, e dessa forma não ocorre a ilusão de convergência.

Projeções Perspectivas são categorizadas pelo seu número de pontos de fuga principais, ou seja o número de eixos que o plano de projeção intercepta. A Figura 6.8 mostra 2 projeções perspectivas (com um ponto de fuga) distintas de um cubo. Está claro que possui apenas um ponto de fuga? Somente as linhas paralelas ao eixo  $z$  convergem, e as linhas paralelas aos eixos  $x$  e  $y$  continuam paralelas!

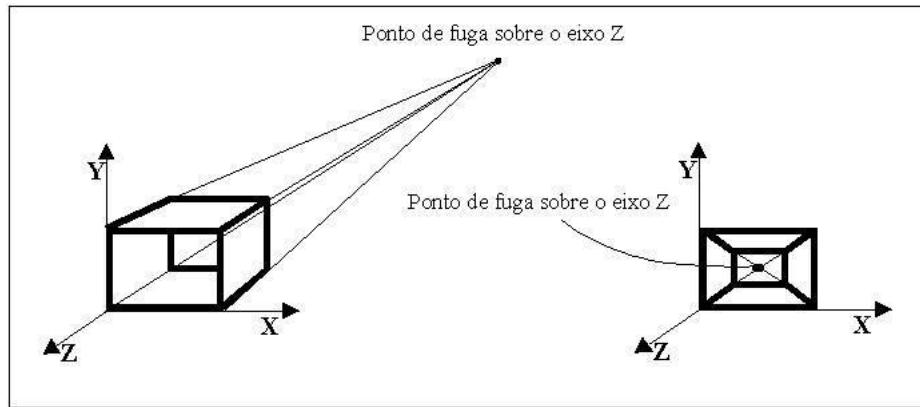


Figura 6.8: Projeções de um cubo (com 1 ponto de fuga) sobre um plano cortando o eixo  $Z$ , apresentando o ponto de fuga.

Projeções perspectivas com 2 pontos de fuga (quando 2 eixos principais são interceptados pelo plano de projeção) são mais comumente usadas em arquitetura, engenharia, desenho publicitário e projeto industrial (ver Figura 6.9). Já as projeções perspectivas com 3 pontos de fuga são bem menos utilizadas, pois adicionam muito pouco em termos de realismo comparativamente às projeções com 2 pontos de fuga, e o custo de implementação é bem maior (veja Figura 6.10).

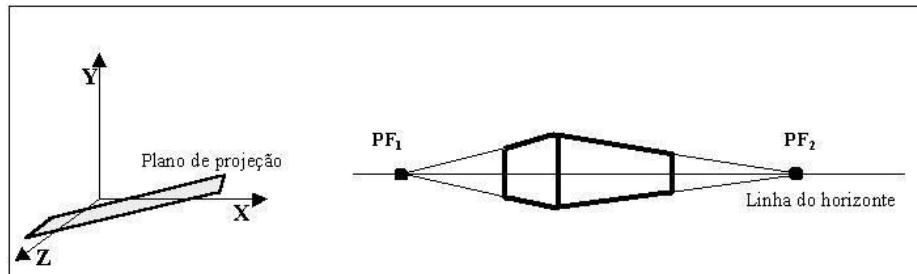


Figura 6.9: Projeções perspectivas com 2 pontos de fuga (o plano de projeção intercepta 2 eixos ( $x$  e  $z$ )).

### Anomalias da Perspectiva

A projeção perspectiva introduz certas anomalias que aumentam o realismo em termos de profundidade, mas também alteram as medidas e formas reais do objetos.

1. **Encurtamento perspectivo:** Quanto mais distante um objeto está do centro de projeção, menor parece ser (o tamanho de sua projeção torna-se menor, como mostra a figura 6.11).
2. **pontos de Fuga:** As projeções de retas não paralelas ao plano de projeção, provocam a ilusão de que se interceptam num ponto do horizonte.
3. **Confusão Visual:** Os objetos situados atrás do centro de projeção são projetados no plano de projeção de cima para baixo e de trás para a frente (ver Figura 6.12)
4. **Distorção Topológica:** Consideremos o plano que contém o centro de projeção e que é paralelo ao plano de projeção. Os pontos deste plano são projetados no infinito pela transformação perspectiva.

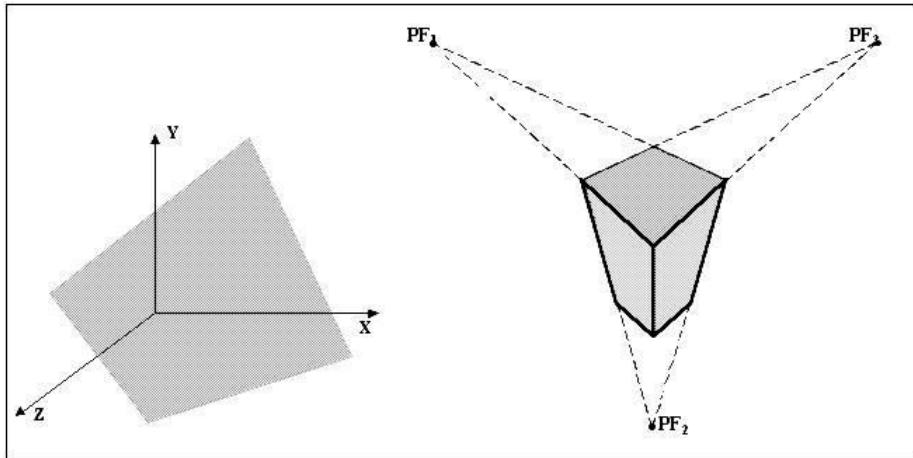


Figura 6.10: Projeções perspectivas com 3 pontos de fuga ( o plano de projeção intercepta os 3 eixos).

Em particular, um segmento de reta que une um ponto situado à frente do observador a um ponto situado atrás dele é efetivamente projetado segundo uma linha quebrada de comprimento infinito.

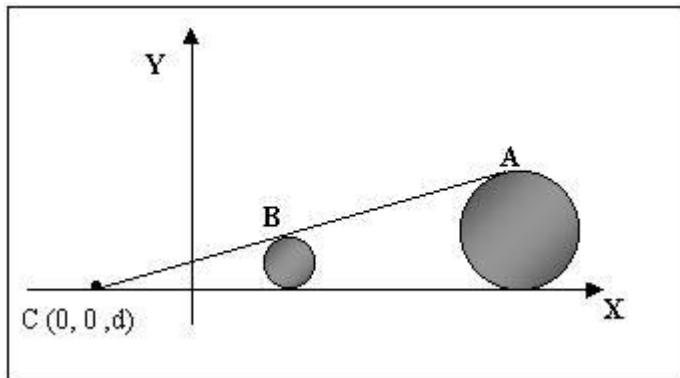


Figura 6.11: A esfera B é bem maior que a esfera A, porém ambas aparecem com o mesmo tamanho quando projetadas no plano de visão.

### Desenvolvimento Matemático para Projeções Perspectivas

Para obter uma projeção perspectiva de um objeto 3D, são transformados os pontos ao longo dos projetores que se encontram no centro de projeção. Suponha que o centro de projeção está posicionado em  $z_{prp}$ , um ponto no eixo  $zv$ , e que o plano de projeção, normal ao eixo  $z$ , está posicionado em  $zvp$ , como mostra a Figura 6.13. Precisamos determinar as coordenadas  $(x_p, y_p, z_p)$ , que são as coordenadas do ponto  $P = (x, y, z)$  projetado no plano de projeção. Podemos escrever as equações que descrevem as coordenadas  $(x', y', z')$  de qualquer ponto ao longo da linha de projeção perspectiva como:

$$\begin{aligned} x' &= x - xu \\ y' &= y - yu \\ z &= z - (z - z_{prp})u \end{aligned} \tag{6.6}$$

O parâmetro  $u$  assume valores no intervalo  $[0, 1]$  quando  $u = 0$ , estamos em  $P = (x, y, z)$ , e quando  $u = 1$  temos exatamente o centro de projeção  $(0, 0, z_{prp})$ . As projeções perspectiva e paralela também podem, assim como as transformações geométricas básicas, ser definidas através de matrizes  $4 \times 4$ , o que é interessante para a composição de transformações juntamente com a projeção. No plano de observação, sabemos que  $z' = z_{vp}$ , e podemos resolver a equação de  $z'$  para obter o valor do parâmetro  $u$  nessa posição ao longo da linha de projeção:

$$u = \frac{Z_{vp} - z}{Z_{prp} - z} \tag{6.7}$$

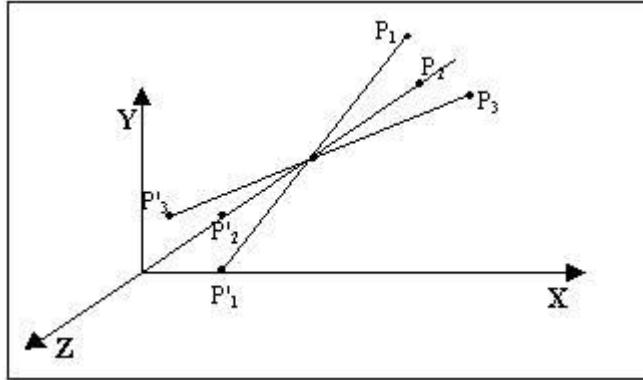


Figura 6.12: Confusão visual da perspectiva (objeto atrás do centro de projeção).

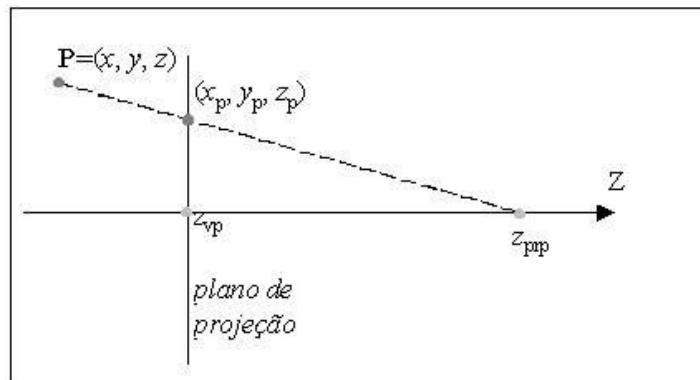


Figura 6.13: Projeção em perspectiva de um ponto  $P=(x, y, z)$  na posição  $(x_p, y_p, z_p)$  sobre o plano de projeção [Hearn and Baker, 1994].

Substituindo esse valor de  $u$  nas equações de  $z'$  e  $y'$ , obtemos as equações de transformação perspectiva:

$$\begin{aligned} x_p &= x \left( \frac{Z_{vp} - z}{Z_{pp} - z} \right) = x \left( \frac{d_p}{Z_{pp} - z} \right) \\ y_p &= y \left( \frac{Z_{vp} - z}{Z_{pp} - z} \right) = y \left( \frac{d_p}{Z_{pp} - z} \right) \end{aligned} \quad (6.8)$$

Usando coordenadas homogêneas pode-se escrever a transformação na forma matricial:

$$\begin{bmatrix} x_h \\ y_h \\ z_h \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{z_{vp}}{d_p} & -z_{vp} \left( \frac{z_{pp}}{d_p} \right) \\ 0 & 0 & -\frac{1}{d_p} & -\frac{z_{pp}}{d_p} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (6.9)$$

Nesta representação o fator homogêneo é

$$h = \frac{z_{pp} - z}{d_p} \quad (6.10)$$

E as coordenadas do ponto projetado no plano de observação são obtidas a partir das coordenadas homogêneas dividindo-se por  $h$ :

$$\begin{aligned} x_p &= \frac{x_h}{h} \\ y_p &= \frac{y_h}{h} \end{aligned} \quad (6.11)$$

Observa-se que o valor original da coordenada  $z$  precisa ser mantido nas coordenadas de projeção para uso por algoritmos de remoção de linhas ocultas. Em geral, o centro da projeção não precisa ser

positionado ao longo do eixo  $z_v$ , e as equações acima podem ser generalizadas para considerar o centro em um ponto qualquer. Existem vários casos especiais a serem considerados para as equações acima. Por exemplo, se o plano de projeção coincide com o plano  $uv$ , i.e.,  $z_{vp} = 0$ , então  $d_p = z_{prp}$ . Em alguns pacotes gráficos é assumido que o centro de projeção coincide com a origem do sistema de coordenadas de observação, i.e., nesse caso  $z_{prp} = 0$ .

### 6.3.2 Desenvolvimento Matemático para Projeções Paralelas

Podemos especificar uma projeção paralela com um vetor de projeção que define a direção dos projetores. Quando a direção projeção é perpendicular ao plano de projeção tem-se uma projeção ortográfica, caso contrário tem-se uma projeção oblíqua. Projeções ortográficas são muito comuns em engenharia e arquitetura para gerar vistas frontais, laterais e traseiras de objetos. A derivação das equações de transformação para uma projeção paralela ortográfica é imediata. Se o plano de observação está posicionado em  $z_{vp}$  ao longo do eixo  $z_v$ , então a descrição de qualquer ponto  $P = (x, y, z)$  em coordenadas do sistema de observação é transformada para as coordenadas de projeção (veja Figura 6.14):

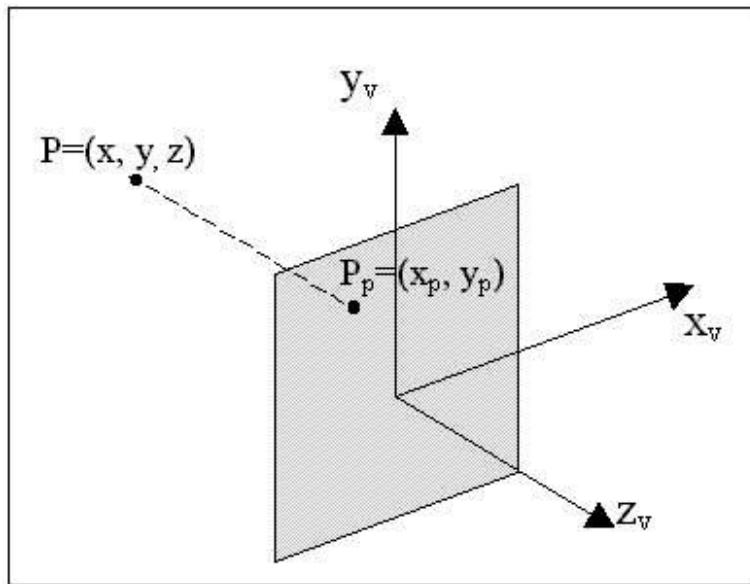


Figura 6.14: Projeção ortogonal de um ponto no plano de projeção [Hearn and Baker, 1994].

A matriz de transformação para coordenadas homogêneas é dada por:

$$M_{ortograf} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6.12)$$

# Capítulo 7

## Recorte de Primitivas 2D

Já vimos que um “pacote gráfico” atua como intermediário entre o aplicativo (e o seu modelo/estrutura de dados interna) e o hardware de visualização. Sua função é aproximar primitivas matemáticas (“ideais”), descritas em termos de vértices num sistema de coordenadas cartesianas, por conjuntos de pixels com a cor ou nível de cinza apropriado. Estes pixels são armazenados num bitmap na memória da CPU, ou num frame buffer (memória de imagem no controlador do dispositivo). Até o momento estudamos, de maneira não extensiva, alguns algoritmos básicos para conversão matricial utilizados por pacotes gráficos. Vamos estudar agora alguns algoritmos para *clipping* [recorte] de primitivas. (O “recorte” é necessário para que a imagem “apareça” dentro do retângulo de visualização definido para ela.

Existem várias abordagens para o processo de *clipping*. Uma técnica óbvia é recortar a primitiva antes da conversão matricial, calculando analiticamente suas intersecções com o retângulo de recorte/visualização [*clip rectangle*]. Estes pontos de intersecção são então usados para definir os novos vértices para a versão recortada da primitiva. A vantagem, evidente, é que o processo de conversão matricial precisa tratar apenas da versão recortada da primitiva, cuja área pode ser muito menor que a original. Esta é a técnica mais freqüentemente utilizada para recortar segmentos de reta, retângulos e polígonos, e os algoritmos que vamos estudar são baseados nesta estratégia.

Outra estratégia seria converter todo o polígono, mas traçar apenas os pixels visíveis no retângulo de visualização [*scissoring*]. Isto pode ser feito checando as coordenadas de cada pixel a ser escrito contra os limites do retângulo. Na prática, existem maneiras de acelerar o processo que evitam o teste de cada pixel individualmente. Se o teste dos limites puder ser feito rapidamente [por hardware especializado, por exemplo], esta abordagem pode ser mais eficiente que a anterior, e tem a vantagem de ser extensível a regiões de recorte arbitrárias.

### 7.1 Recorte de segmentos de reta

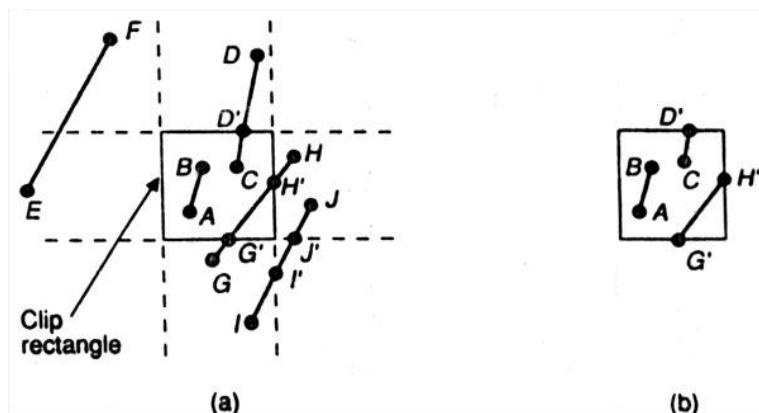


Figura 7.1: Exemplos de recorte de segmentos de reta.

Vamos estudar especificamente o processo de recorte de segmentos de reta contra retângulos. Segmentos que interceptam uma região de recorte retangular, depois de recortados, são sempre transformados num único segmento. Segmentos que estão sobre a fronteira do retângulo de recorte são considerados como dentro dele, e portanto devem ser mostrados (Figura 7.1).

### 7.1.1 Recorte de Pontos Extremos

Antes de discutir o problema do recorte de segmentos de reta, vamos considerar o problema mais simples de recortar pontos extremos. Se a fronteira do retângulo de recorte tem coordenada  $x$  no intervalo  $x_{min}$  e  $x_{max}$ , e coordenada  $y$  no intervalo  $y_{min}$  e  $y_{max}$ , então 4 desigualdades precisam ser satisfeitas para que um ponto  $(x, y)$  esteja dentro do retângulo de recorte:

$$\begin{aligned} x_{min} \leq x \leq x_{max} \\ y_{min} \leq y \leq y_{max} \end{aligned} \quad (7.1)$$

### 7.1.2 Algoritmo de Cohen-Sutherland para Recorte de Segmentos de Reta

Para recortar um segmento de reta, precisamos considerar apenas os seus pontos extremos, e não os infinitos pontos no interior. Se ambas as extremidades estão dentro do retângulo de visualização (AB na Figura 7.1), toda a linha está dentro do retângulo e pode ser traçada (neste caso diz-se que a reta foi trivialmente aceita). Se apenas uma das extremidades está dentro (CD na Figura 7.1), a linha intercepta o retângulo de visualização, e o ponto de intersecção precisa ser calculado. Se ambas as extremidades estão fora, a linha pode ou não interceptar o retângulo, e precisamos verificar se as intersecções existem e onde estão.

Uma estratégia de força bruta seria checar a linha contra cada aresta do retângulo de visualização, para localizar um eventual ponto de intersecção. Se existe um, a linha corta o retângulo, e está parcialmente dentro dele. Para cada linha e cada aresta do retângulo, tomamos as duas retas (matematicamente infinitas) que as contém, e calculamos a intersecção entre elas. A seguir, testamos se este ponto é interior - ou seja, se está dentro do retângulo de visualização e da linha. Se este é o caso, existe uma intersecção com o retângulo de visualização. Na Figura 7.1, os pontos de intersecção  $G'$  e  $H'$  são interiores, mas  $I'$  e  $J'$  não são.

Nesta abordagem, precisamos resolver duas equações simultâneas usando multiplicação e divisão para cada par  $\langle$ aresta, linha $\rangle$ . Entretanto, este é um esquema bastante ineficiente, que envolve quantidade considerável de cálculos e testes, e portanto deve ser descartado.

O algoritmo de Cohen-Sutherland é mais eficiente, e executa testes iniciais na linha para determinar se cálculos de intersecção podem ser evitados. Primeiramente, verifica pares de pontos extremos. Se a linha não pode ser trivialmente aceita, são feitas verificações por regiões. Por exemplo, duas comparações simples em  $x$  mostram que ambos os pontos extremos da linha EF na Figura 7.1 têm coordenada  $x$  menor que  $x_{min}$ , e portanto estão na região à esquerda do retângulo de visualização (ou seja, fora do semi-plano definido pela aresta esquerda). A consequência é que o segmento EF pode ser trivialmente rejeitado, e não precisa ser recortado ou traçado. Da mesma forma, podemos rejeitar trivialmente linhas com ambos os extremos em regiões à direita de  $x_{max}$ , abaixo de  $y_{min}$  e acima de  $y_{max}$ . Se o segmento não pode ser trivialmente aceito ou rejeitado, ele é subdividido em dois segmentos por uma aresta de recorte, um dos quais pode ser trivialmente rejeitado. Dessa forma, um segmento é recortado interativamente testando-se aceitação ou rejeição trivial, sendo subdividido se nenhum dos dois testes for bem-sucedido, até que o segmento remanescente esteja totalmente contido no retângulo ou totalmente fora dele.

Para executar os testes de aceitação ou rejeição trivial, as arestas do retângulo de visualização são estendidas de forma a dividir o plano do retângulo de visualização em 9 regiões [Figura 7.2]. A cada região é atribuído um código de 4 bits, determinado pela posição da região com relação aos semi-planos externos às arestas do retângulo de recorte. Uma maneira eficiente de calcular os códigos resulta da observação de que o bit 1 é igual ao bit de sinal de  $(y_{max} - y)$ ; o 2 é o bit de sinal de  $(y - y_{min})$ ; o 3 é o bit de sinal de  $(x_{max} - x)$ , e o 4 é o de  $(x - x_{min})$ .

- 1º bit: semiplano acima da aresta superior  $y > y_{max}$
- 2º bit: semiplano abaixo da aresta inferior  $y < y_{min}$
- 3º bit: semiplano a direita da aresta direita  $x > x_{max}$
- 4º bit: semiplano a esquerda da aresta esquerda  $x < x_{min}$

A cada extremidade do segmento de reta é atribuído o código da região a qual ela pertence. Esses códigos são usados para determinar se o segmento está completamente dentro do retângulo de visualização ou em um semiplano externo a uma aresta. Se os 4 bits dos códigos das extremidades são iguais a zero, então a linha está completamente dentro do retângulo. Se ambas as extremidades estiverem no semiplano externo a uma aresta em particular, como para EF na Figura 7.1, os códigos de ambas as extremidades tem o bit correspondente àquela aresta igual a 1. Para EF, os códigos são 0001 e 1001, respectivamente,

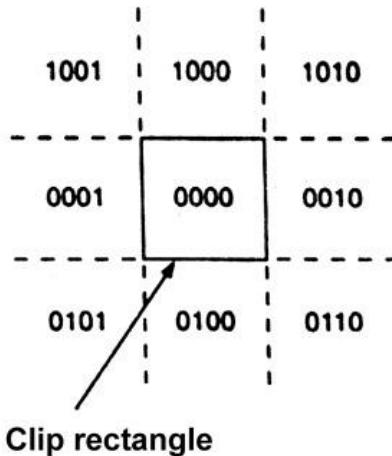


Figura 7.2: Códigos das regiões.

o que mostra com o 4º bit que o segmento está no semiplano externo à aresta esquerda. Portanto, se um *and* bit a bit dos códigos das extremidades for diferente de zero, o segmento pode ser trivialmente rejeitado.

Se a linha não puder ser trivialmente aceita ou rejeitada, devemos subdividi-la em dois segmentos de forma que um deles possa ser rejeitado. A subdivisão é feita usando uma aresta que seja interceptada pela linha para subdividi-la em dois segmentos: a seção que fica no semiplano externo à aresta é descartada. O teste de quais arestas interceptam a linha segue a mesma ordem dos bits nos códigos: de cima-para-baixo, da direita-para-a-esquerda. Uma propriedade chave dos códigos é que os bits iguais a um correspondem a arestas interceptadas: se uma extremidade está no semiplano externo a uma aresta, e o segmento falha nos testes de rejeição trivial, então a outra extremidade deve estar no semiplano interno àquela aresta, que é portanto interceptada pelo segmento. Assim, o algoritmo sempre escolhe um ponto que esteja fora, e usa os bits do código iguais a um) para determinar uma aresta de recorte. A aresta escolhida é a primeira encontrada na ordem pré-definida, ou seja, a correspondente ao primeiro bit igual a 1 no código.

O algoritmo funciona da seguinte maneira: Calcula primeiramente os códigos de ambas as extremidades, e checa aceitação ou rejeição trivial. Se nenhum dos testes for bem-sucedido, tomamos uma extremidade externa (pelo menos uma delas é externa), e testamos seu código para obter uma aresta que seja interceptada e determinar o ponto de intersecção. Podemos então descartar o segmento que vai do extremo externo ao ponto de intersecção, substituindo o ponto externo pela intersecção. O novo código para o segmento correspondente é calculado, preparando os dados para a próxima iteração.

Considere por exemplo o segmento AD na Figura 7.4. O código do ponto A é 0000, e o do ponto D é 1001. O segmento não pode ser trivialmente aceito ou rejeitado. O algoritmo escolhe D como o ponto externo, e o seu código mostra que a linha cruza as arestas superior e esquerda. De acordo com a ordem de teste, usamos primeiro a aresta superior para recortar AD, gerando B. O código de B é 0000 e na próxima interação o segmento será trivialmente aceito e traçado.

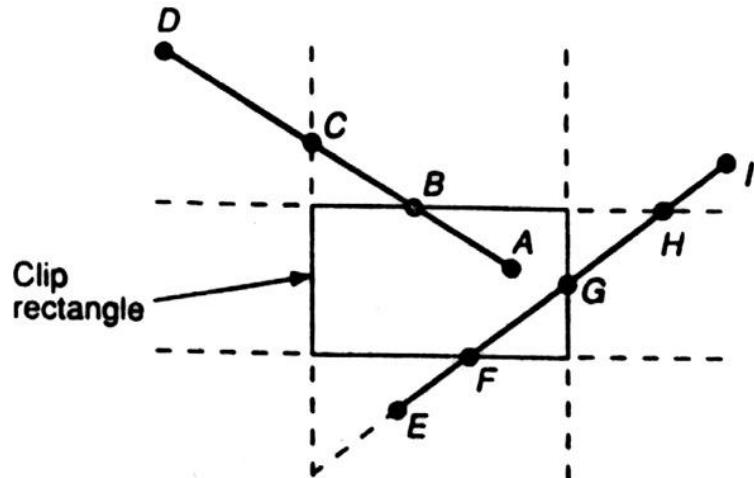


Figura 7.3: Funcionamento do algoritmo de Cohen-Sutherland para recorte de segmentos.

A linha EI requer mais iterações. O primeiro extremo, E, tem código 0100 e é escolhido pelo algoritmo como ponto externo. O código é testado para obter a primeira aresta que a linha cruza, no caso a aresta inferior, e EI é recortado para FI. Na segunda iteração, FI não pode ser trivialmente aceito ou rejeitado. O código da primeira extremidade, F, é 0000, de forma que o algoritmo escolhe o ponto externo I, cujo código é 1010. A primeira aresta interceptada é a superior, resultando no segmento FH. O código de H é 0010, e a terceira iteração resulta num corte pela aresta direita. O segmento resultante é trivialmente aceito e traçado na quarta iteração. O Algoritmo 7.1 e 7.2 ilustram este procedimento.

Este algoritmo não é o mais eficiente - como os testes e recorte são executados numa ordem fixa, eventualmente serão feitos recortes desnecessários. Por exemplo, quando a intersecção com uma aresta do retângulo é uma “intersecção externa”, que não está na fronteira do retângulo de visualização (ponto H na linha EI, Figura 7.4). Existem algoritmos mais eficientes (v. Foley, 3.12), mas este ainda é o mais utilizado, por ser simples e bastante conhecido.

```

typedef enum {LEFT = 0, RIGHT = 1, BOTTOM = 2, TOP = 3} aresta;
typedef enum {TRUE, FALSE} boolean;
typedef boolean outcode;

/* Retorna TRUE se todos os outcodes em codigo sao FALSE, e
 * FALSE c.c. */
boolean Vazio(outcode codigo[]);

/* Retorna TRUE se a interseccao logica entre codigo1 e codigo
 * 2 e vazia, e FALSE c.c. */
boolean InterseccaoVazia(outcode codigo0[], outcode codigo1 []);

/* Retorna TRUE se codigo0 e codigo 1 sao iguais, e FALSE c.c. */
boolean Igual(outcode codigo0[], outcode codigo1 []);

/* Calcula outcode para ponto (x,y) */
void CalculaOutCode(float x, float y, outcode codigo[]) {
    int i;

    for (i = LEFT; i <= TOP; i++)
        codigo[i] = FALSE;
    /* Fim for */
    if (y > YMAX)
        codigo[TOP] = TRUE;
    else if (y < YMIN)
        codigo[BOTTOM] = TRUE;
    /* Fim se */
    if (x > xmax)
        codigo[RIGHT] = TRUE;
    else if (x < xmin)
        codigo[BOTTOM] = TRUE;
    /* Fim Se */
}/* End CalculaOutCode */

```

**Algoritmo 7.1:** Calculando os códigos do algoritmo de Cohen-Sutherland.

## 7.2 Recorte de Circunferências

Para recortar uma circunferência contra um retângulo, podemos primeiro executar um teste de aceitação/rejeição trivial interceptando a extensão da circunferência (um quadrado do tamanho do diâmetro da circunferência) com o retângulo de visualização usando o algoritmo para recorte de polígonos que será visto a seguir. Se a circunferência intercepta o retângulo, ela é subdividida em quadrantes, e testes de aceitação/rejeição trivial são feitos para cada um. Estes testes podem levar a testes por octantes. É possível então calcular a intersecção da circunferência e da aresta analiticamente, resolvendo suas equações simultaneamente, e a seguir fazer a conversão matricial dos arcos resultantes. Se a conversão matricial é rápida, ou se a circunferência não é muito grande, seria provavelmente mais eficiente usar a técnica de *scissoring*, testando cada pixel na circunferência contra os limites do retângulo antes de traçá-lo.

Um algoritmo para recorte de polígonos precisa tratar de muitos casos distintos, como mostra a Figura 7.5. O caso (a) é particularmente interessante porque um polígono côncavo é recortado em dois polígonos

```

/* algoritmo de recorte de Cohen-Sutherland para linha P0(x0,y0) a
 * P1(x1,y1), e retangulo de recorte com diagonal de (xmin,ymin) a
 * (xmax,ymax). */
void RecorteLinhaCohenSutherland(float x0,float y0,float x1,float y1,int valor){
    boolean aceito, pronto;
    outcode outcode0[4], outcode1 [4], *outcodeOut; /* outcodes para P0, P1 e
                                                       quaisquer outros pontos que
                                                       estao fora do retangulo de
                                                       recorte */

    float x, y;
    aceito = FALSE; pronto = FALSE;

    CalculaOutCode(x0,y0,outcode0); CalculaOutCode(x1,y1,outcode1);
    do {
        if (vazio(outcode0) && vazio(outcode1)){
            /* aceitacao trivial e sai */
            aceito = TRUE;
            pronto = TRUE;
        }else if (interseccao_vazia(outcode0,outcode1))
            /* rejeicao trivial e sai */
            pronto = TRUE;
        else {
            /* ambos os testes falharam, entao calcula o segmento de
               reta a recortar: a partir de um ponto externo a uma
               interseccao com a aresta de recorte */
            /* pelo menos um ponto esta fora do retangulo de recorte;
               seleciona-o */

            outcodeOut= vazio(outcode0) ? outcode1 : outcode0;
            /* acha agora o ponto de interseccao, usando as formulas:
               y = y0 + inclinacao * (x-x0), x = x0 + ( 1 /inclinacao) * (y-y0) */
            if (outcodeOut(TOP)) {
                /* divide a linha no topo do retangulo de recorte */
                x= x0 + (x1-x0) * (ymax-y0)/(y1-y0); y= ymax;
            }else if (outcodeOut[BOTTOM]) {
                /* divide a linha na base do retangulo de recorte */
                x = x0 + (x1-x0) + (ymin-y0)/(y1-y0); y = ymin;
            }else if (outcodeOut[RIGHT]) {
                /* divide a linha na aresta direita do retangulo de recorte */
                y = y0 + (y1 -y0) + (xmax-x0)/(x1-x0); x = xmax;
            }else if (outcodeOut[LEFT]) {
                /* divide a linha na aresta esquerda do retangulo de recorte */
                y = y0 + (y1 -y0) + (xmin-x0)/(x1-x0); x = xmin;
            } /* Fim se */

            /* agora move o ponto externo para o ponto de interseccao, para
               recortar e preparar para o proximo passo */
            if (igual(outcodeOut,outcode0)) {
                x0 = x; y0 = y; CalculaOutCode(x0,y0,outcode0);
            }else{
                x1 = x; y1 = y; CalculaOutCode(x1,y1,outcode1);
            }
        }/* fim do else da subdivisao */
    } while (pronto);

    if (aceito)
        MeioPontoLinhaReal(x0,y0,x1,y1),valor); /* versao do algoritmo para coordenadas reais */
} /* fim do algoritmo de recorte e tracado */

```

**Algoritmo 7.2:** Desenhando a linha com o algoritmo de Cohen-Sutherland. Algumas das funções usadas estão ilustradas no Algoritmo 7.2.

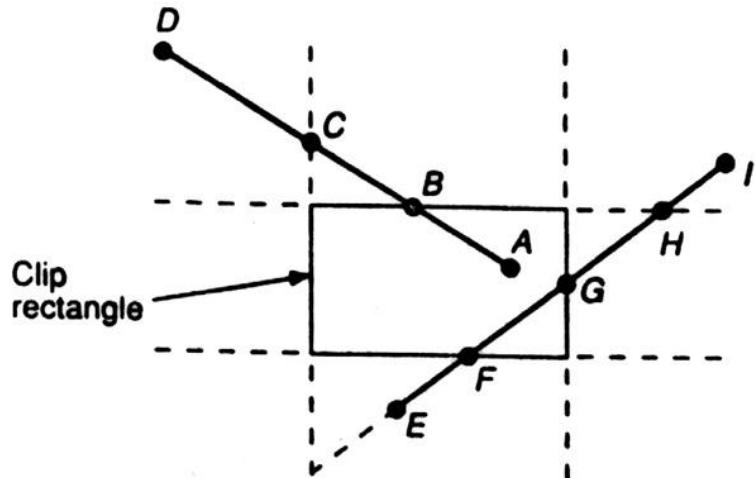


Figura 7.4: Algoritmo de Sutherland-Hodgman para Recorte de Polígonos.

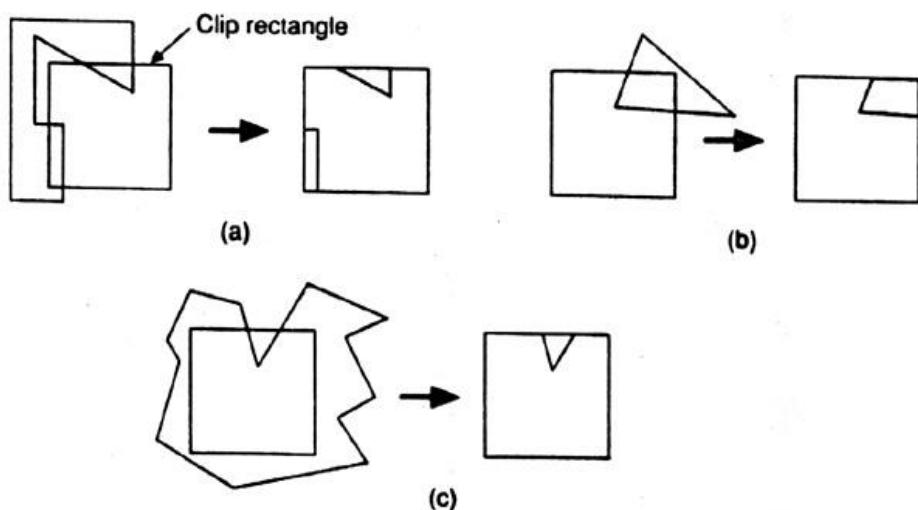


Figura 7.5: Exemplos de recorte de polígonos. (a) Múltiplos componentes. (b) Caso convexo (c) Caso côncavo com muitas arestas exteriores.

separados. O problema é complexo porque cada aresta do polígono precisa ser testada contra cada aresta do retângulo de visualização; novas arestas precisam ser acrescentadas, arestas existentes precisam ser descartadas, mantidas ou subdivididas. Vários polígonos podem resultar do recorte de um único, e é necessário uma abordagem organizada para tratar todos os casos possíveis.

O algoritmo para recorte de polígonos de Sutherland-Hodgman usa a estratégia “dividir para conquistar”; resolve uma série de problemas simples e idênticos cujas soluções, quando combinadas, resolvem o problema todo. O problema básico consiste em recortar o polígono contra uma única aresta de recorte infinita. Quatro arestas, cada qual definindo um lado do retângulo, recortam o polígono sucessivamente contra o retângulo de visualização.

Note a diferença entre esta estratégia para um polígono e o algoritmo de Cohen-Sutherland para recorte de linhas: O algoritmo para polígonos recorta quatro arestas sucessivamente, enquanto que o algoritmo para linhas testa o código para verificar qual aresta é interceptada, e recorta apenas quando necessário. O algoritmo de Sutherland-Hodgman é na verdade mais geral: um polígono (côncavo ou convexo) pode ser recortado contra qualquer polígono de recorte convexo. Em 3D, polígonos podem ser recortados contra volumes poliedrais convexas definidos por planos. O algoritmo aceita como entrada uma série de vértices  $v_1, v_2, \dots, v_n$  que, em 2D, definem as arestas de um polígono. A seguir, recorta o polígono contra uma única aresta de recorte infinita, e retorna outra série de vértices que definem o polígono recortado. Num segundo passo, o polígono parcialmente recortado é recortado novamente contra a segunda aresta, e assim por diante.

O algoritmo movimenta-se em torno do polígono, indo do vértice  $v_n$  ao vértice  $v_1$ , e de volta a  $v_n$ , examinando a cada passo a relação entre um vértice e a aresta de recorte. Em cada passo, zero, um ou dois vértices são adicionados à lista de saída dos vértices que definem o polígono recortado, dependendo

da posição do vértice sendo analisado em relação à aresta de recorte.

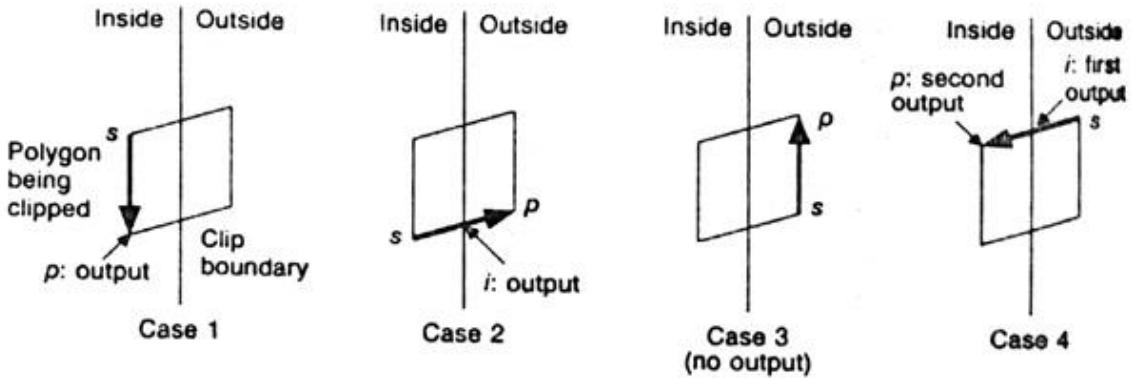


Figura 7.6: Quatro situações possíveis no recorte de polígonos.

São quatro situações possíveis (Figura 7.6). Consideremos uma aresta do polígono que vai do vértice *s* ao vértice *p*. Suponha que o vértice *s* foi analisado na iteração anterior, e a interação corrente está analisando *p*. No caso 1 , a aresta do polígono está completamente dentro da fronteira do retângulo de visualização, e o vértice *p* é adicionado à lista de saída. No caso 2, o ponto *i*, que define a intersecção entre a aresta do polígono e a aresta do retângulo, é colocado como um vértice na lista de saída. No caso 3, ambos os vértices estão fora do retângulo, e nenhum é colocado na lista de saída. No caso 4, ambos os pontos *i* (intersecção) e *p*, são colocados na lista de saída.

O procedimento, mostrado no Algoritmo 7.3, aceita um vetor de vértices, `inVertexArray`, e cria outro vetor `outVertexArray`. Para manter o código simples, não incluímos verificação dos erros nos limites dos vetores, e usamos o procedimento `Output()` para incluir um vértice em `outVertexArray`. O procedimento `Intersect()` calcula a intersecção de uma aresta *s*-*p* do polígono com a aresta `clipBoundary`, definida por dois vértices da fronteira do polígono de recorte. A função `Inside()` retorna true se o vértice está no interior da fronteira de recorte. Assume-se que o vértice está no interior da fronteira se estiver “a esquerda da aresta de recorte, quando olha-se do primeiro para o segundo vértice da aresta”. Esta orientação corresponde a uma enumeração das arestas no sentido anti-horário. Para calcular se um ponto está fora da fronteira de recorte, podemos testar o sinal do produto vetorial do vetor normal à fronteira de recorte e da aresta do polígono.

O algoritmo é bastante geral, e pode ser estruturado para chamar a si próprio recursivamente de uma forma que o torna bastante eficiente para implementação em hardware.

```

/* Algoritmo de Sutherland-Hodgman para recorte do polígonos */
#define MAX 20

typedef struct {float x,y;} ponto;
typedef ponto vertice;
typedef vertice aresta[2];
typedef vertice VetorVertices[MAX];
typedef enum {TRUE = 1, FALSE = 0} boolean;

/* Acrescenta um novo vertice a vetorSaida, e atualiza tamSaida */
void saida(vertice novoVertice, int *tamSaida, VetorVertices vetorSaida);

/* Checa se o vertice esta dentro ou fora da aresta de recorte */
boolean dentro(vertice testeVertice, aresta arestaRecorte);

/* recorta a aresta (prim,seg) do poligono contra arestaRecorte,
 * retorna o novo ponto */
vertice intercepta(vertice prim, vertice seg, aresta arestaRecorte);

void RecortePoligonosSutherlandHodgman(
    VetorVertices vetorEntrada, /* vetor de vertices de entrada */
    VetorVertices vetorSaida, /* vetor de vertices de saida */
    int tamEntrada, /* numero de entradas em vetorEntrada */
    int tamSaida, /* numero de vertices em vetorSaida */
    aresta arestaRecorte /* aresta do poligono de recorte */{
    vertice s, p; /* pontos inicial e final da aresta de recorte corrente */
    vertice i; /* ponto de interseccao com uma aresta de recorte */
    int j; /* contador para o loop de vertices */
    *tamSaida = 0;

    s = vetorEntrada[tamEntrada]; /* começa com o ultimo vertice em vetorEntrada */
    for (j = 1 ;j <= tamEntrada;j ++ ) {
        p= vetorEntrada[j]; /* agora s e p correspondem aos vertices na Fig. */
        if (dentro(s,arestaRecorte)) {
            /* casos 1 e 4 */
            if (dentro(s,arestaRecorte))
                saida(p,tamSaida,vetorSaida);
            else {
                i = intercepta(s,p,arestaRecorte);
                saida(i,tamSaida,vetorSaida);
                saida(p,tamSaida,vetorSaida);
            }/* Fim Se */
        }else{
            /* casos 2 e 3 */
            if (dentro(s,arestaRecorte)) {
                i = intercepta(s,p,arestaRecorte);
                saida(i,tamSaida,vetorSaida);
            }/* Fim Se */
        }/*Fim Se */
        s = p;
    }/* fim do for */
}/* fim do algoritmo de recorte */

```

**Algoritmo 7.3:** Algoritmo de Cohen-Sutherland.

## Capítulo 8

# Curvas e Superfícies em Computação Gráfica

Curvas e superfícies tridimensionais (espaciais) desempenham um papel importante:

- na engenharia, projeto e manufatura de uma ampla gama de produtos, como automóveis, cascos de navios, fuselagem e asas de aviões, lâminas de propulsão, sapatos, garrafas, edificações, etc.
- na descrição e interpretação de fenômenos físicos em áreas como geologia, física e medicina.
- **Projeto Geométrico Apoiado por Computador** (*Computer Aided Geometric Design*): nova disciplina que incorpora modelos matemáticos e computacionais desenvolvidos para apoiar os processos de engenharia, projeto e manufatura.

Frequentemente, superfícies são descritas por uma malha de curvas definidas em planos ortogonais (Figura 8.1). As curvas podem ser obtidas através da digitalização de um modelo físico ou desenho, e posterior ajuste de uma curva matemática aos pontos digitalizados, ou geradas a partir de formulações criadas especificamente para a definição de curvas no espaço.

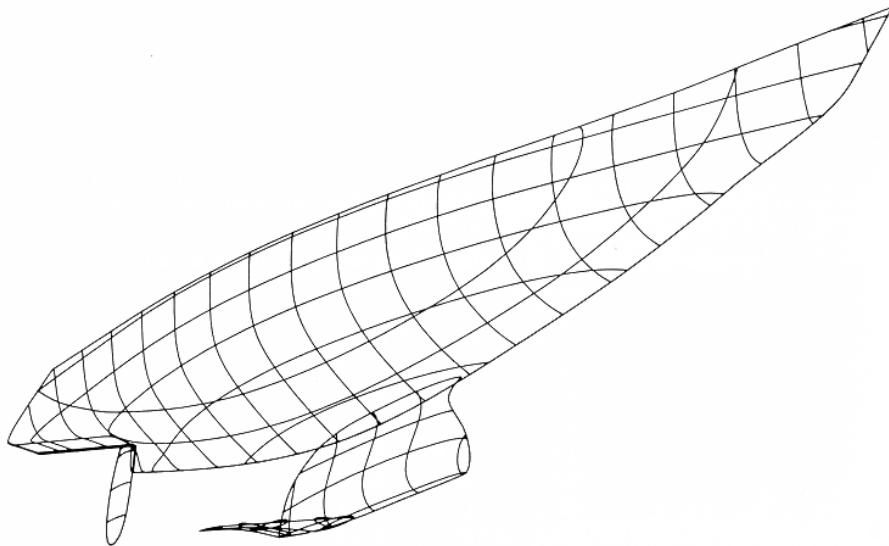


Figura 8.1: Representação de superfícies através de malhas.

### 8.1 Representação de Curvas

Existem duas formas alternativas de representação:

- por um conjunto de pontos: nesse caso, uma representação visual adequada pode ser gerada conectando-se os pontos por segmentos de reta, se estes forem espaçados adequadamente (Figura ??). Observe que nas regiões da curva onde a curvatura é grande, a proximação por segmentos de

reta é particularmente ruim (Figura ??(a)). Uma maneira de melhorar a representação é aumentar a densidade dos pontos nessas regiões (Figura ??(b)).

- analítica: ou seja, através de formulações matemáticas, o que apresenta várias vantagens em relação à representação anterior:
  - precisão,
  - armazenagem compacta,
  - facilidade de cálculo (exato) de pontos intermediários (em uma representação por pontos pontos intermediários precisam ser determinados por interpolação),
  - facilidade para calcular propriedades da curva como inclinação e curvatura (em uma representação por pontos tais propriedades precisam ser calculadas por diferenciação numérica, um procedimento que é pouco preciso),
  - facilidade para desenhar as curvas,
  - facilidade para fazer alterações contínuas no formato da curva de forma a atender requisitos de projeto (“design”).

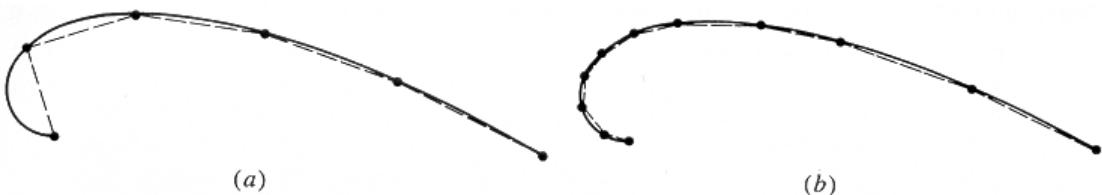


Figura 8.2: Aproximação de curvas por segmentos de reta conectados.

## 8.2 Curve Fitting x Curve Fairing

### 8.2.1 Ajuste de curvas (curve fitting)

Frequentemente, deseja-se obter representações analíticas para curvas definidas originalmente por conjuntos de pontos - por exemplo, pontos em uma curva ou superfície real foram digitalizados. Do ponto de vista matemático, esse é um problema de **interpolação**. Uma curva que **ajusta** (*fit*) os pontos dados passa por todos esses pontos. Uma técnica usual de ajuste de curvas são as **splines cúbicas**, uma estratégia de aproximação polinomial por partes.

### 8.2.2 Aproximação de curvas (curve fairing)

Se os pontos dados são aproximações para valores desconhecidos, por exemplo, são pontos coletados ou obtidos em medidas experimentais, então o que se deseja é uma curva que mostre a tendência dos dados. Em geral, a curva obtida pode não passar por nenhum dos pontos dados, e diz-se que a curva **aproxima** (*fair*) os dados. Alternativamente, pode-se desejar gerar uma descrição matemática de uma curva no espaço sem qualquer conhecimento prévio da forma da curva. Para esse caso, técnicas usuais são as representações de Bézier e B-splines (ambas são estratégias de **aproximação de curvas**).

## 8.3 Representações Paramétricas e Não Paramétricas (explícita e implícita)

Para uma curva plana, a forma não paramétrica explícita é dada por

$$y = f(x) \quad (8.1)$$

por exemplo, a equação de uma linha reta é dada por

$$y = mx + b \quad (8.2)$$

Desta forma, obtém-se um valor de  $y$  para cada valor de  $x$  dado. Consequentemente, curvas fechadas, ou com valores múltiplos, como um círculo, não podem ser representadas explicitamente. Essa limitação não existe no caso de representações implícitas, na forma

$$f(x, y) = 0 \quad (8.3)$$

A equação implícita de segundo grau genérica:

$$ax^2 + 2bxy + cy^2 + 2dx + 2ey + f = 0 \quad (8.4)$$

engloba uma variedade de curvas bi-dimensionais denominadas **seções cônicas**. Os três tipos de seções cônicas são a **parábola**, a **hipérbole** e a **elipse** (um círculo é um caso especial de uma elipse) (Figura ??).

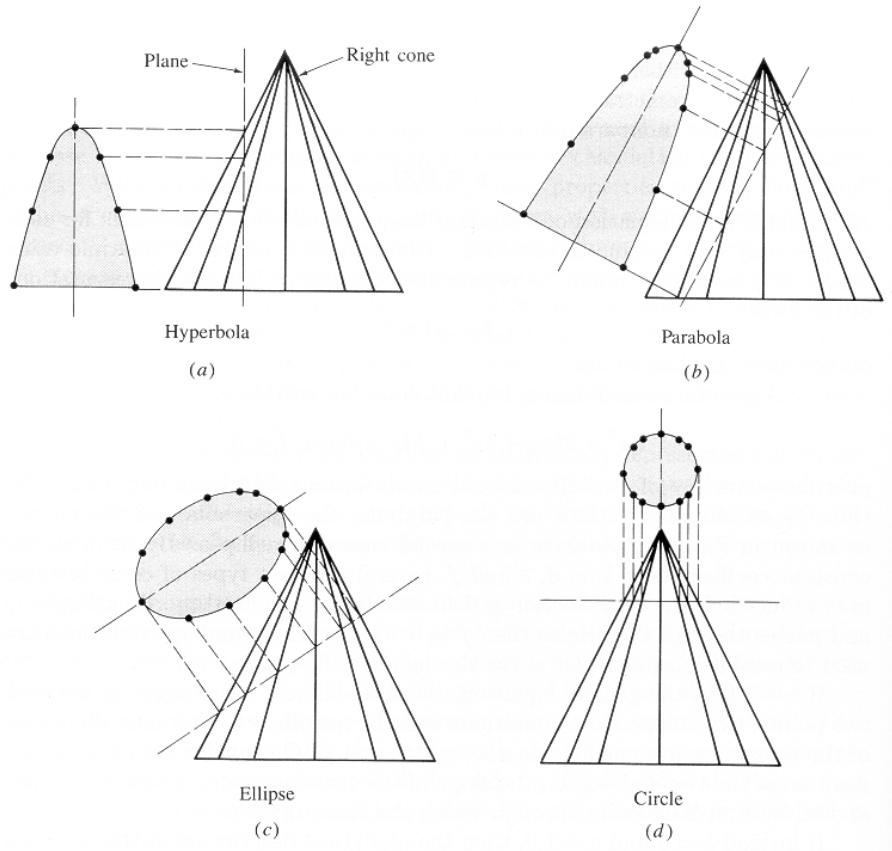


Figura 8.3: Aproximação de curvas por segmentos de reta conectados.

Dependendo dos valores de  $a, b, c, d, e$  e  $f$ , diferentes tipos de seções cônicas são produzidas. Se a seção cônica é definida em relação a um sistema de coordenadas local e passa pela origem, então  $f = 0$ .

### 8.3.1 Limitações das representações não paramétricas

- Ambas (explícita e implícita) são dependentes do sistema de coordenadas, cuja escolha afeta a facilidade de uso. Por exemplo, surgem problemas se no sistema de coordenadas escolhido for necessária uma inclinação infinita como condição de contorno, pois essa inclinação não pode ser usada diretamente como uma condição de contorno numérica. Ou o sistema de coordenadas é alterado, ou a inclinação infinita será representada numericamente por um valor muito grande, positivo ou negativo.

- pontos em uma curva calculados a partir de incrementos uniformes em  $x$  ou  $y$  não estão distribuídos uniformemente ao longo da curva, o que afeta a qualidade e precisão de uma representação gráfica. Estas limitações são superadas pelo uso de representações paramétricas.

Na forma paramétrica, cada coordenada de um ponto em uma curva é representada como uma função de um único parâmetro, sendo que a posição de um ponto na curva é fixada pelo valor do parâmetro. Para uma curva 2D que usa  $t$  como parâmetro, as coordenadas cartesianas de um ponto na curva são:

$$\begin{aligned} x &= x(t) \\ y &= y(t) \end{aligned} \quad (8.5)$$

O vetor que define a posição de um ponto na curva é portanto dado por

$$P(t) = [x(t) \quad y(t)] \quad (8.6)$$

A derivada, ou vetor tangente da curva é dada por

$$P'(t) = [x'(t) \quad y'(t)] \quad (8.7)$$

A inclinação da curva é dada por

$$\frac{dx}{dy} = \frac{\frac{dx}{dt}}{\frac{dy}{dt}} = \frac{x'(t)}{y'(t)} \quad (8.8)$$

## Observações

- A forma paramétrica é adequada para representar curvas fechadas e com valores múltiplos. A forma não paramétrica pode ser obtida eliminando-se o parâmetro para obter uma única equação em  $x$  e  $y$ .
- Uma vez que um ponto na curva é especificado por um único valor de parâmetro, a forma paramétrica é independente do sistema de coordenadas. Os extremos e o comprimento da curva são fixos pelo intervalo de variação do parâmetro, frequentemente normalizado para , por conveniência. Como as curvas são independentes do sistema de coordenadas, elas são facilmente manipuladas usando as transformações geométricas afins (ver Capítulo 5).
- Determinar um ponto em uma curva, i.e., determinar o valor de  $y$ , dado  $x$ , é trivial no caso da representação explícita. No caso da paramétrica, é necessário obter o valor do parâmetro  $t$ , a partir de  $x$ , e a seguir usar este valor para obter  $y$ . (Para equações paramétricas mais complexas, uma técnica iterativa pode ser mais conveniente).
- Ambas as formas de representação têm vantagens e desvantagens em situações específicas!

Exemplos:

### 1. segmento de reta

Para 2 vetores que especificam as posições iniciais  $P_1$  e  $P_2$ , uma representação paramétrica do segmento é dada por

Como  $P(t)$  é um vetor de posição, cada um de seus componentes tem uma representação paramétrica  $x(t)$  e  $y(t)$  entre  $P_1$  e  $P_2$ .

### 2. Círculo no primeiro quadrante

Uma comparação das representações visuais obtidas a partir das formas paramétrica e não-paramétrica para um círculo no primeiro quadrante é apresentada na Figura 8.4 (observe que a representação paramétrica para uma curva não é única!)

Os pontos na circunferência mostrados nas Figuras 8.4(b) e 8.4(c) foram obtidos com intervalos constantes dos parâmetros  $q$  e  $t$ , respectivamente. A primeira representação produz comprimentos de arcos idênticos ao longo da circunferência, ao contrário da segunda (a qual, entretanto, tem aparência visual melhor que a mostrada em 8.4(a), gerada pela representação explícita). Por outro lado, o cálculo das funções trigonométricas é computacionalmente caro, de maneira que a segunda forma de representação paramétrica representa um meio termo entre a forma explícita e a forma paramétrica padrão.

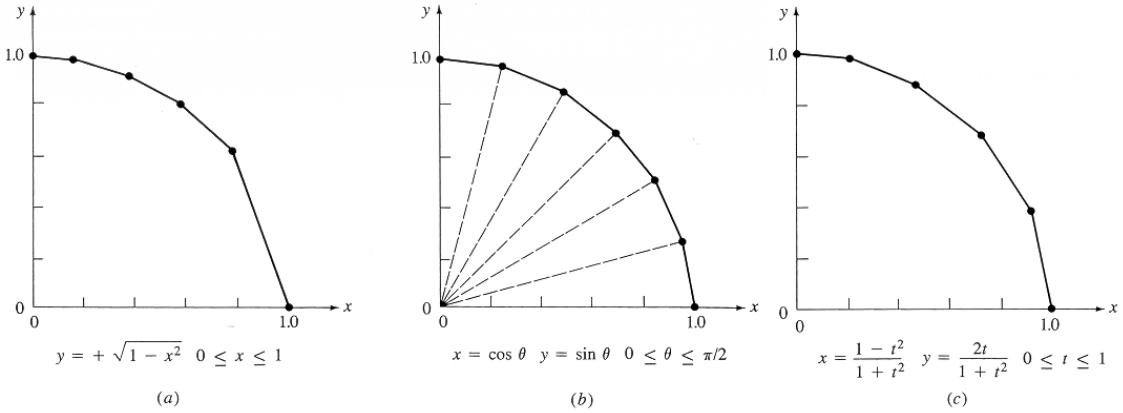


Figura 8.4: Pontos de uma circunferência.

## 8.4 Curvas de Bézier

Técnicas de aproximação de curvas são muito usadas em ambientes de projeto (CAD) interativos, por serem mais intuitivas do que as técnicas de ajuste. Um método adequado para o design de curvas e superfícies de forma-livre em ambientes interativos foi desenvolvido por Pierre Bézier. Uma curva de Bézier é determinada por um polígono de definição (Figuras 8.5 e ??).

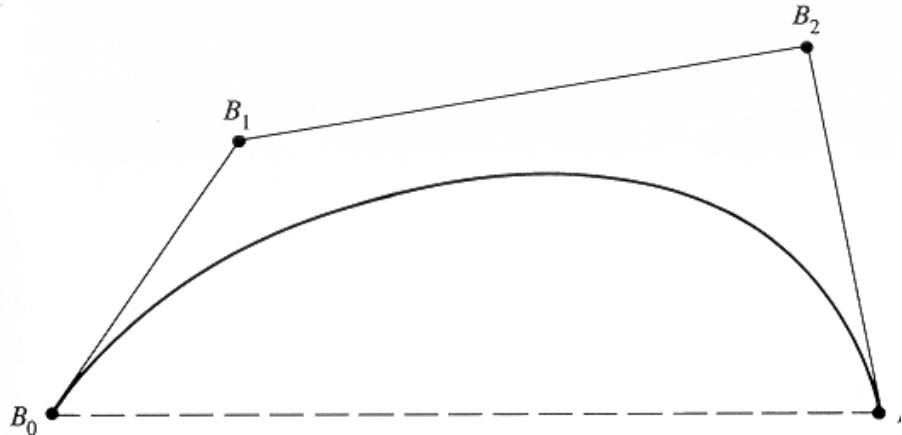


Figura 8.5: Representação de uma curva de Bézier definida pelos pontos  $B_0$ ,  $B_1$ ,  $B_2$  e  $B_3$ .

Matematicamente, uma curva de Bézier paramétrica é definida como

$$P(t) = \sum_{i=0}^n B_i J_{n,i}(t) \quad 0 \leq t \leq 1 \quad (8.9)$$

onde as funções de blending, ou funções base de Bézier são

$$J_{n,i}(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad (8.10)$$

com

$$\binom{n}{i} = \frac{n!}{i!(n-i)!} \quad (8.11)$$

$J_{n,i}(t)$  é a  $i$ -ésima função base de Bernstein de ordem  $n$ , onde  $n$ , o grau da função-base, e portanto do segmento de curva polinomial, é um menos o número de pontos do polígono de definição. Os vértices do polígono são numerados de 0 a  $n$  (como na Figura 8.5). A Figura 8.6 mostra as funções de blending

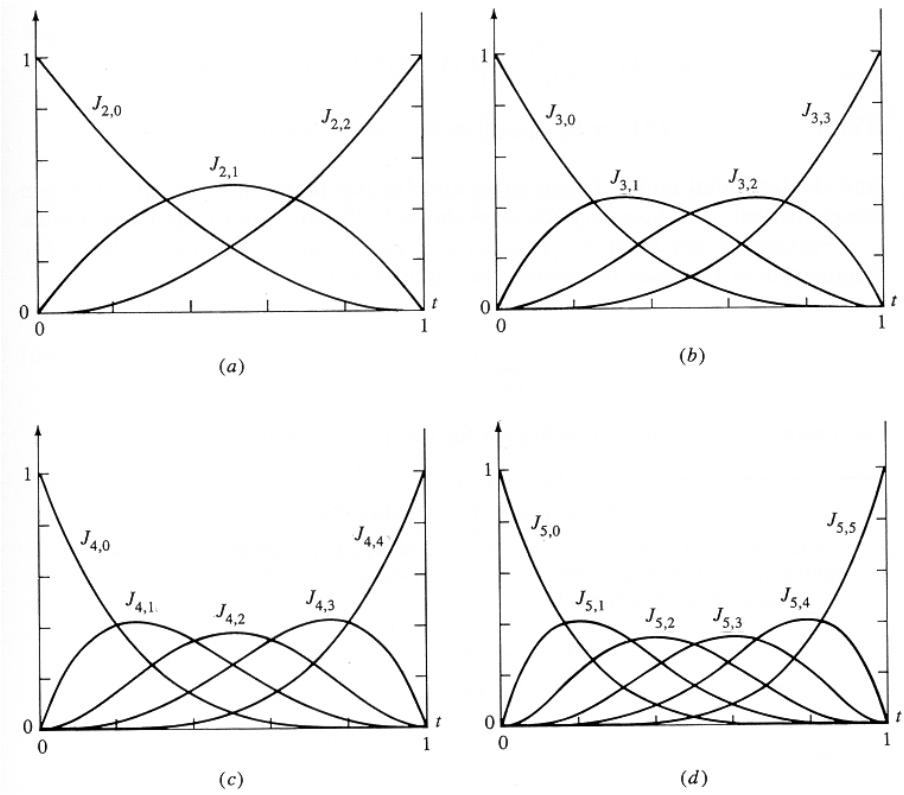


Figura 8.6: Funções de blending para vários valores de  $n$ .

para vários valores de  $n$ . Observe a simetria das funções. Examinando-se as Equações 8.9 a 8.11 para o primeiro ponto na curva, i.e., em  $t = 0$ , verifica-se que

$$\begin{aligned} J_{n,0}(0) &= 1 & i = 0 \\ J_{n,i}(0) &= 0 & i \neq 0 \end{aligned} \tag{8.12}$$

Portanto,

$$P(0) = B_0 J_{n,0}(0) = B_0 \tag{8.13}$$

e o primeiro ponto na curva coincide com o primeiro ponto do polígono de controle. Verificação análoga pode ser feita para o último ponto, i.e., em  $t = 1$ . As funções base da Figura 8.6 ilustram estes resultados.

Além disso, pode-se mostrar que para qualquer valor do parâmetro  $t$ , o somatório das funções base é 1, i.e.,

$$\sum_{i=0}^n J_{n,i} = 1 \tag{8.14}$$

A equação para uma curva de Bézier pode ser expressa na forma matricial como

$$P(t) = TNG = FG \tag{8.15}$$

onde  $F = J_{n,0} J_{n,1} \cdots J_{n,n}$  e  $G^r = B_0 B_1 \cdots B_n$ .

As matrizes específicas para valores pequenos de  $n$  ( $n = 3, 4$ ) são de particular interesse. Para qualquer valor de  $n$ , a matriz  $[N]$  é simétrica em relação à diagonal principal e o canto triangular inferior direito contém apenas zeros.

Em geral, uma forma complexa não pode ser modelada por uma única curva, mas por várias curvas que são conectadas em seus pontos extremos. Ao criar as junções, o projetista, em geral, deseja controlar

a continuidade nos pontos de junção. Continuidade de ordem 0 significa que as 2 curvas se encontram; continuidade de primeira ordem exige que as curvas sejam tangentes no ponto de junção, e continuidade de segunda ordem exige que as curvaturas sejam as mesmas.

A formulação de Bézier apresenta várias propriedades interessantes:

- As funções-base são reais.
- O grau do polinômio que define uma curva é um menos o número de pontos do polígono de definição.
- A forma da curva geralmente acompanha a forma do polígono de definição (na verdade é uma versão “suavizada” da forma do polígono (v. Fig. 3.3)). Assim, para desenhar uma curva, basta definir o polígono e depois ajustar os pontos que forem necessários para aproximar melhor a forma desejada. Isso torna a formulação adequada para o design interativo. Um projetista experiente consegue obter a forma desejada depois de 2 ou 3 interações com um sistema computacional.
- O primeiro e último pontos da curva coincidem com o primeiro e último pontos do polígono de definição. Em situações práticas, em geral é desejável ter controle direto sobre os pontos extremos da curva.
- Os vetores tangentes nos extremos da curva têm a mesma direção que o primeiro e o último segmentos do polígono de definição, respectivamente.
- A curva está contida no fecho convexo do polígono de definição, i.e., no maior polígono convexo que pode ser obtido com os vértices do polígono de definição (Figura 8.5). Uma consequência simples deste fato é que um polígono plano sempre gera uma curva plana.

A importância desta propriedade está principalmente no contexto de verificação de interferência. Suponha que desejamos saber se 2 curvas se interceptam. Por exemplo, cada uma representa o caminho a ser percorrido pelo braço de um robô, e nosso objetivo é garantir que os caminhos não se interceptam, evitando colisões. Ao invés de tentar calcular uma possível intersecção, podemos fazer um teste muito mais barato: envolver o polígono de controle no menor retângulo que o contém e cujas arestas são paralelas a um sistema de coordenadas (minmax boxes). O retângulo contém o polígono de controle e, pela propriedade do fecho convexo, contém também a curva. Verificar se os 2 retângulos se interceptam é um teste trivial, e fica simples checar a não-intersecção. Se houver intersecção dos retângulos, mais verificações precisam ser feitas nas curvas. A possibilidade de detectar não-interferência de forma rápida é muito importante, pois na prática é freqüente que tais testes precisem ser feitos milhares de vezes para cada objeto.

- A curva exibe a propriedade da variação decrescente (variation diminishing property). Isto significa, basicamente, que a curva não oscila em relação a qualquer linha reta com mais frequência que o polígono de definição. Algumas representações matemáticas têm a tendência de ampliar, ao invés de suavizar, quaisquer irregularidades de formato esboçadas pelos pontos de definição, enquanto que outras, como as curvas de Bézier, sempre suavizam os pontos de controle. Assim, a curva nunca cruza uma linha reta arbitrária mais vezes que a sequência de segmentos que conectam os pontos de controle (v. Fig.)
- A curva é invariante sob transformações afins. Transformações afins estão disponíveis em qualquer sistema de CAD, pois é essencial reposicionar, escalar, etc. os objetos. Esta propriedade garante que os 2 procedimentos abaixo produzem os mesmos resultados: a) primeiro calcula um ponto na curva, e depois aplica a ele uma transformação afim; e b) primeiro, aplica uma transformação afim ao polígono de definição, e depois gera o ponto na curva.

Uma consequência prática: suponha que traçamos uma curva cúbica calculando 100 pontos sobre ela; e que agora queremos desenhar a mesma curva depois de uma rotação. Podemos aplicar a rotação a cada um dos 100 pontos, e desenhar os pontos resultantes, ou aplicar a rotação a cada um dos 4 pontos do polígono de controle, calcular novamente os 100 pontos e traçá-los. A primeira estratégia requer que a rotação seja aplicada 100 vezes, e a segunda requer a sua aplicação apenas 4 vezes! É interessante observar que as curvas de Bézier não são invariantes sob transformações projetivas.

## Bibliografia

Fonte: Caps. 4,5 Rogers & Adams - Mathematical Elements for Computer Graphics

## **Capítulo 9**

# **Apostila Modelagem**

O texto sobre modelagem pode ser encontrado no seguinte endereço:

<http://gbdi.icmc.usp.br/documentacao/apostilas/cg/downloads/modelagem.pdf>

# Capítulo 10

## Rendering

Esta apostila está sob revisão e será adicionada nesta versão assim que possível. A versão anterior pode ser encontrada em:

<http://gbdi.icmc.usp.br/~cg/ap10.html>

# Capítulo 11

## Cores e Sistemas de Cores

### 11.1 Percepção de Cor

A cor é o atributo da percepção visual que pode ser descrito através dos nomes usados para identificar as cores, como branco, cinza, preto, amarelo, etc., ou da combinação delas. As diferentes cores, ou espectros luminosos, que podem ser percebidos pelo sistema visual humano correspondem a uma pequena faixa de freqüências do espectro eletromagnético, que inclui as ondas de rádio, microondas, os raios infravermelhos e os raios X, como mostrado na Figura 11.1.

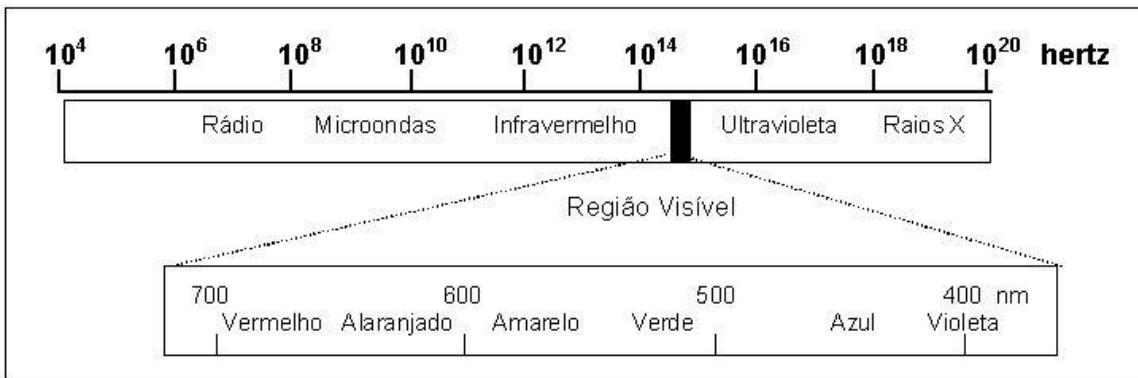


Figura 11.1: Espectro eletromagnético [Gro94].

A freqüência mais baixa do espectro visível corresponde à cor vermelha ( $4.3 \times 10^{14}$  hertz) e a mais alta à cor violeta ( $7.5 \times 10^{14}$  hertz). Os valores de freqüência intermediários correspondem a cores que passam pelo alaranjado e amarelo e por todas as outras cores, até chegar nos verdes e azuis. As cores são ondas eletromagnéticas descritas pelo seu comprimento de onda ( $l$ ) e especificadas, tipicamente, em nanômetros (nm).

Os comprimentos de onda maiores possuem distâncias focais maiores e, consequentemente, requerem maior curvatura da lente do olho para serem focalizados, (a cor vermelha possui a maior distância focal e a azul, a menor). A utilização simultânea de cores localizadas em extremos opostos do espectro fazem com que a lente altere o seu formato constantemente, causando cansaço no olho. A visão estereoscópica da cor é um efeito relacionado ao processo de focalização da imagem na retina, que faz com que cores puras localizadas à mesma distância do olho pareçam estar a distâncias diferentes; por exemplo, a cor vermelha parece estar mais próxima e a cor azul parecer mais distante.

Uma fonte de luz (como, por exemplo, o sol ou uma lâmpada) emite em todas as freqüências do espectro visível, produzindo a luz branca que incide sobre um objeto. Parte dessa luz é absorvida e a outra parte é refletida, determinando a cor resultante do objeto. Quando há predominância das freqüências baixas, diz-se que o objeto é vermelho, ou que a luz percebida possui uma freqüência dominante (ou um comprimento de onda dominante) na freqüência baixa do espectro. A freqüência dominante também é chamada de matiz ou, simplesmente, de cor da luz.

O matiz é o atributo de uma sensação visual que faz com que uma área pareça ser similar a uma ou 2 das cores percebidas vermelha, amarela, laranja, azul, púrpura [Hun78]. A partir dessa definição, pode-se diferenciar a cor cromática, que possui matiz, da acromática, que é desprovida de matiz. As características da luz são descritas através de propriedades como o matiz e as sensações de brilho e saturação. O matiz é usado para dar um nome a uma cor, o brilho corresponde ao grau de luminância

de uma cor em relação à luminância de outra ou em relação ao fundo, e a saturação é a pureza aparente de um matiz. Quanto maior o domínio de um comprimento de onda, maior é a sua saturação. As cores preta, branca e cinza possuem saturação uniforme em todos os comprimentos de onda e, por isso, são diferenciadas apenas pelo brilho. As propriedades de saturação e de matiz de uma cor são referenciadas como cromaticidade.

Algumas pessoas possuem uma anomalia, denominada daltonismo, que impede a distinção de uma ou mais cores. O daltonismo se deve a um defeito na constituição dos cones e está vinculada ao sexo. Ele atinge cerca de 8% dos homens, e apenas 0.5% das mulheres. Esse defeito pode se manifestar em 1, 2 ou nos 3 tipos de receptores. Os tricomatas são daltônicos que possuem os 3 sistemas de pigmentos, mas que utilizam os sistemas em proporções diferentes das pessoas normais e das pessoas com o mesmo defeito. Os dicromatas percebem as cores defeituosamente porque combinam apenas 2 sistemas. Os monocromatas percebem apenas graduações de claro e de escuro, pois sua estimulação visual se baseia em um único sistema cromático. Um outro aspecto que também influencia na percepção de cores é o amarelamento das lentes do olho que ocorre com o passar dos anos, fazendo com que as pessoas se tornem menos sensíveis, por exemplo à cor amarela do que à cor azul.

## 11.2 Sistemas de Cores Primárias

O conteúdo desta seção, que descreve alguns sistemas de cores primárias, foi baseado principalmente em Hearn [Hea94], Capítulo 15.

Um sistema de cores é um método que explica as propriedades ou o comportamento das cores num contexto particular. Não existe um sistema que explique todos os aspectos relacionados à cor. Por isso, são utilizados sistemas diferentes para ajudar a descrever as diferentes características da cor que são percebidas pelo ser humano. Existem vários sistemas de cores, sendo que serão apresentados apenas alguns dos principais: o XYZ, o RGB, o HSV e o HLS.

As cores primárias são as 2 ou 3 cores que um sistema utiliza para produzir outras cores. As cores podem ser produzidas a partir de uma combinação das primárias, ou então, da composição de 2 combinações. O universo de cores que podem ser reproduzidas por um sistema é chamado de espaço de cores (color space ou color gamut). Alternativamente, um espaço de cores (color space) pode ser definido como uma representação visual de um modelo de cores, como o cubo definido pelas componentes do modelo RGB, ou o cone definido pelo modelo HSV. Não existe um conjunto finito de cores primárias que produza todas as cores visíveis, mas sabe-se que uma grande parte delas pode ser produzida a partir de 3 primárias.

O estudo da utilização de 3 fontes de luz espectral para a geração de cores é chamado de colorimetria, e tem como um de seus objetivos determinar espaços de cor perceptualmente uniformes. Um espaço de cores (ou sistema de cores) perceptualmente uniforme é um no qual distâncias perceptuais iguais separam todas as cores. Por exemplo, a escala de cinzas do espaço deve transmitir uma transição suave entre o preto e o branco. A definição de um espaço de cores uniforme é feita através de medições empíricas obtidas sob condições experimentais rigidamente controladas - as condições do ambiente e outros parâmetros importantes devem ser mantidos constantes, como o tamanho das amostras de cores, o espaçamento entre as amostras, a luminância e cromaticidade do fundo e da luz ambiente. Apesar dessa limitação, os espaços perceptuais de cores fornecem ferramentas adequadas para a solução de problemas como a compressão de imagens (para decidir o nível de codificação da informação de cor) e pseudo-coloração<sup>1</sup> (para mapear as cores da imagem em um conjunto com espaçamento perceptual máximo).

Os sistemas de cores podem ser aditivos ou subtrativos. Nos modelos aditivos (por exemplo, RGB e XYZ), as intensidades das cores primárias são adicionadas para produzir outras cores. A Figura 11.2 ilustra a demonstração do funcionamento desses modelos através da sobreposição de círculos coloridos.

Pode-se pensar que o branco é a mistura das intensidades máximas das 3 cores primárias aditivas (vermelha, verde e azul). Os matizes intermediários (amarelo, turquesa e magenta) são obtidos através da combinação das intensidades máximas de 2 cores.

Nos modelos subtrativos (por exemplo, o CMY), as cores são geradas subtraindo-se o comprimento da onda dominante da luz branca, por isso, a cor resultante corresponde à luz que é refletida. A Figura 11.3 ilustra a demonstração do funcionamento desses modelos através da sobreposição de círculos coloridos.

Pode-se pensar que o preto é a combinação das 3 cores subtrativas (turquesa, magenta e amarela). A quantidade de preto numa cor é indicada pela diferença entre o branco e a intensidade máxima das 3 cores primárias aditivas. E, da mesma forma, a quantidade de branco numa cor é indicada pela diferença entre o preto e a intensidade mínima das 3 cores primárias aditivas.

<sup>1</sup>A coloração “falsa” (“pseudocoloring”) ocorre quando as cores “verdadeiras” de uma imagem são mapeadas em outro conjunto de cores, ou quando uma imagem adquirida em tons de cinza é colorida.

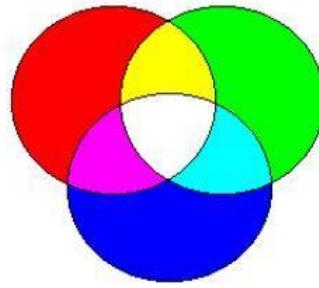


Figura 11.2: Ilustração da mistura de cores aditivas [For94].

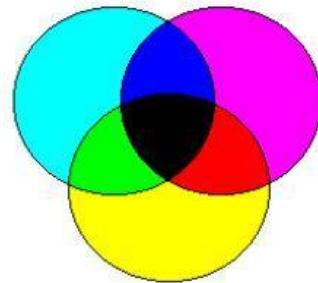


Figura 11.3: Ilustração da mistura de cores subtrativas [For94].

As cores puras e saturadas não representam toda a classe de cores possíveis, existem ainda os tints, shades e tones que correspondem, respectivamente, às cores obtidas através da adição de branco, preto e cinza às cores saturadas, causando uma alteração no efeito da cor. A Figura 11.4<sup>2</sup> ilustra de tints, shades e tones obtidos a partir da cor vermelha.

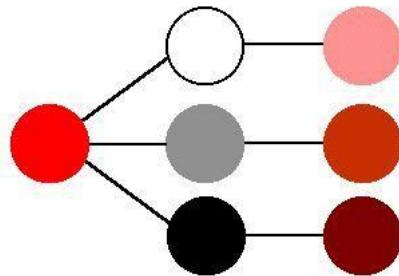


Figura 11.4: Ilustração da obtenção de tints, shades e tones.

A adição de branco clareia uma cor e cria um tint (por exemplo, adicionando branco ao vermelho para obter a cor pink). A adição de preto escurece uma cor e cria um shade (por exemplo, adicionando preto ao vermelho para obter um castanho-avermelhado (maroon)). Além disso, a adição de cinza reduz o brilho de uma cor e cria um tone. Uma composição monocromática é formada inteiramente de tints, shades e tones da mesma cor.

### 11.3 Modelo XYZ

O sistema *XYZ* de cores primárias da CIE (Comissão Internacional de Iluminação) é um sistema aditivo que descreve as cores através de 3 cores primárias virtuais *X*, *Y* e *Z*. Esse sistema foi criado devido à inexistência de um conjunto finito de cores primárias que produza todas as cores visíveis possíveis. Nesse sistema, as cores  $C_l$  podem ser expressas pela seguinte equação:

$$C_l = x.X + y.Y + z.Z \quad (11.1)$$

---

<sup>2</sup>reproduzida de <http://www.contrib.andrew.cmu.edu:8001/usr/dw4e/color/tints.html>.

em que  $X$ ,  $Y$  e  $Z$  especificam as quantidades das primárias padrões necessárias para descrever uma cor espectral. A normalização dessa quantidade em relação à luminância ( $X + Y + Z$ ) possibilita a caracterização de qualquer cor. As cores desse sistema podem ser expressas como combinações das quantidades normalizadas abaixo:

$$x = \frac{X}{X + Y + Z} \quad y = \frac{Y}{X + Y + Z} \quad z = \frac{Z}{X + Y + Z} \quad (11.2)$$

com  $x + y + z = 1$ . Assim, qualquer cor pode ser definida apenas pelas quantidades de  $x$  e  $y$  que, por dependerem apenas do matiz e da saturação, são chamadas de coordenadas de cromaticidade. A descrição completa de uma cor é dada pelas coordenadas de cromaticidade e pelo valor de um dos 3 estímulos originais, normalmente do  $Y$ , que contém a informação de luminância. Essa descrição possibilita a obtenção das quantidades de  $X$  e  $Z$  com as equações abaixo:

$$X = \frac{x}{y} Y \quad Z = \frac{z}{y} Y \quad (11.3)$$

onde  $z = 1 - x - y$ .

O sistema XYZ é formado por cores imaginárias que são definidas matematicamente. Nesse sistema, as combinações de valores negativos e outros problemas relacionados à seleção de um conjunto de primárias reais são eliminados. As coordenadas de cromaticidade  $x$  e  $y$  permitem representar todas as cores num gráfico bidimensional. O traçado dos valores normalizados de  $x$  e  $y$  para as cores no espectro visível resulta na curva ilustrada na Figura 11.5.

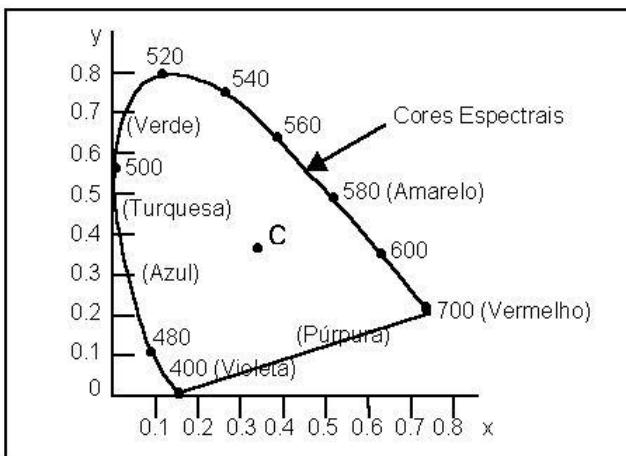


Figura 11.5: Diagrama de cromaticidade do CIE [Hea94].

Os pontos que representam as cores puras no espectro eletromagnético são rotulados de acordo com os seus comprimentos de onda e estão localizados ao longo da curva que vai da extremidade correspondente à cor vermelha até a extremidade correspondente à cor violeta. A linha reta que une os pontos espectrais vermelho e violeta é chamada linha púrpura, e não faz parte do espectro. Os pontos internos correspondem a todas as combinações possíveis de cores visíveis, e o ponto C corresponde à posição da luz branca.

Devido à normalização, o diagrama de cromaticidade não representa os valores de luminância. Por isso, as cores com luminâncias diferentes e cromaticidades iguais são mapeadas no mesmo ponto. Através desse diagrama, é possível determinar e comparar os espaços de cores dos diferentes conjuntos de primárias, identificar as cores complementares (2 cores que, somadas, produzem a cor branca) e determinar o comprimento de onda dominante e a saturação de uma cor.

Os espaços de cor são representados no diagrama, ilustrado na Figura 11.6a, através de linhas retas ou de polígonos. Todas as cores ao longo da linha que une os pontos  $C_1$  e  $C_2$  na Figura 11.6a podem ser obtidas através da mistura de quantidades apropriadas das cores correspondentes a esses pontos. A escala de cores para 3 pontos (por exemplo,  $C_3$ ,  $C_4$  e  $C_5$  na Figura 11.6a) é representada por um triângulo cujos vértices são definidos pelas cores correspondentes às 3 posições e inclui cores contidas no interior e nas margens fronteiriças desse triângulo.

Observando o diagrama é possível perceber que nenhum conjunto formado por 3 primárias pode gerar todas as cores, pois nenhum triângulo contido no diagrama abrange todas as cores possíveis. As cores complementares são identificadas por 2 pontos localizados em lados opostos do ponto  $C$  e conectados por uma linha reta. Por exemplo, misturando quantidades apropriadas de 2 cores  $C_1$  e  $C_2$  (Figura 11.6b), obtém-se a luz branca.

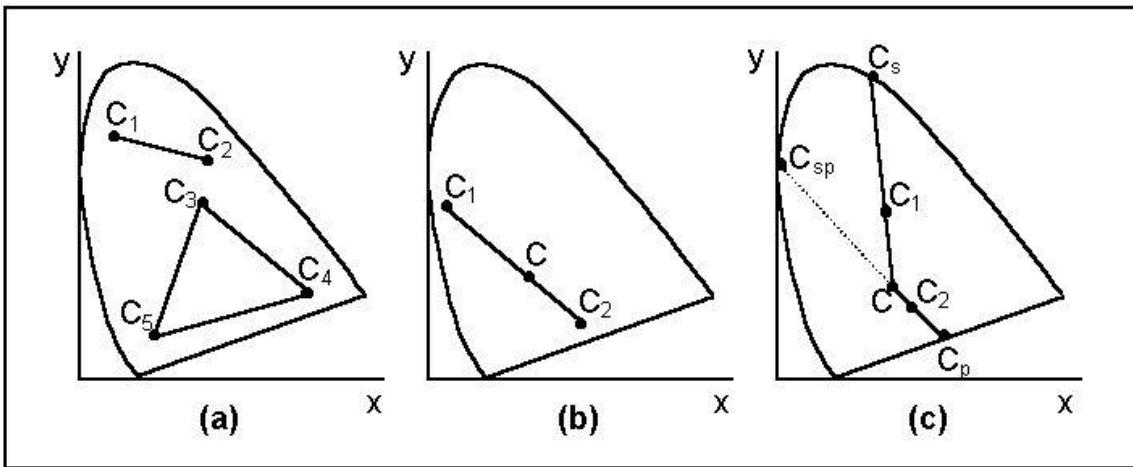


Figura 11.6: Representação de escalas de cor no diagrama de cromaticidade do CIE [Hea94].

A determinação do comprimento de onda dominante de uma cor pode ser feita interpretando-se a escala de cores entre 2 primárias. O comprimento de onda dominante da cor  $C_1$ , representada na Figura 11.6c, é determinado traçando-se uma linha reta que parte do ponto  $C$  passando pelo ponto  $C_1$  e intersectando a curva espectral no ponto  $C_s$ . A cor  $C_1$ , corresponde então, à combinação da luz branca com a cor espectral  $C_s$ , pois  $C_s$  é o comprimento de onda dominante de  $C_1$ . O comprimento de onda dominante das cores que estão entre o ponto  $C$  e a linha púrpura é determinado de outra forma. Traça-se uma linha a partir do ponto  $C$  (Figura 11.6c) passando pelo ponto  $C_2$  e intersectando a linha púrpura no ponto  $C_p$ . Como esse ponto não pertence ao espectro visível, o ponto  $C_2$  é referenciado como sendo uma cor não espectral e o seu comprimento de onda dominante é obtido através do prolongamento da reta até que ela intercepte a curva espectral, no ponto  $C_{sp}$ . As cores não espectrais estão entre púrpura e magenta, e são geradas através da subtração do comprimento da onda dominante (como, por exemplo, o  $C_{sp}$ ) da luz branca.

A pureza de uma cor (por exemplo, de  $C_1$  na figura 11.6c) é determinada através da distância relativa do ponto  $C_1$ , que corresponde à linha reta que vai do ponto  $C$  até o ponto  $C_s$ . Pode-se calcular a pureza do ponto  $C_1$  através da relação  $\frac{d_{c1}}{d_{cs}}$ , onde  $d_{c1}$  representa a distância entre  $C$  e  $C_1$  e  $d_{cs}$  representa a distância entre  $C$  e  $C_s$ . A cor  $C_1$  é cerca de 25% pura porque está situada a aproximadamente  $\frac{1}{4}$  da distância total entre  $C$  e  $C_s$ .

## 11.4 Modelo RGB (Red, Green, Blue)

O sistema RGB de cores primárias também é aditivo e está baseado na teoria dos 3 estímulos (*Tristimulus Color Theory*) proposta por Young-Helmholtz e discutida na seção 2.3. Segundo essa teoria, o olho humano percebe a cor através da estimulação dos 3 pigmentos visuais presentes nos cones da retina, que possuem picos de sensibilidade aproximada nos seguintes comprimentos de onda: 630 nm (Vermelho-Red), 530 nm (Verde-Green) e 450 nm (Azul-Blue). Esse sistema pode ser representado graficamente através do cubo unitário definido sobre os eixos R, G e B, como ilustrado na Figura 11.7.

A origem representa a cor preta, o vértice de coordenadas (1, 1, 1) representa a cor branca, os vértices que estão sobre os eixos representam as cores primárias e os demais vértices representam o complemento de cada cor primária. Cada ponto no interior do cubo corresponde a uma cor que pode ser representada pela tripla  $(R, G, B)$ , com os valores  $R$ ,  $G$  e  $B$  variando de 0 a 1. Os tons de cinza são representados ao longo da diagonal principal do cubo, que vai da origem (ponto correspondente a cor preta) até o vértice que corresponde à cor branca. Cada tom ao longo dessa diagonal é formado por contribuições iguais de cada primária. Logo, um tom de cinza médio entre o branco e o preto é representado por  $(0.5, 0.5, 0.5)$ . As cores  $C_l$  desse sistema podem ser expressas na forma:

$$C_l = r.R + g.G + b.B \quad (11.4)$$

A resposta do olho aos estímulos espetrais não é linear e, por isso, algumas cores não podem ser reproduzidas pela sobreposição das 3 primárias. Isso significa que algumas cores existentes na natureza não podem ser mostradas nesse sistema. Por isso, um fenômeno natural colorido como, por exemplo, de formação de rochas, não pode ser reproduzido com precisão (conforme discutido em Wolf [Wol93]).

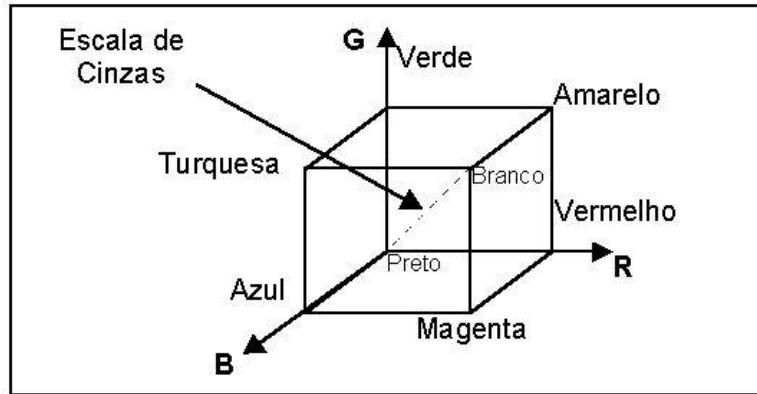


Figura 11.7: Cubo do RGB [Fol96].

## 11.5 Modelo HSV (Hue, Saturation, Value)

O sistema HSV utiliza descrições de cor que são mais intuitivas do que combinações de um conjunto de cores primárias e, por isso, é mais adequado para ser usado na especificação de cores em nível de interface com o usuário. A cor é especificada através de uma cor espectral e das quantidades de branco e preto que serão adicionadas para a obtenção de shades, tints e tones diferentes. A representação gráfica tridimensional do sistema HSV é um cone de 6 lados derivado do cubo RGB, mostrado na figura 11.8.

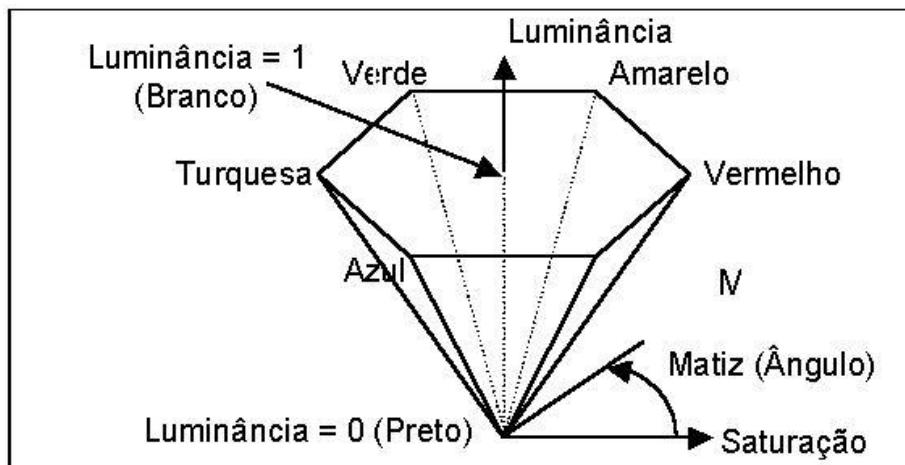


Figura 11.8: Cone hexagonal do HSV [Fol96].

Os parâmetros de cor utilizados nesse sistema são o matiz (*hue*), a saturação (*saturation*) e a luminância (*value*). Os vários matizes estão representados na parte superior do cone, a saturação é medida ao longo do eixo horizontal e a luminância é medida ao longo do eixo vertical, que passa pelo centro do cone. O matiz, que corresponde às arestas ao redor do eixo vertical, varia de  $0^\circ$  (vermelho) a  $360^\circ$ , e o ângulo entre os vértices é de  $60^\circ$ . A saturação varia de 0 a 1 e é representada como sendo a razão entre a pureza de um determinado matiz e a sua pureza máxima ( $S = 1$ ). Um determinado matiz possui  $\frac{1}{4}$  de pureza em  $S = 0.25$ . Quando  $S = 0$  tem-se a escala de cinzas. A luminância varia de 0 (no pico do cone), que representa a cor preta, a 1 (na base), onde as intensidades das cores são máximas.

## 11.6 Modelo HLS (Hue, Lightness, Saturation)

O sistema HLS<sup>3</sup> também é baseado em parâmetros mais intuitivos para a descrição de cores.

A representação gráfica tridimensional desse sistema é um cone duplo, como mostra a figura 9. Os 3 parâmetros de cor utilizados são o matiz (*hue*), a luminosidade (*lightness*<sup>4</sup>) e a saturação (*saturation*).

<sup>3</sup>Outros autores denominam esse sistema como *Hue, Luminance, Saturation* [Rog93a].

<sup>4</sup>Luminosidade (*lightness*): Segundo Foley [Fol96], esse termo está relacionado à noção acromática da intensidade percebida de um objeto que reflete luz. Brilho (*brightness*) é usado no lugar de luminosidade para denotar a intensidade percebida de um objeto que emite luz.

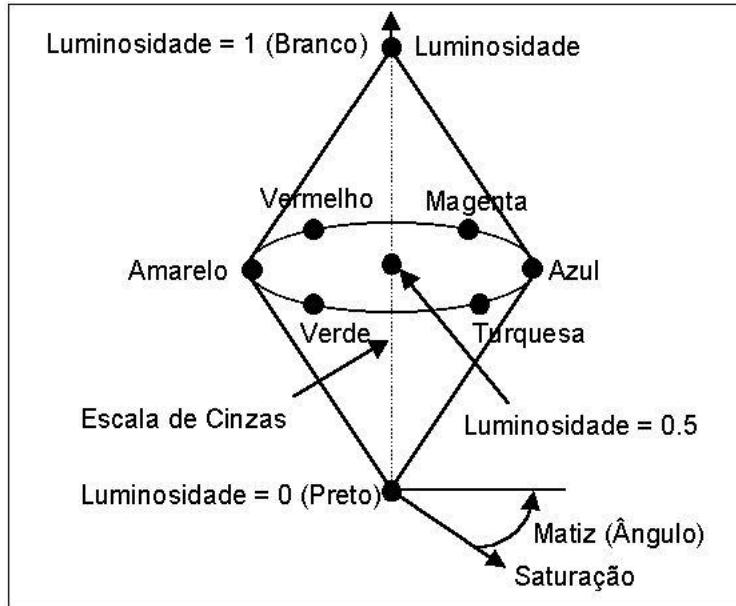


Figura 11.9: Cone duplo do HLS [Fol96].

O ângulo em relação ao eixo vertical varia de  $0^\circ$  (matiz azul) a  $360^\circ$  em intervalos de  $60^\circ$  e especifica um matiz. O eixo vertical corresponde à luminosidade e varia de 0 (preto) a 1 (branco) e é onde se encontra a escala de cinzas. A saturação varia de 0 a 1, e os matizes puros são encontrados no plano onde a luminosidade é igual a 0.5 e a saturação é igual a 1. Quanto menor o valor da saturação menor é a pureza do matiz; e quando a saturação é igual a 0, tem-se a escala de cinzas.

Os sistemas HLS e HSV permitem que se pense em termos de cores mais “claras” e mais “escuradas”. As cores são especificadas através de um ângulo, e os diversos *shades*, *tints*, e *tones* de cada cor são obtidos através do ajuste do brilho ou luminosidade e da saturação. As cores mais claras são obtidas através do aumento do brilho ou da luminosidade e as cores mais escuradas pela diminuição dos mesmos. As cores intermediárias são obtidas através da diminuição da saturação.

## Capítulo 12

# Processamento Digital de Imagens

### 12.1 Introdução

A área de Processamento Digital de Imagens tem crescido consideravelmente nas últimas duas décadas, com a utilização de imagens e gráficos em uma grande variedade de aplicações, aliado ao fato de que a tecnologia de construção de computadores também tem se aprimorado, possibilitando a utilização de sistemas mais eficientes e mais baratos. Muitas áreas vêm utilizando Sistemas de Processamento Digital de Imagens, tais como: reconhecimento de padrões (indústria), medicina, agricultura, pesquisas espaciais, meteorologia, etc.

### 12.2 Considerações Sobre Imagens

Uma imagem contínua, é digitalizada através de dois processos:

- *Amostragem (Sampling)*: digitalização em termos espaciais (tamanho da grade de digitalização);
- *Quantização*: digitalização em termos de amplitude (nº de níveis de cinza).

A Figura 12.1, ilustra o processo de quantização, para vários tamanhos de grade. A Figura ??, apresenta uma imagem de 512x512 pixels, quantizada de várias formas.

Matematicamente, Imagem Digital pode ser definida como uma função bidimensional  $A(x, y)$  definida em uma certa região do plano.

$$A : [0, r] \times [0, s] \rightarrow [0, t] \quad (12.1)$$

Assim a imagem é definida num retângulo  $[0, r] \times [0, s]$ , e os valores tomados estão contidos no intervalo  $[0, t]$ . Ao valor  $A(x, y)$  no ponto  $(x, y)$  dá-se o nome de *Nível de cinza*.

Imagens são geralmente definidas como uma grande coleção de dados, que estão frequentemente dispostos em forma de uma matriz ou alguma outra estrutura de dados discreta.

Uma imagem, é representada através de um conjunto de valores, onde cada valor é um número que descreve os atributos de um pixel na imagem. Usualmente estes números são inteiros positivos, que representam a intensidade na escala de tons de cinza de um ponto da imagem, ou a cor associada ao ponto pela “look-up table”, em sistemas que comportam este procedimento. Quando utiliza-se escalas de tons de cinza, usualmente associa-se a tonalidade de cinza mais escura, o preto, ao nível zero da escala de cinza, e o mais claro, o branco, ao maior valor permitido na escala. Ou seja, se cada pixel é representado em um sistema de 8-bits, o preto é associado ao valor zero, e o branco ao valor 255.

As dimensões do conjunto de valores que especifica a imagem são chamadas de largura e altura da imagem, e o número de bits associado com cada pixel da matriz é chamado de profundidade da imagem. A profundidade da imagem está associada ao número de planos de memória em que uma imagem é mapeada, sendo que cada bit da palavra que armazena o valor do pixel reside em um plano da imagem (“frame buffer”). Na Figura 12.3 mostra-se um esquema de um pixel p com profundidade 8.

Através de Processamento Digital de Imagens, pode-se analisar e modificar imagens por computador. Os objetivos deste processamento são:

1. Extrair informação da imagem, ou seja, reconhecer elementos que compõem a imagem, permitir medir elementos contidos na imagem, classificá-los e compará-los com algum modelo colocado.

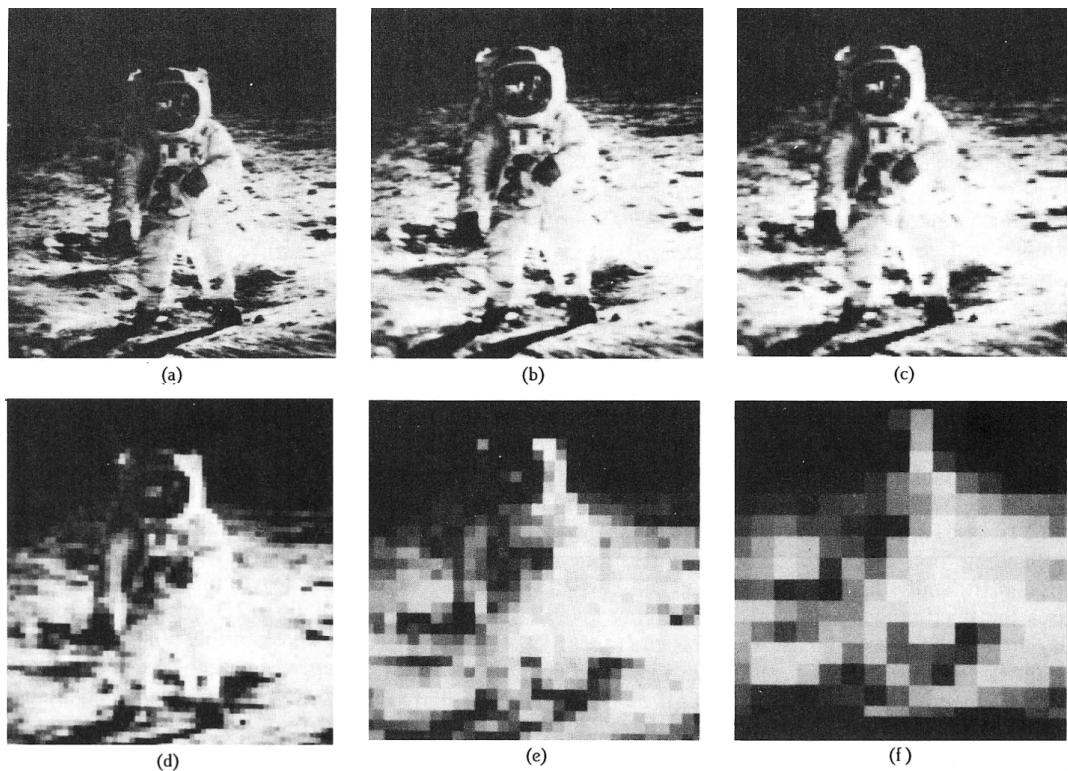


Figura 12.1: Efeitos sobre uma imagem, de se reduzir o tamanho da grade de amostragem.

2. Transformar a imagem (aumentando contraste, realçando bordas, corrigindo alguma imperfeição gerada na imagem durante o processo de aquisição), de forma que a informação seja mais facilmente discernível por um analista humano.

### 12.3 Tabelas “Look-up”

Em sistemas que permitem trabalhar com cores, cada cor é uma composição de três cores primárias: vermelho, verde e azul (do inglês, **RGB**). E os níveis de cinza também são uma composição das três cores, sendo que todas entram em partes iguais. Logo, o número de bits utilizados para representar o valor do pixel é dividido entre as três cores. Assim, se um pixel possui profundidade 1 (um bit por pixel), o pixel poderá ser pintado apenas com duas cores ( $2^1$ ) (por exemplo, em preto e branco). Se o pixel utiliza dois bits, poderá produzir no máximo quatro cores ( $2^2$ ), e assim por diante. Usualmente 6 bits são suficientes para permitir a visualização da imagem de forma que não se perceba descontinuidade entre as tonalidades de cinza, mas 8 ou mais bits podem ser necessários em algumas aplicações. Assim, para monitores coloridos, sugere-se que três vezes a quantidade de bits necessária é um bom valor, ou seja, 8 bits para cada cor primária aditiva (vermelha, verde e azul - RGB), produzindo 24 bits por pixel.

Em muitos sistemas gráficos é desejável poder-se alterar rapidamente a coloração da imagem sem alterar a sua forma. Assim permite-se alterar a aparência visual de uma imagem, sem na realidade alterar os próprios dados que a definem. Por exemplo, mostrar todos os pixels abaixo de um certo valor com o preto, ou mesmo a partir de uma imagem monocromática criar uma imagem em pseudo-cor.

Um dispositivo que facilita grandemente essa manipulação com cores, sem comprometer a eficiência da operação de visualização de imagens são as tabelas de Busca (“**Look-up Table**” - **LUT**), que os controladores de monitores por varredura vem utilizando. A tabela “look-up” possui tantas entradas quanto o número de valores armazenados pelo pixel, ou seja, se o pixel tem profundidade 8, a LUT terá 256 ( $2^8$ ) entradas. Assim, o valor do pixel não indica a cor em que ele é mapeado, mas sim a entrada na LUT onde a cor está armazenada. A Figura 12.4 ilustra o procedimento de mapeamento de um pixel através da LUT.

### 12.4 Tipos de Manipulação de Imagens

Na análise de imagens, a entrada do processamento é uma imagem, enquanto a saída é uma descrição não pictórica da imagem. Este processo pode ser entendido como de “redução de dados”, em que se diminui

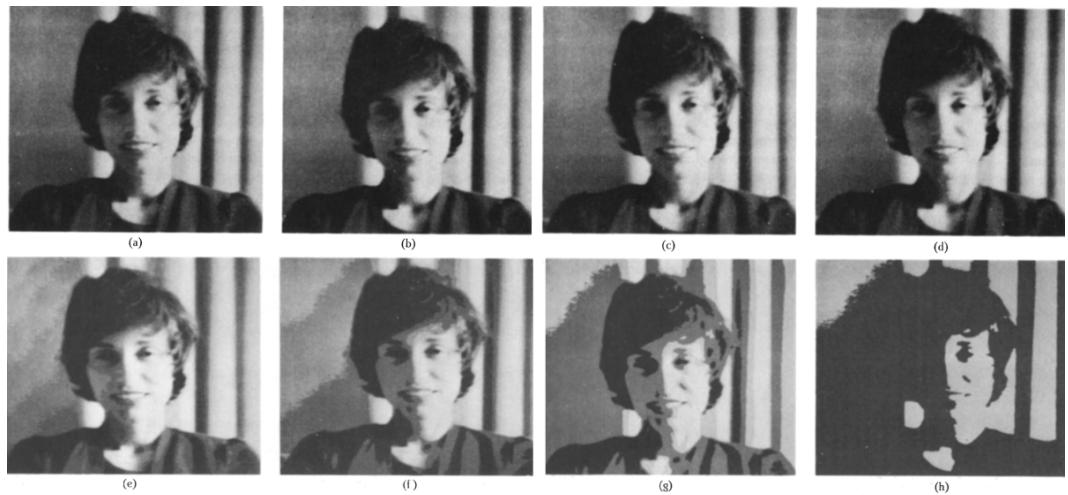


Figura 12.2: Uma imagem de 512 x 512 pixels, quantizada em (a) 256 níveis, (b) 128 níveis, (c) 64 níveis, (d) 32 níveis, (e) 16 níveis, (f) 8 níveis, (g) 4 níveis e (h) 2 níveis de cinza.

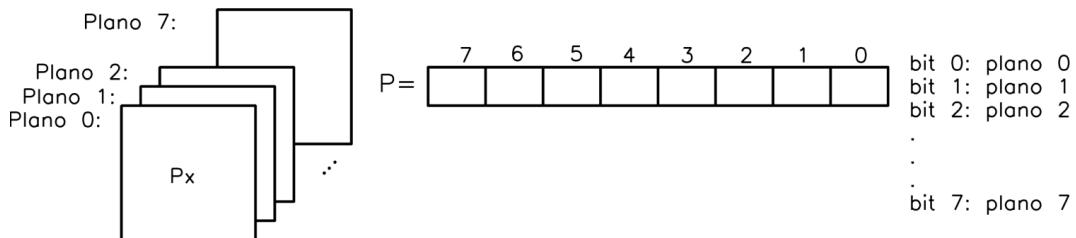


Figura 12.3: Representação de um pixel  $P$  com 8-bits (8 planos) de profundidade.

o volume dos dados, mas mantém-se o conteúdo da informação relevante para uma dada aplicação.

Na manipulação de imagens em geral, tanto as entradas quanto as saídas são imagens. Duas grandes classes de transformações são:

- *Transformações Radiométricas*: onde os valores de níveis de cinza dos pontos da imagem são alterados, sem modificação de sua geometria.
- *Transformações Geométricas*: onde a geometria da imagem é alterada, mantendo-se o máximo possível os valores dos níveis de cinza.

As transformações radiométricas e geométricas podem ser feitas com a finalidade de eliminar distorções da imagem, introduzidas geralmente pelo próprio sistema de aquisição de imagens (“restauração”), ou enfatizar certas características da imagem (“realce”).

## 12.5 Transformações Radiométricas

Pode-se ter basicamente dois tipos de transformações ou operações radiométricas:

- Restauração - que visa corrigir alguma distorção sofrida pela imagem;
- Realce - que procura enfatizar alguma característica de interesse da imagem. Por vezes estes dois objetivos produzem resultados coincidentes.

É o caso, por exemplo, quando uma imagem sofreu uma distorção que diminuiu seu contraste; uma transformação que realce as bordas dos objetos pode, de fato, restaurá-la.

Embora muitas das técnicas de restauração e realce sejam as mesmas (por exemplo, filtragem), os objetivos e enfoques divergem num e outro caso. O procedimento geral da restauração é a modelagem do processo de distorção para tentar invertê-lo. No realce esta preocupação não existe, pois no realce as técnicas utilizadas são, na maioria, heurísticas, não havendo compromisso com a imagem original.

As transformações radiométricas podem também ser classificadas quanto ao seu grau de abrangência, em:

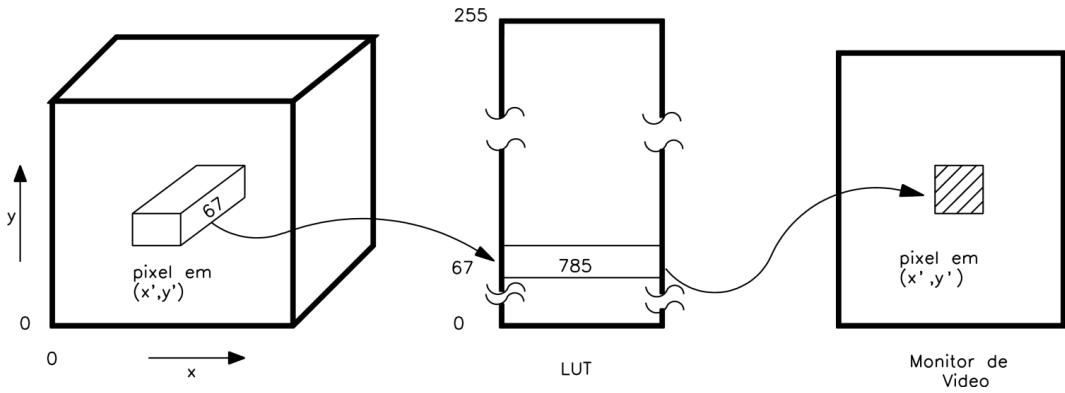


Figura 12.4: Organização de uma LUT, para um sistema de pixels de 8 bits.

1. **Operações Pontuais:** onde o nível de cinza de um ponto na imagem transformada depende só do nível de cinza do ponto da imagem original (ou nas imagens originais se houver mais de uma imagem de entrada (“frames”)).
2. **Operações Locais:** onde o novo nível de cinza de um ponto depende não só de seu antigo nível de cinza, mas também dos níveis de cinza de seus pontos vizinhos.

### 12.5.1 Operações Pontuais sobre Imagens

Algoritmos que realizam Operações Pontuais percorrem a imagem e utilizam o valor do pixel associado a cada ponto, modificando a escala de tonalidades de cinza de uma imagem. Por exemplo caso deseja-se tornar uma imagem mais brilhante, pode-se adicionar algum valor a todos os pontos da imagem. Por exemplo, em um sistema de 8-bits por pixel, os valores associados vão de 0 a 255, pode-se definir uma função  $brilho(p) = p + 40$ , cuja função é tornar a imagem mais “clara”. Através de operações pontuais pode-se também corrigir sombras que porventura tenham surgido junto com o processo de aquisição da imagem. Ou seja, define-se uma função cujo efeito reverta a operação de sombreamento indesejável, porém isto requer que se estime as funções de sombreamento.

Assim, uma operação pontual que toma uma imagem  $A(x, y)$  e produz outra imagem  $B(x, y)$ , pode ser definida matematicamente como:

$$B(x, y) = f[A(x, y)] \quad (12.2)$$

#### Histogramas de Intensidades

O **Histograma de Intensidades** indica para cada nível de cinza da imagem a quantidade de pontos mapeados com tal nível. Ele contém uma informação global sobre os “objetos” da imagem. Se todos os pontos da imagem são de um mesmo objeto, o histograma mostra a probabilidade condicional  $p(z/\text{objeto})$  de um ponto possuir um dado nível de cinza  $z$ , sendo que o ponto pertence ao objeto. A Figura 12.5 apresenta uma imagem e seu histograma de intensidades associado.

Através de Histogramas de Intensidades pode-se obter uma maneira simples para se manipular o contraste de imagens. Dessa forma, pode-se definir as “janelas” de intensidade que se quer manipular. Por exemplo, se ao construir o histograma de uma imagem vê-se que não se está utilizando toda a abrangência dos níveis de cinza, resultando uma imagem com pouco contraste, pode-se forçar que ela seja mapeada usando todos os níveis de cinza, ou uma faixa deles (“janela”), ou mesmo aumentar o contraste em apenas uma região dos níveis de intensidade da imagem [Gonzalez and Wintz, 1987].

Suponha-se uma imagem  $A(x, y)$  e seu histograma  $H_1(A)$ . Se um ponto  $p$  da imagem está mapeado com o nível  $N$  dentro da faixa de valores  $\Delta_1$ , cujo nível mais baixo é  $min_1$  e o mais alto é  $max_1$ , e através de manipulação no histograma deseja-se transformá-lo para o valor  $p'$  com nível  $N'$ , o qual estará dentro da faixa  $\Delta_2$  cujo valor mínimo é  $min_2$  e máximo  $max_2$ , a função de transformação será:

$$f(N) = \frac{(N - min_1)(max_2 - min_2)}{(max_1 - min_1)} + min_2 \quad (12.3)$$

$$N' = f(N) \quad (12.4)$$

A Figura 12.6 ilustra a transformação do histograma  $H_1(A)$  da imagem A para o histograma  $H_2(A)$ .

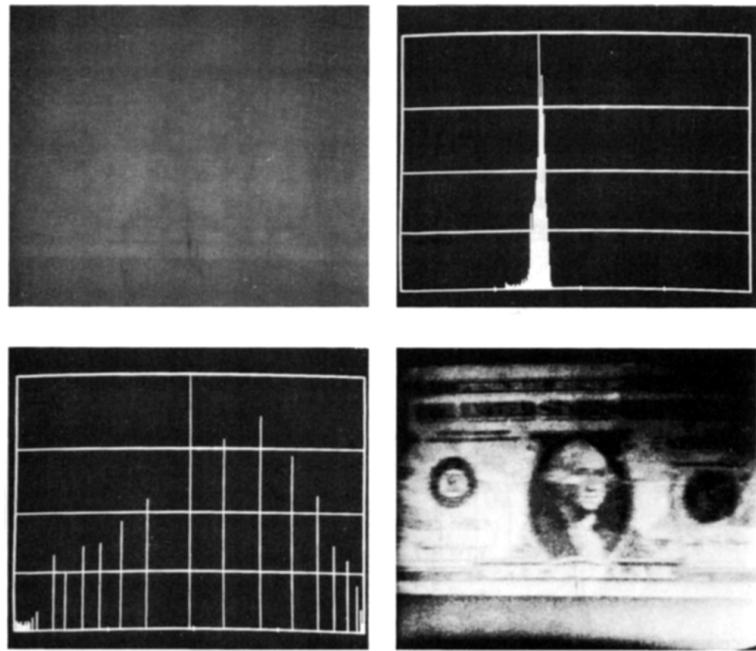


Figura 12.5: Imagens e histogramas de intensidade.

Esse procedimento de manipulação do histograma para ampliar ou diminuir o contraste de uma imagem é mostrado na Figura 12.7. Na parte (a) apresenta-se um histograma de uma imagem, na qual especifica-se uma janela de visualização, e na parte (b) o histograma da mesma imagem depois da manipulação do histograma.

### Operações Algébricas

Por definição, **operações Algébricas**, são aquelas que produzem uma imagem que é a soma, diferença, produto ou quociente pixel a pixel de suas imagens. No caso de soma ou multiplicação mais de duas imagens podem estar envolvidas.

As quatro operações algébricas de processamento de imagens podem ser explicitadas:

$$C(x, y) = A(x, y) + B(x, y) \quad (12.5)$$

$$C(x, y) = A(x, y) - B(x, y) \quad (12.6)$$

$$C(x, y) = A(x, y) \times B(x, y) \quad (12.7)$$

$$C(x, y) = A(x, y) / B(x, y) \quad (12.8)$$

onde:  $A(x, y)$  e  $B(x, y)$  são imagens de entrada e  $C(x, y)$  é a imagem resultante.

A seguir são apresentadas algumas aplicações imediatas para as operações algébricas:

1. Uma importante utilização de adição de imagens, é para se obter a média de múltiplas imagens de uma mesma cena. Isto é eficazmente utilizado para reduzir os efeitos de ruídos aleatórios aditivos. Pode também ser utilizado para colocar o conteúdo de uma imagem sobrepondo outra.
2. Subtração de imagens pode ser utilizada para remover algum padrão indesejável presente em uma imagem. A subtração é também utilizada para detectar mudanças entre duas imagens da mesma cena. Por exemplo, pode-se detectar movimento subtraindo-se imagens sequenciais de uma cena. Subtração de imagens pode ser usada para calcular o gradiente, uma função muito usada para detecção de bordas (ver seção 2.4.2.2?). Uma utilização típica em medicina, é para se comparar duas imagens de uma mesma região do corpo. Obtém-se a imagem da região (ou órgão) normal e também após injetar um meio de contraste, adquire outra imagem. Após esta operação, pode-se comparar as duas através de subtração digital. Assim, por exemplo, se o coração é o órgão em apreciação, consegue-se obter uma imagem apenas das artérias e veias.
3. Multiplicação e divisão de imagens podem ser utilizadas, por exemplo, para corrigir possíveis defeitos de um digitalizador em que a sensibilidade do sensor de luz varia conforme a posição dos pontos da imagem. Multiplicar uma imagem por uma “máscara” pode esconder certas regiões da imagem deixando expostos apenas objetos de interesse.

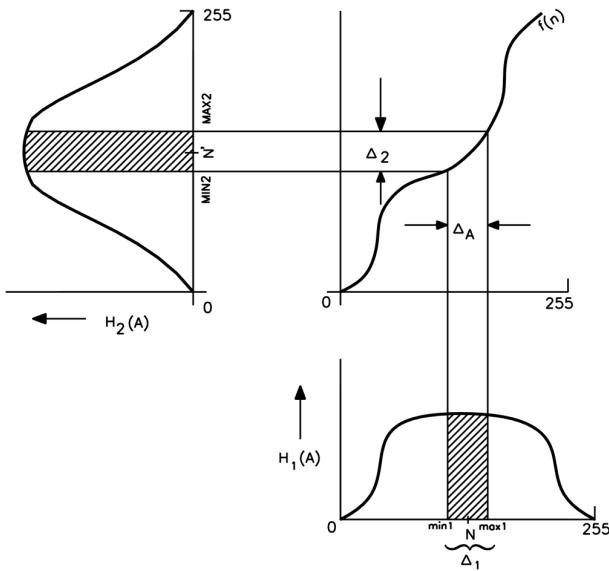


Figura 12.6: Efeito de uma transformação dos níveis de cinza efetuado sobre o histograma.

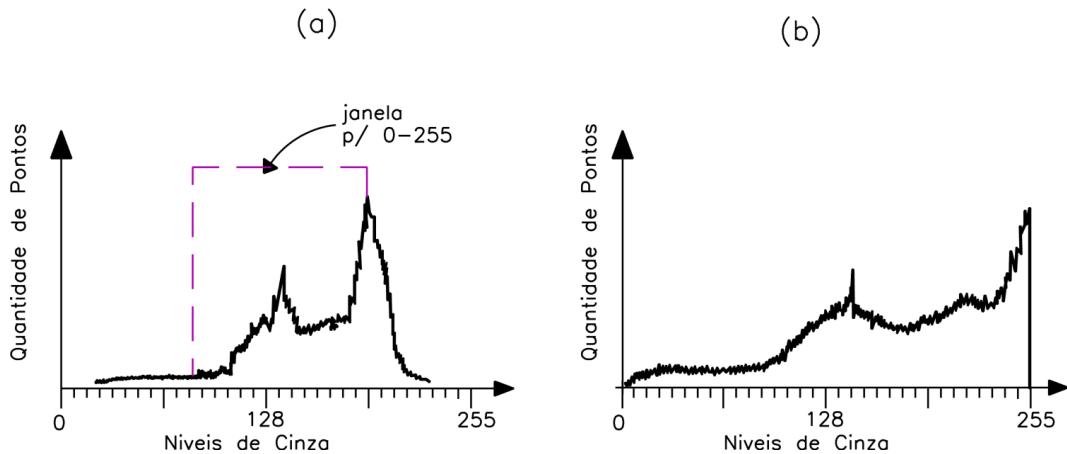


Figura 12.7: Manipulação de janelas de intensidade da imagem através do histograma.

Ao se efetuar operações sobre as imagens, deve-se estar atento para que o novo valor obtido não ultrapasse o domínio de valores, permitido pela amplitude dos níveis de intensidade em que a imagem está mapeada. Se este cuidado não for tomado, pode-se obter valores totalmente inconsistentes, levando-se a uma degeneração da imagem.

### 12.5.2 Operações Locais Sobre a Imagem

Algorítmos que realizam Operações Locais utilizam informação dos valores dos pontos vizinhos para modificar o valor de um ponto, ou mesmo para verificar a existência de alguma propriedade neste ponto da imagem.

Operações Locais são tipicamente utilizadas para filtragem espacial e alteração da própria estrutura da imagem. Elas podem “aguçar” a imagem, acentuando as mudanças de intensidades (através de filtros Passa-Altas), ou “suavizar” a imagem, tornando as mudanças de intensidade menos abruptas (através de filtros Passa-Baixas), ou podem prover outras formas de se melhorar a imagem. Assim, pode-se procurar formas na imagem através de “padrões de busca” (“match”), obter medidas dos elementos da imagem, definir bordas na imagem, remover ruído suavizando a imagem [?].

As transformações locais são divididas em dois grandes grupos:

- **Transformações Lineares:** são aquelas em que os pixels vizinhos ao que está sendo calculado influenciam no novo valor segundo uma combinação linear dos mesmos. Tais operações incluem superposição e convolução de imagens, o que é usualmente utilizado para implementar filtros discretos.
- **Transformações não Lineares:** utilizam outra forma de arranjo que não a linear.

## Operação de Convolução

Um dos algoritmos clássicos de processamento de imagens é aquele que efetua a **Convolução** entre imagens, sendo comumente usado para realizar filtragem espacial e procurar peculiaridades na imagem.

A operação de convolução substitui o valor de um ponto com o valor obtido através de uma operação linear do ponto e de seus vizinhos, sendo que é atribuído “peso” aos vizinhos e ao ponto. Estes “pesos” são colocados em uma matriz, usualmente de dimensão pequena, chamada de **máscara**. Se para obter um novo valor de um ponto deseja-se que apenas os vizinhos imediatos influenciem, escolhe-se uma máscara de dimensão 3x3, onde o ponto base para a geração do novo ponto é colocado no centro da máscara.

Dadas duas imagens  $A(x, y)$  e  $B(x, y)$ , com  $x = 0, 1, \dots, M - 1$  e  $y = 0, 1, \dots, N - 1$ , a convolução (discreta) destas duas imagens é uma terceira imagem  $C(x, y)$  com  $x = 0, 1, \dots, M - 1$  e  $y = 0, 1, \dots, N - 1$ , dada pela seguinte expressão:

$$C(x, y) = \frac{1}{M \cdot N} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A(m, n) \cdot B(x - m, y - n) \quad (12.9)$$

e para os valores da borda da imagem, que se encontram fora dos intervalos  $[0, M - 1]$  e  $[0, N - 1]$ , deve-se utilizar as seguintes relações:

$$A(x, -y) = A(x, N - y) \quad (12.10)$$

$$A(-x, y) = A(M - x, y) \quad (12.11)$$

As relações acima são equivalentes a considerar uma imagem  $A(x, y)$  definida em  $[0, M - 1]$  e  $[0, N - 1]$ , como definida em  $[-M, M - 1]$  e  $[-N, N - 1]$ . O resultado da convolução de  $A$  por  $B$  num ponto  $p$  é na realidade uma média ponderada dos pontos de  $A$ , onde os pesos são dados pela imagem  $B$ .

Como foi mencionado acima, a operação de convolução é a base para efetuar-se filtragem espacial de imagens. A imagem que indica os pesos para a convolução normalmente possui dimensão pequena e ímpar (3, 5, 7 ou 9), pois a operação de convolução envolve uma grande quantidade de cálculos, e quanto maior o máscara maior será o tempo de processamento da operação.

A seguir serão apresentadas algumas máscaras normalmente utilizadas para filtragem espacial [Mascarenhas and Velho, 2002].

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \quad (12.12)$$

Implementa um filtro passa-baixas (de suavização):

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (12.13)$$

Normalmente utilizada para “aguçar” uma imagem, pois como alguns pesos são negativos eles tendem a realçar diferenças.

$$(a) \begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{bmatrix} \quad (b) \begin{bmatrix} -1 & -1 & -1 \\ 1 & -2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (12.14)$$

Apresenta uma máscara de filtro para deteção de borda ao (a) norte e (b) ao sul do ponto.

$$(a) \begin{bmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad (b) \begin{bmatrix} 1 & 1 & -1 \\ 1 & -2 & -1 \\ 1 & 1 & -1 \end{bmatrix} \quad (12.15)$$

Apresenta uma máscara de filtro para deteção de borda a (a) leste e (b) a oeste do ponto. O realce de bordas, independente da direção pode ser obtido através de máscaras como:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} \quad (12.16)$$

Pode-se também utilizar máscaras que contenham algum padrão que se quer procurar na imagem. Neste caso a máscara é normalmente chamada de **tentativa** (“template”), onde o tamanho da máscara depende do padrão a ser procurado.

## Exemplos de Transformações não Lineares

Usualmente as transformações não lineares conseguem uma melhor relação sinal/ruído na imagem produzida do que por transformações lineares. Os filtros lineares utilizados para atenuar os ruídos de uma imagem (passa-baixas) produzem uma imagem “borrada”, ou seja as bordas não ficam mais tão nítidas. Este efeito indesejado pode ser minorado através de transformações (filtros) não lineares.

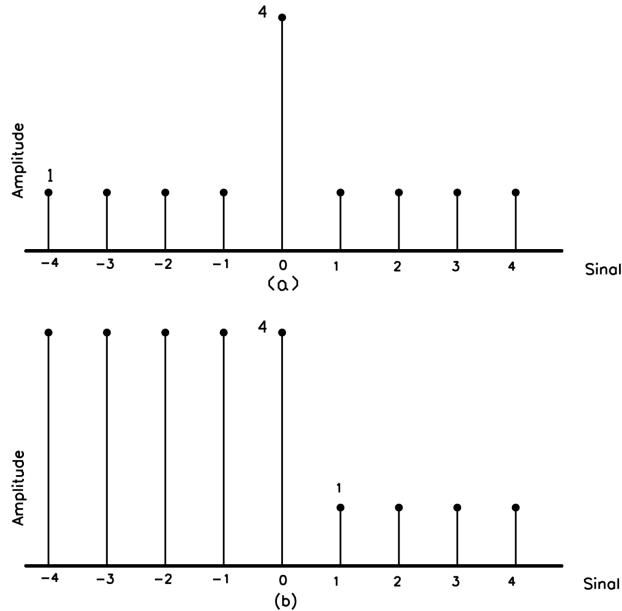


Figura 12.8: Sinais “ruído” (a) e “degrau” (b).

Tome-se como exemplo os sinais unidimensionais dos tipos “ruído” e “degrau” mostrados na Figura 12.9 [Mascarenhas and Velasco, 1989]. Os efeitos de uma transformação linear (filtro) dos tipos média  $[1/3, 1/3, 1/3]$  e diferença  $[-1, 3, -1]$  são mostrados na Figura 12.9. Pode-se notar nestes casos os efeitos de atenuação do ruído (a), “borramento” da borda (b), ampliação do ruído (c) e realce da borda (d). Os efeitos indesejados mostrados na Figura 12.10 (c) e (d) podem ser minorados através de filtros não lineares, como será visto a seguir.

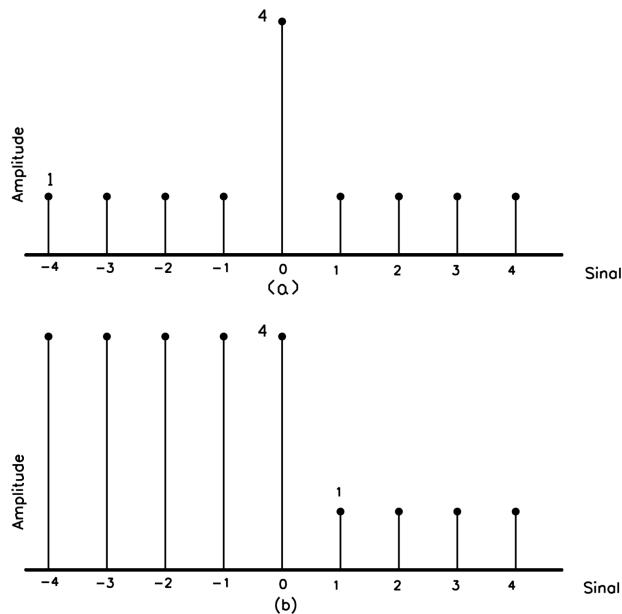


Figura 12.9: Sinais “ruído” (a) e “degrau” (b).

Por exemplo, no filtro da mediana, os pontos da vizinhança do ponto  $(x, y)$  são ordenados, e tomado como novo valor para  $(x, y)$  o valor mediano obtido pela ordenação. O efeito do filtro da mediana (unidimensional) que envolve 3 elementos (o ponto e seus vizinhos mais próximos) é mostrado na Figura

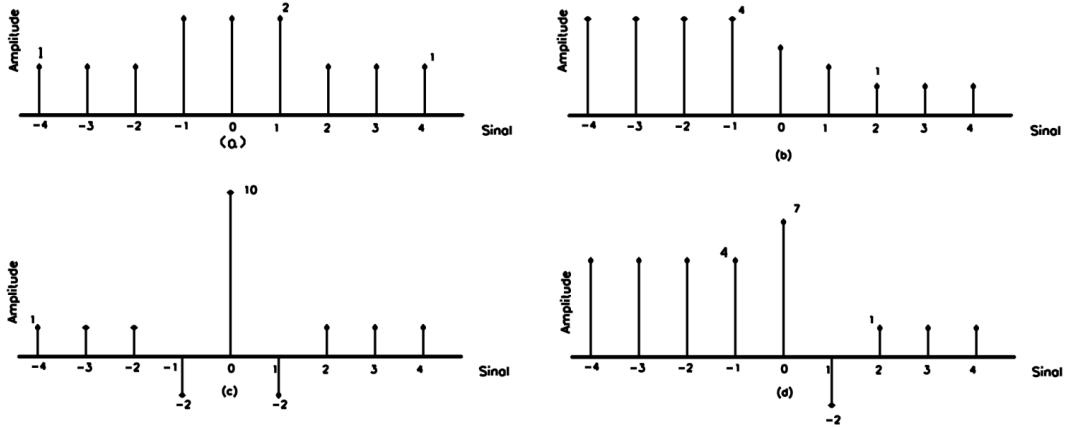


Figura 12.10: Efeitos de filtros lineares ao “ruído” e à “borda”: (a) efeitos de atenuação; (b) borramento da borda; (c) ampliação do ruído; (d) realce da borda.

12.11. Note-se que na parte (a) da figura o ruído foi eliminado (não somente atenuado), enquanto em (b) o degrau permaneceu inalterado.

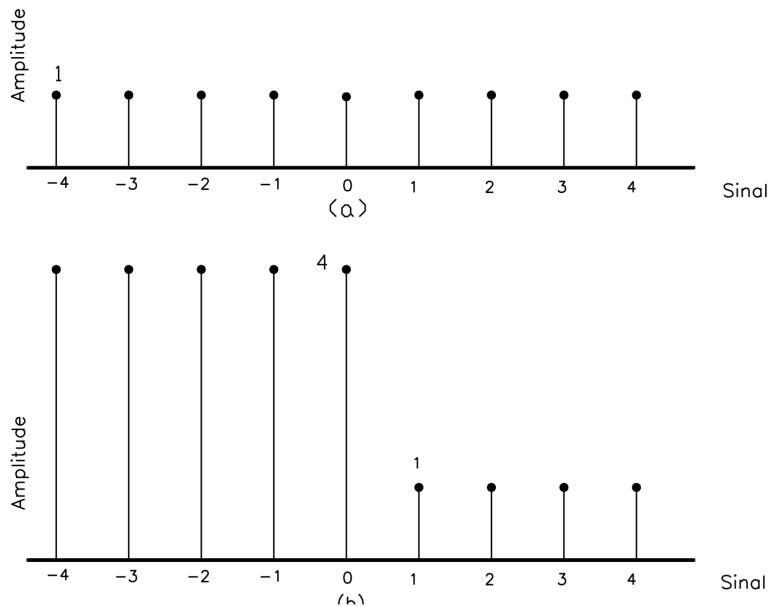


Figura 12.11: Efeito do filtro da mediana sobre o “ruído” e o “degrau”.

É também usual, em vez de se utilizar a mediana da vizinhança, escolher o valor máximo ou o valor de ordem qualquer. Se a imagem é binária, para se obter o valor mínimo entre dois pontos, pode-se utilizar a operação lógica **E** dos valores da vizinhança dois a dois, e para o valor máximo a operação lógica **OU**.

Uma outra alternativa utilizada é tomar-se o valor mais frequente de uma vizinhança, ou seja, utilizar a **Moda**, entre eles.

Pode-se utilizar vizinhança seletiva de um ponto, isto é, tomar a média não em toda a vizinhança, mas sim numa sub-vizinhança que satisfaça alguma propriedade dada, como uniformidade, ou seja, que apresente um valor médio próximo do valor do ponto  $(x, y)$ . O uso do filtro da vizinhança seletiva presume que a imagem seja composta de regiões razoavelmente uniformes. Assim, a seleção da vizinhança procura fazer com que o ponto  $(x, y)$  seja operado só com pontos de sua região.

Pode-se também utilizar filtros não-lineares para detectar, bordas e linhas curvas. Para o realce de bordas o método não-Linear mais simples é o **Operador Gradiente de Roberts (GR)**, dado por [Mascarenhas and Velasco, 1989]:

$$GR(x, y) = [[A(x, y) - A(x - 1, y - 1)]^2 + [A(x, y - 1) - A(x - 1, y)]^2]^{\frac{1}{2}} \quad (12.17)$$

onde  $A(x, y)$  é a imagem dada.

Uma desvantagem do operador de Roberts é sua assimetria, pois, dependendo da direção, certas bordas são mais realçadas que outras, mesmo tendo igual magnitude.

Um operador gradiente para delimitação de bordas mais sofisticado (3x3), é o **Operador de Sobel (GS)**, o qual é frequentemente utilizado como o primeiro passo para algoritmos de visão computacional. Este operador é dado por:

$$GS(x, y) = (X^2 + Y^2)^{\frac{1}{2}} \quad (12.18)$$

onde:

$$\begin{cases} X &= [A(x-1, y+1) + 2.A(x, y+1) + A(x+1, y+1)] - \\ &\quad [A(x-1, y-1) + 2A(x, y-1) + A(x+1, y-1)] \\ Y &= [A(x-1, y-1) + 2.A(x-1, y) + A(x-1, y+1)] - \\ &\quad [A(x+1, y+1) + 2A(x+1, y) + A(x+1, y-1)] \end{cases} \quad (12.19)$$

### Transformações Geométricas

As transformações geométricas mudam o arranjo espacial dos pixels da imagem. Muitas vezes são utilizadas para corrigir distorções causadas pelo processo de aquisição, ou mesmo para efetuar alguma manipulação solicitada pelo usuário. Transformações geométricas típicas são: rotação, translação, ampliação, ou diminuição da imagem. Algoritmos para efetuarem estas operações podem ser apresentados como um conjunto de equações que mapeiam um pixel localizado na posição  $(x, y)$  em um novo endereço,  $(x', y')$ .

Na realidade, em processamento digital de imagens, geralmente são necessários dois algoritmos separados para a maior parte das transformações geométricas. O primeiro é o algoritmo que define a movimentação dos pixels, como ele move a partir da posição inicial para a posição final.

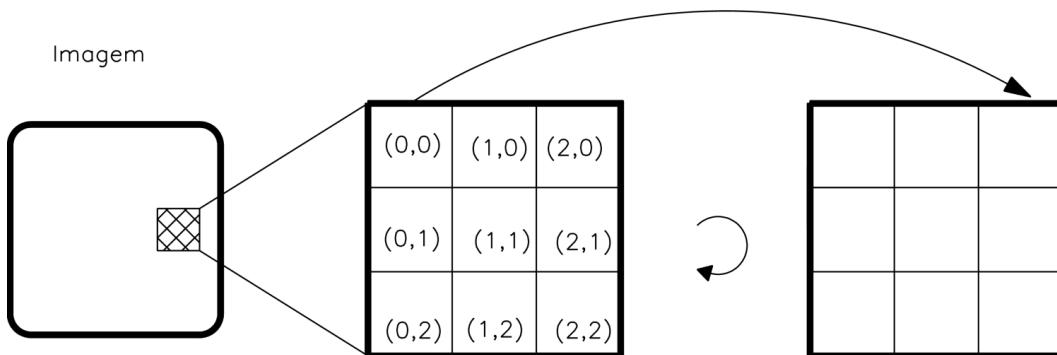


Figura 12.12: Uma transformação geométrica de 90° graus.

Como exemplo, pode-se tomar a imagem ilustrada na Figura 12.12. Ali uma imagem é rotacionada de 90° graus. Um pixel na posição  $(0,0)$  é mapeado segundo a fórmula:

$$\begin{aligned} x' &= (YSIZE - 1) - y \\ y' &= x \end{aligned} \quad (12.20)$$

onde  $YSIZE$  é o número de pixels no eixo  $y$ .

Em muitas aplicações, é necessário preservar a continuidade dos objetos que definem a imagem. Isso pode ser feito transformando-se ponto a ponto a imagem, ou transformando-se apenas pontos chave da imagem, fazendo que os outros pontos acompanhem a transformação por algum outro processo. Pode-se ver que o primeiro processo é geralmente mais oneroso em termos de tempo de computação do que o segundo.

É conveniente especificar matematicamente o relacionamento espacial entre pontos das imagem de entrada e da imagem de saída. A definição geral para operações geométricas é [Castleman, 1979]:

$$g(x, y) = f(x', y') = f(a(x, y), b(x, y)) \quad (12.21)$$

onde  $f(x, y)$  é a imagem de entrada e  $g(x, y)$  é a imagem de saída. As funções  $a(x, y)$  e  $b(x, y)$  especificam as transformações espaciais. Se essas transformações forem contínuas, a continuidade será preservada na nova imagem.

O segundo algoritmo necessário para uma transformação geométrica é um algoritmo para interpolação dos valores dos níveis de cinza. Na imagem de entrada  $f(x, y)$ , os valores do nível de cinza são definidos somente para valores inteiros de  $x$  e  $y$ . Mas na equação geral acima, pode-se ver que o valor do nível de cinza para  $g(x, y)$  pode ser tomado a partir de coordenadas não inteiras (pela continuidade) de  $f(x, y)$ . Se a operação geométrica é considerada como um mapeamento de  $f$  para  $g$ , pixels de  $f$  podem ser mapeados entre pixels de  $g$  e vice-versa. Assim novamente deve ser realizada alguma operação (ex. escala) para corrigir a distorção de pontos não inteiros.

# Referências Bibliográficas

- [Castleman, 1979] Castleman, K. R. (1979). *Digital Image Processing*. Prentice-Hall, Inc., Englewood Cliffs, N.J.
- [Gonzalez and Wintz, 1987] Gonzalez, R. C. and Wintz, P. (1987). *Digital Imagem Processing*. Addison-Wesley Publishing Company, second edition edition.
- [Hearn and Baker, 1994] Hearn, D. and Baker, P. (1994). *Computer Graphics*. Prentice-Hall.
- [Mascarenhas and Velasco, 1989] Mascarenhas, N. D. A. and Velasco, F. R. D. (1989). Processamento digital de imagens. In *IV Escola Brasileiro-Argentina de Informática*, Universidade Católica de Santiago del Estero, Termas do Rio Hondo - Argentina.
- [Minghim and Oliveira, 1997] Minghim, R. and Oliveira, M. C. F. (1997). Uma introdução à visualização computacional xvi jai'97. In *XVII Congresso da SBC*, volume 1777 of *Jornadas de Atualização em Informática*, pages 85–131, Brasília - Brazil.
- [Persiano and Oliveira, 1989] Persiano, R. C. M. and Oliveira, A. A. F. (1989). *Introdução à Computação Gráfica*. Livros Técnicos e Científicos Editora Ltda.
- [Plastock and Kalley, 1999] Plastock, R. and Kalley, G. (1999). *Computação Gráfica*. Mc Graw Hill.
- [Schröeder, 1998] Schröeder, W. (1998). *The Visualization Toolkit*.
- [Schröeder et al., 1996] Schröeder, W. J., Martin, K., and Lorensen, W. (1996). *The Visualization Toolkit - An Object-Oriented Approach to 3D Graphics*. Prentice-Hall.

# **Apêndice A**

## **Histórico**

**01.11.2003**

### **Novidades**

- Apostila de processamento de imagens adicionada.
- Links externos corrigidos.

### **Problemas**

- Os capítulos de Rendering e Modelagem ainda estão faltando;
- Ainda há referências com problemas.

**27.08.2003**

### **Novidades**

- Versão totalmente revisada.

### **Problemas**

- Os capítulos de Rendering, Modelagem e Processamento de Imagens estão faltando;
- Algumas referências precisam ser corrigidas.