



Universidade Estadual do Oeste do Paraná - UNIOESTE
Centro de Ciências Exatas e Tecnológicas - CCET
Curso de Ciência da Computação

COMPUTAÇÃO GRÁFICA

**CASCABEL - PR
2021**

SUMÁRIO

UNIDADE 1 – INTRODUÇÃO	1
1.1 INTRODUÇÃO À COMPUTAÇÃO GRÁFICA	1
1.2 SISTEMAS GRÁFICOS	2
1.3 APLICAÇÕES DA COMPUTAÇÃO GRÁFICA	2
1.4 HARDWARE GRÁFICO	4
1.5 RESOLUÇÃO GRÁFICA	4
1.6 SISTEMAS DE COORDENADAS	5
UNIDADE 2 – HARDWARE GRÁFICO	7
2.1 INTRODUÇÃO	7
2.2 OBJETOS GRÁFICOS	7
2.2.1 FORMATO DE OBJETOS GRÁFICOS	8
2.2.2 FORMATO VETORIAL	8
2.2.3 FORMATO MATRICIAL	8
2.3 CLASSIFICAÇÃO DOS DISPOSITIVOS	9
2.3.1 CRITÉRIOS DE CLASSIFICAÇÃO	10
2.4 OPERAÇÃO DOS EQUIPAMENTOS GRÁFICOS	10
2.4.1 MODELO DE INTERAÇÃO	11
2.4.2 REALIDADE VIRTUAL	12
2.5 ESTAÇÃO GRÁFICA	12
2.5.1 SISTEMA DE COORDENADAS DO DISPOSITIVO	13
2.6 EQUIPAMENTOS DE ENTRADA GRÁFICA	13
2.6.1 DISPOSITIVOS ABSOLUTOS DE ENTRADA VETORIAL	13
2.6.2 DISPOSITIVOS RELATIVOS DE ENTRADA VETORIAL	15
2.6.3 DISPOSITIVOS DE ENTRADA MATRICIAL	16
2.7 EQUIPAMENTOS DE SAÍDA GRÁFICA	17
2.7.1 MONITORES DE VÍDEO	18
2.7.2 DISPOSITIVOS DE SAÍDA VETORIAL	21
2.7.3 DISPOSITIVOS DE SAÍDA MATRICIAL	22
2.7.4 DISPOSITIVOS TRIDIMENSIONAIS	22
2.8 EQUIPAMENTOS DE PROCESSAMENTO GRÁFICO	23
2.8.1 DISPOSITIVOS DE PROCESSAMENTO VETORIAL	23
2.8.2 DISPOSITIVOS DE PROCESSAMENTO MATRICIAL	24
UNIDADE 3 – MODELAGEM DE SÓLIDOS, CURVAS E SUPERFÍCIES	25
3.1 REPRESENTAÇÃO POLIÉDRICA DOS OBJETOS	25
3.1.1 CURVAS E REGIÕES POLIGONAIS	25
3.2 ARMAZENAMENTO DE POLÍGONOS	26
3.2.1 VÉRTICES EXPLÍCITOS	26
3.2.2 POLÍGONOS DEFINIDOS POR PONTEIROS PARA LISTA DE VÉRTICES	26
3.2.3 ARESTAS EXPLÍCITAS	26
3.3 REPRESENTAÇÃO DE SÓLIDOS POR APROXIMAÇÕES	27
3.3.1 MODELAGEM POR GEOMETRIA SÓLIDO CONSTRUTIVA (CSG)	27
3.3.2 MODELAGEM POR REPRESENTAÇÃO DE FRONTEIRAS (B-REP)	28
3.3.3 MODELAGEM BASEADA EM CARACTERÍSTICAS (SWEEP)	28
3.3.4 ENUMERAÇÃO DA OCUPAÇÃO ESPACIAL	29
3.3.5 OCTREES	29
3.4 COMPARAÇÃO DAS REPRESENTAÇÕES	31
3.5 MODELAGEM DE CURVAS	31
3.5.1 REPRESENTAÇÃO POR HERMITE	33
3.5.2 REPRESENTAÇÕES POR SPLINE	37

3.5.3 CURVAS BÉZIER E SUPERFÍCIES BÉZIER	39
UNIDADE 4 – TRANSFORMAÇÕES GEOMÉTRICAS	42
4.1 TRANSFORMAÇÕES 2D	42
4.1.1 TRANSLAÇÃO	42
4.1.2 ESCALA	43
4.1.3 ROTAÇÃO	43
4.2 COORDENADAS HOMOGÊNEAS E MATRIZES DE TRANSFORMAÇÃO	45
4.3 CONCATENAÇÃO DE MATRIZES	46
4.4 TRANSFORMAÇÕES 2D ADICIONAIS	47
4.4.1 ESPELHAMENTO (MIRROR)	47
4.4.2 CISALHAMENTO	49
4.5 TRANSFORMAÇÕES ENTRE SISTEMAS DE COORDENADAS	49
4.7 TRANSFORMAÇÕES 3D	50
4.8 COMPOSIÇÃO DE TRANSFORMAÇÕES EM 3D	52
UNIDADE 5 – VISUALIZAÇÃO EM 3D	56
5.1 PIPELINE DE VISUALIZAÇÃO	56
5.2 COORDENADAS DE CÂMERA	58
5.2.1 ESPECIFICAÇÃO DO SISTEMA DE REFERÊNCIA DA CÂMERA	58
5.2.2 TRANSFORMAÇÃO DO SISTEMA DE REFERÊNCIA DO UNIVERSO PARA O SISTEMA DE REFERÊNCIA DA CÂMERA	59
5.3 PROJEÇÕES	61
5.3.1 PROJEÇÃO PERSPECTIVA	62
5.3.2 ANOMALIAS DA PERSPECTIVA	64
5.4 DESENVOLVIMENTO MATEMÁTICO PARA PROJEÇÕES PERSPECTIVAS	65
5.5 DESENVOLVIMENTO MATEMÁTICO PARA PROJEÇÕES PARALELAS	67
5.6 PROJEÇÃO PERSPECTIVA UTILIZANDO COORDENADAS ESFÉRICAS	67
5.6.1 PONTO DE VISTA E PERSPECTIVA	73
5.7 TRANSFORMAÇÃO JANELA–PORTA DE VISÃO ("Window-to-Viewport")	73
5.7.1 EIXO VERTICAL DA VIEWPORT EM SENTIDO OPPOSTO A WINDOW	75
5.8 CONCATENAÇÃO DAS MATRIZES DO PIPELINE	76
5.9 OCULTAÇÃO DE LINHAS E SUPERFÍCIES	77
5.9.1 OCULTAÇÃO DE LINHAS	77
5.9.2 ALGORITMOS BÁSICOS PARA DETERMINAÇÃO DE SUPERFÍCIES VISÍVEIS	82
5.9.3 ALGORITMOS PARA DETERMINAÇÃO DE SUPERFÍCIES VISÍVEIS	83
UNIDADE 6 – ALGORITMOS RASTER BÁSICOS PARA DESENHO DE PRIMITIVAS EM 2D	90
6.1 TRAÇADO DE CURVAS EM DISPOSITIVOS GRÁFICOS MATRICIAIS	90
6.2 SIMETRIA E REFLEXÃO	90
6.3 CARACTERÍSTICAS DESEJÁVEIS PARA OS ALGORITMOS DE CONVERSÃO MATRICIAL DE SEGMENTOS DE RETAS	91
6.3.1 CRITÉRIO ADOTADO	91
6.4 ALGORITMOS BÁSICOS PARA O TRAÇADO DE RETAS	92
6.4.1 ALGORITMO DO PONTO MÉDIO	93
6.5 CONVERSÃO MATRICIAL DE CIRCUNFERÊNCIAS	94
6.5.1 SIMETRIA DE ORDEM 8	95
6.5.2 ALGORITMOS DO PONTO-MÉDIO PARA CIRCUNFERÊNCIAS	95
6.6 CONVERSÃO MATRICIAL DE ELIPSES	96
6.7 CORREÇÃO NO TRAÇADO	97
6.8 ANTIALIASING	99
6.8.1 AMOSTRAGEM DE ÁREAS NÃO PONDERADA	99
6.8.2 AMOSTRAGEM DE ÁREAS PONDERADAS	101

6.9 PREENCHIMENTO DE POLÍGONOS	101
6.9.1 RETÂNGULOS	102
6.9.2 POLÍGONOS DE FORMA ARBITRÁRIA	103
6.9.3 ARESTAS HORIZONTAIS	104
6.9.4 SLIVERS	105
6.10 PREENCHIMENTO COM PADRÕES (PATTERNS)	106
6.10.1 MATRIZ DE PONTOS	106
6.10.2 DETERMINAÇÃO DA COR PARA UM PIXEL QUALQUER NA TELA	107
6.11 PRIMITIVAS ESPESSAS	107
6.11.1 REPRODUÇÃO DE PIXELS	107
6.11.2 MOVIMENTO DA CANETA	109
6.11.3 PREENCHIMENTO DE ÁREA ENTRE DOIS LIMITES	110
6.11.4 TIPOS DE LINHAS	111
6.12 CLIPPING	112
6.12.1 POINT CLIPPING - RECorte POR PONTOS	112
6.12.2 LINE CLIPPING – RECorte POR LINHAS OU RECorte POR EQUAÇÕES SIMULTÂNEAS	112
6.12.3 ALGORITMO DE COHEN-SUTHERLAND	114
6.12.4 RECorte DE POLÍGONOS	117
6.12.5 RECorte DE POLÍGONOS – SUTHERLAND-HODGEMAN	118
6.12.6 RECorte DE POLÍGONOS – WEILER-ATHERTON	120
6.13 TESTES DENTRO-FORA	121
UNIDADE 7 – CORES E SISTEMAS DE CORES	123
7.1 PERCEPÇÃO DE COR	123
7.2 SISTEMAS DE CORES PRIMÁRIAS	124
7.3 MODELO XYZ	126
7.4 MODELO RGB (RED, GREEN, BLUE)	128
7.5 MODELO HSV (HUE, SATURATION, VALUE)	129
7.6 MODELO HLS (HUE, LIGHTNESS, SATURATION)	130
7.7 USO DE CORES NA COMPUTAÇÃO GRÁFICA	131
UNIDADE 8 – ILUMINAÇÃO E SOMBREAMENTO	133
8.1 ILUMINAÇÃO	133
8.1.1 LUZ AMBIENTE	133
8.1.2 REFLEXÃO DIFUSA	133
8.1.3 ATENUAÇÃO DAS FONTES DE LUZ	135
8.1.4 REFLEXÃO ESPECULAR	135
8.1.5 CONSIDERAÇÕES SOBRE CORES	137
8.1.6 TRANSPARÊNCIA	138
8.1.7 SOMBRAS	141
8.2 SOMBREAMENTO PARA POLÍGONOS	142
8.2.1 SOMBREAMENTO CONSTANTE	142
8.2.2 SOMBREAMENTO GOURAUD	142
8.2.3 SOMBREAMENTO PHONG	144
8.2.4 SOMBREAMENTO “FAST PHONG”	146
8.3 MÉTODOS DE RAY TRACING	147
8.3.1 ALGORITMO DE RAY TRACING BÁSICO	148
8.3.2 CALCULANDO A INTERSEÇÃO DO RAIO COM AS SUPERFÍCIES	150

UNIDADE 1 – INTRODUÇÃO

1.1 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 subdividir a Computação Gráfica em três subáreas:

Síntese de Imagens: subárea que se preocupa com a produção de representações visuais a partir das especificações geométrica e visual de seus componentes. É frequentemente confundida com a própria Computação Gráfica. As imagens produzidas por esta subárea são colocadas nos chamados *Display-File*.

Processamento de Imagens: envolve as técnicas de transformação de Imagens, em 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 subárea são colocadas/recuperadas nos chamados *Raster-File*.

Análise de Imagens: subárea que procura obter a especificação dos componentes de uma imagem a partir de sua representação visual. Ou seja, através da informação pictórica da imagem (a própria imagem!) produz uma informação não pictórica da imagem (por exemplo, as primitivas geométricas elementares que a compõem).

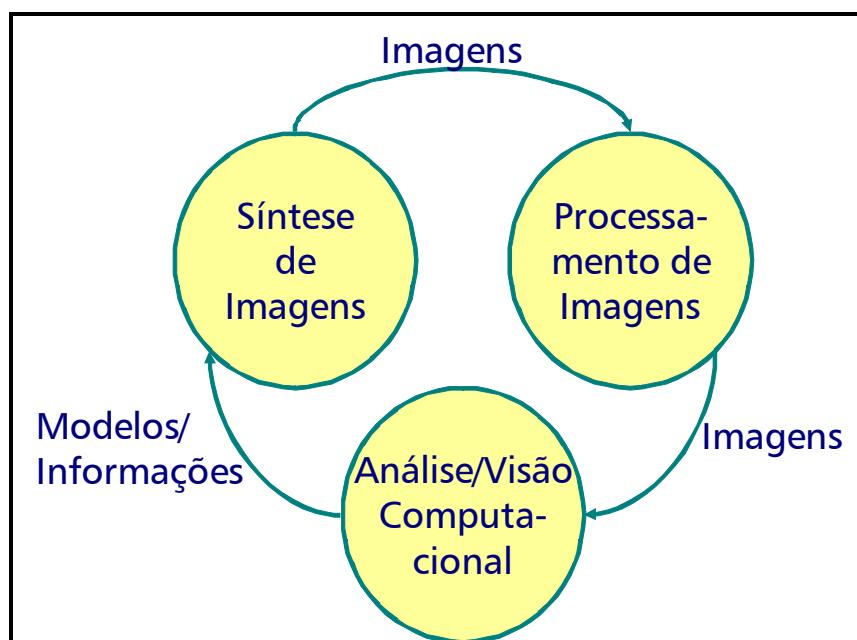


Figura 1.1 - Relacionamento entre as três subáreas da Computação Gráfica

Na última década adicionou-se a esse contexto a área de *Visualização de Dados*, também chamada **Visualização Computacional**, 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.

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.

A computação gráfica cuida 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 câmara de vídeo, ou imagens de satélite - para serem manipuladas visando diferentes objetivos.

1.2 SISTEMAS GRÁFICOS

A Computação Gráfica (especialmente as partes relativas a gráficos 3D e a gráficos 3D interativos) desenvolveu-se de modo bem diverso: de simples programas gráficos para computadores pessoais a 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 padronizar a construção de aplicativos que se utilizam de recursos gráficos e torná-los os mais independentes possíveis de máquinas, portanto facilmente portáveis, foram desenvolvidos os chamados Sistemas Gráficos.

Exemplos de Sistemas Gráficos:

- GKS - *Graphical Kernel System*;
- PHIGS - *Programmer's Hierarchical Interactive Graphics System*;
- PEX – Extensão do PHIGS para o XWindow;
- OpenGL (GL - *Graphics Library*);
- DirectX.

1.3 APLICAÇÕES DA COMPUTAÇÃO GRÁFICA

- **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 ópticos, 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:** É a utilização de técnicas gráficas para demonstração de resultados, ideias e gráficos, com o intuito de mostrar ou transmitir conhecimento específico como, por exemplo, em uma aula, ou reunião, ou na construção de material didático.

1.4 HARDWARE GRÁFICO

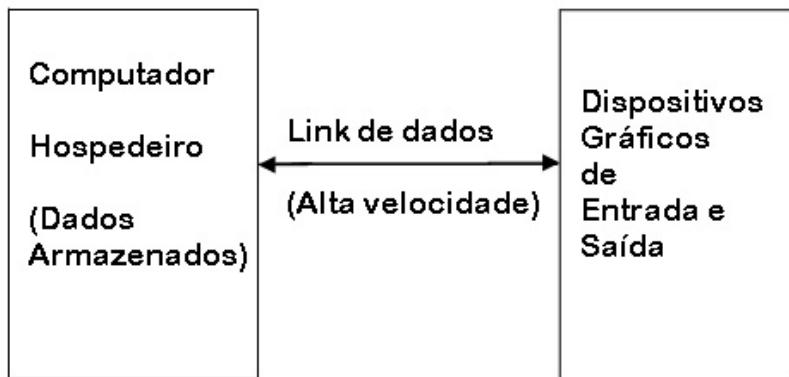


Figura 1.2 – Sistema de Hardware para Computação Gráfica

Um sistema de hardware para computação gráfica consiste essencialmente de dispositivos gráficos de entrada e saída (I/O) ligados a um computador (Fig. 1.2).

Ao conjunto de dispositivos de I/O gráficos alocado para utilização por uma única pessoa por vez denomina-se genericamente de "estação de trabalho gráfica", ou *graphics workstation*.

Um sistema gráfico multiusuário 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 na CG é frequente 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 será necessário e, nesse caso, deverá possuir taxas de transmissão compatíveis com a aplicação desejada. Velocidades inferiores a 1 Mbps podem ser insuficientes para animação gráfica de alta resolução, onde cada frame de imagem pode ter um megabyte de dados, mesmo utilizando técnicas de compressão de imagens. Um sistema de conexão ideal nesse caso é uma *Local Area Network* (LAN) como a *Ethernet*, ou conexões ADSL de alta velocidade.

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).

1.5 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 quatro parâmetros que definem a resolução:

- ndh - o número de posições endereçáveis horizontalmente.

- ndv - o número de posições endereçáveis verticalmente.
- $width$ - a largura do retângulo de visualização em mm.
- $height$ - a altura do retângulo de visualização em mm.

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

- **resolução horizontal:** $horiz_res := ndh/width$
- **tamanho ponto horizontal:** $horiz_dot_size := width/ndh$
- **resolução vertical:** $vert_res := ndv/height$
- **tamanho ponto vertical:** $vert_dot_size := height/ndv$
- **total pontos endereçáveis:** $total_nr_dots := ndh*ndv$
- **resolução de área:** $area_res := total_nr_dots/(width*height)$
- **razão de aspecto gráfica:** $aspect_ratio := vert_dot_size/horiz_dot_size$
- **razão de aspecto física:** $physical_aspect_ratio := height/width$

1.6 SISTEMAS DE COORDENADAS

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

$$0 \leq dcx \leq ndhm1 \equiv ndh - 1$$

$$0 \leq dcy \leq ndvm1 \equiv ndv - 1$$

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:

$$0 \leq ndcx \leq 1$$

$$0 \leq ndcy \leq 1$$

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:

$$dcx := \text{round}(ndcx * ndhm1)$$

$$dcy := \text{round}(ndcy * ndvml)$$

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:

$$dcx := \text{trunc}(ndhm1 * pcx / width)$$

$$dcy := \text{trunc}(ndvml * pcy / height)$$

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:

$$xmin \leq x \leq xmax$$

$$ymin \leq y \leq ymax$$

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:

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

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

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 (Fig. 1.3).

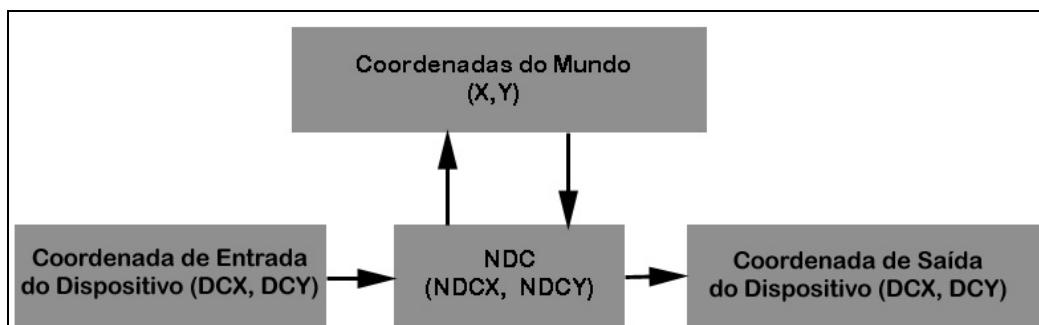


Figura 1.3: Sistemas de coordenadas e suas transformações.

UNIDADE 2 – HARDWARE GRÁFICO

2.1 INTRODUÇÃO

Os equipamentos desempenham um papel importante na computação gráfica. Não só pela relevância da imagem como um dos produtos da atividade, mas também pela influência de características dos equipamentos nos processos da área. Em alguns casos, como no desenvolvimento de interfaces com o usuário, torna-se impossível dissociar as duas coisas. Os diversos parâmetros de funcionalidade e desempenho dos dispositivos gráficos são determinados em grande parte pelas áreas de aplicação que os utilizam.

Neste capítulo vamos estudar os equipamentos utilizados em computação gráfica fazendo uma abordagem dos dispositivos gráficos do ponto de vista da classificação funcional, e em cada caso daremos alguns exemplos de equipamentos que se utilizam de diversos formatos gráficos.

2.2 OBJETOS GRÁFICOS

Um objeto gráfico é definido através da especificação de seus diversos atributos. Dentre esses atributos podemos destacar os **atributos geométricos**, que estabelecem as propriedades métricas dos objetos; os **atributos topológicos**, que tratam da forma dos objetos, e os **atributos de cor**, que definem a informação de textura dos objetos.

Os objetos manipulados em computação gráfica podem ser especificados em dois níveis distintos de abstração: nível contínuo e nível discreto. Ilustramos esses níveis na Figura 2.1, para objetos de dimensão 1, 2 e 3.

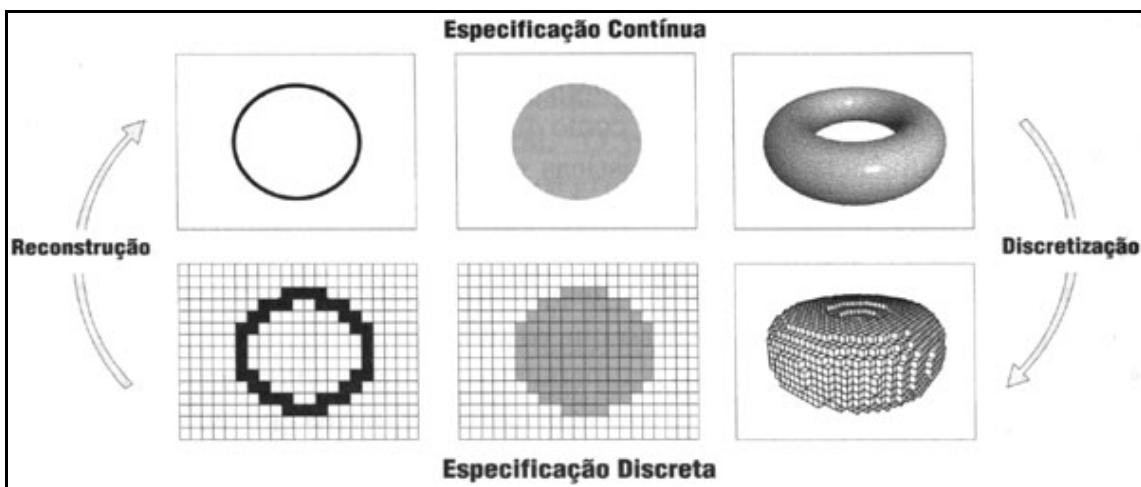


Figura 2.1 – Níveis de abstração de objetos gráficos

No nível contínuo, os diversos atributos do objeto gráfico são descritos através de modelos definidos em espaços topológicos utilizando funções contínuas ou de classe de diferenciabilidade mais alta.

No domínio discreto, os diversos atributos do objeto gráfico são discretizados com o objetivo de se obter uma representação finita, que possa ser utilizada no computador.

2.2.1 FORMATO DE OBJETOS GRÁFICOS

Correspondendo aos níveis de abstração de objetos contínuos e discretos, existem dois formatos básicos para representar os objetos no computador: o formato vetorial e o formato matricial. O **formato vetorial** descreve os atributos que definem o objeto no domínio contínuo; o **formato matricial** é utilizado como uma maneira simples e eficiente de descrever os atributos dos objetos no domínio discreto.

Os diferentes formatos de objetos gráficos são utilizados na especificação e representação dos objetos no computador. Existe, portanto, uma estreita relação entre os formatos vetorial e matricial descritos acima, e os diversos equipamentos utilizados em computação gráfica, e áreas afins.

2.2.2 FORMATO VETORIAL

No formato vetorial, os diversos atributos dos objetos são definidos no domínio contínuo. Para isso utilizamos um espaço topológico, e funções contínuas definidas nesse espaço. Introduzimos coordenadas no espaço e essas coordenadas são utilizadas para indicar posições ou ações no espaço. No primeiro caso, elas podem ser usadas na especificação dos pontos iniciais e finais de segmentos de reta, vértices de polígonos e malhas de controle de curvas ou superfícies paramétricas. Já no segundo caso, elas podem especificar campos de vetores associados a forças, velocidades, etc.

As informações geométricas precisam ser complementadas por informações topológicas para especificar completamente o modelo de um objeto gráfico. A discussão desse tópico, no âmbito da computação gráfica, é feita na área de Modelagem.

2.2.3 FORMATO MATRICIAL

O formato matricial é obtido fazendo uma discretização uniforme do espaço no qual está definido o objeto gráfico. Para isso, definimos um reticulado nesse espaço, e cada célula do reticulado determina uma amostra do objeto gráfico. Um reticulado bidimensional, também chamado de matriz de discretização, é ilustrado na Figura 2.2.

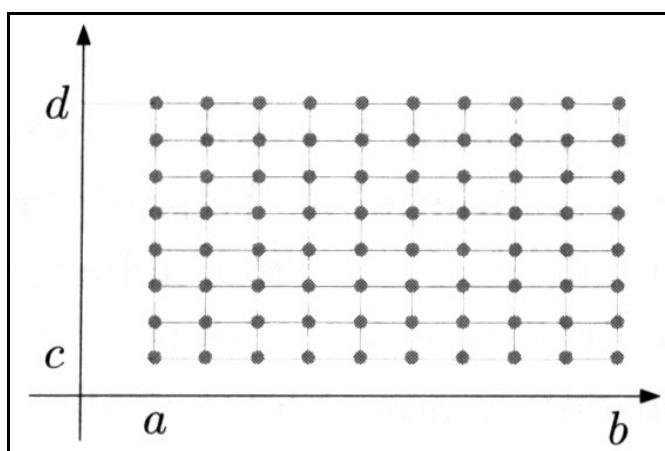


Figura 2.2 – Reticulado uniforme da representação matricial bidimensional no plano

O número de linhas da matriz de discretização é chamado de *resolução vertical* da representação matricial, e o número de colunas n é chamado de *resolução horizontal*. Denominamos *resolução espacial*, ou *resolução geométrica*, do formato matricial a ordem $m \times n$ da matriz de discretização. Quanto maior a resolução, mais detalhes da representação contínua do objeto gráfico podem ser captados no seu formato matricial. No caso de objetos gráficos do espaço euclidiano tridimensional, a sua representação matricial define uma *imagem volumétrica* tomando matrizes de ordem $m \times n \times p$ onde cada entrada contém atributos dos objetos gráficos do espaço.

Na Figura 2.3, à esquerda, mostramos uma elipse representada no formato vetorial, juntamente com um reticulado de discretização de resolução 21×15 . Na figura à direita, mostramos a mesma elipse representada no formato matricial.

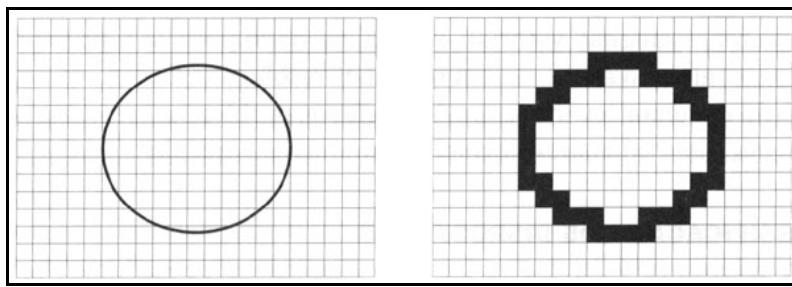


Figura 2.3 – Representação vetorial e matricial de uma elipse

2.3 CLASSIFICAÇÃO DOS DISPOSITIVOS

Os dispositivos gráficos são projetados usualmente de forma a privilegiar um dos dois formatos de especificação de objetos descritos acima. Isso não significa que um tipo de equipamento somente possa operar com um determinado formato de um objeto gráfico. Os dispositivos do tipo matricial podem, por exemplo, reproduzir gráficos vetoriais utilizando técnicas de rasterização. Alguns equipamentos deste tipo dispõem até mesmo de suporte em hardware para tal operação. Embora não seja tão comum, os dispositivos do tipo vetorial podem também reproduzir imagens utilizando padrões de linhas. Por exemplo, em mapas e desenhos técnicos, várias hachuras são empregadas para diferenciar áreas, simulando tonalidades de cinza.

Diversos fatores influenciam o desenvolvimento dos equipamentos gráficos. Dentre eles, podemos citar os fatores de natureza técnica, industrial e econômica, bem como a influência do próprio desenvolvimento das técnicas de computação gráfica.

Os fatores de natureza técnica, industrial e econômica fizeram com que os equipamentos vetoriais, que predominavam nos primórdios do desenvolvimento da área, dessem lugar aos equipamentos do tipo matricial. Com efeito, os dispositivos do tipo matricial necessitam do uso de muita memória para armazenar os objetos gráficos em sua forma matricial. Por outro lado, os dispositivos do tipo vetorial se beneficiaram da tecnologia utilizada de forma ampla nos osciloscópios, numa época em que o custo da memória inviabilizava o uso de dispositivos de formato matricial (1960-70). Já os dispositivos do tipo matricial foram impulsionados na década de 80 por dois fatores: a queda do preço de memória, e a revolução provocada pelo rápido crescimento da indústria de televisão. Os avanços recentes nas áreas de supercomputação e da computação paralela, tem tido um impacto significativo nos dispositivos de processamento gráfico.

A influência da evolução das técnicas da computação gráfica na evolução dos equipamentos também é relevante. Inicialmente, quando havia grande concentração de pesquisas relacionadas com a criação e representação de objetos gráficos tridimensionais no computador, os dispositivos vetoriais eram os mais populares. Isso porque esses dispositivos permitiam a visualização do objeto e sua manipulação em tempo real. No entanto os dispositivos vetoriais não reproduzem os diversos atributos de textura dos objetos em cena. Desse modo, quando a ênfase nas pesquisas da área passou da fase de criação de modelos para o processo de síntese de imagens realistas, os equipamentos matriciais passaram a ter maior importância.

Atualmente, a tendência é a busca de soluções integradas usando dispositivos que possam combinar representação vetorial e matricial de objetos gráficos. De um modo geral, os dispositivos vetoriais estão vinculados à especificação e manipulação dos modelos geométricos; os dispositivos matriciais estão relacionados com a exibição, o processamento, e a análise de imagens. Do ponto de vista tecnológico, ao invés de se utilizar dispositivos genuinamente vetoriais, utilizam-se dispositivos matriciais que fazem a representação vetorial de objetos gráficos através do processo de rasterização.

2.3.1 CRITÉRIOS DE CLASSIFICAÇÃO

No estudo dos dispositivos gráficos é necessário criar abstrações das suas características operacionais de modo que o vínculo entre programas e equipamentos não se transforme num fator de dependência. Vários aspectos contribuem para o estabelecimento de critérios para uma classificação dos equipamentos gráficos. Nessa análise definiremos categorias de equipamentos estruturadas hierarquicamente segundo dois pontos de vista: o funcional, e o do formato dos objetos gráficos.

Em relação ao critério funcional, podemos dividir os dispositivos gráficos em três tipos: **equipamentos de entrada**, **equipamentos de processamento** e **equipamentos de saída**.

- **equipamentos de entrada:** são os equipamentos cuja finalidade é capturar informações gráficas para o computador;
- **equipamentos de processamento:** são computadores com uma arquitetura especial orientada para a manipulação de objetos gráficos;
- **equipamentos de saída:** são dispositivos que permitem a exibição de objetos gráficos.

Quanto ao formato de dados os dispositivos gráficos se dividem em equipamentos do **tipo vetorial** e do **tipo matricial**. Os dispositivos do tipo vetorial podem por sua vez ser classificados de acordo com a dimensão do espaço vetorial a eles associados em dispositivos bidimensionais, tridimensionais, etc. O diagrama da Figura 2.4 ilustra o uso dos dois formatos de objetos gráficos em um sistema.

2.4 OPERAÇÃO DOS EQUIPAMENTOS GRÁFICOS

Do ponto de vista do usuário, podemos classificar os dispositivos gráficos quanto aos diferentes modos de operação. Nesse caso eles se dividem nas classes

de dispositivos **interativos** e **não-interativos**. A diferença central entre estes dois modos de operação está no papel desempenhado pelo usuário na utilização do equipamento.

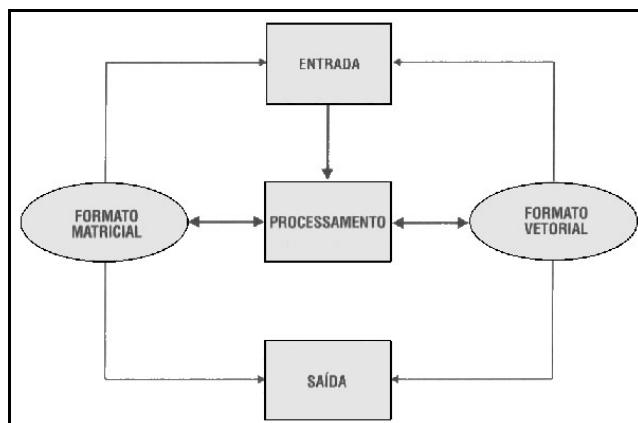


Figura 2.4 – Formatos de dados no processamento gráfico

No modo interativo o usuário participa ativamente do processo fechando o ciclo

entrada → processamento → saída

que se dá necessariamente em tempo real; isto é, o tempo de resposta às ações do usuário não é perceptível. Os dispositivos gráficos interativos são, portanto, utilizados de forma integrada em **estações de trabalho**.

No modo não-interativo o usuário atua passivamente, controlando os dispositivos gráficos para aquisição de dados ou produção de imagens, que geralmente não é feita em tempo real. Os dispositivos gráficos não-interativos são, em geral, utilizados de forma independente para a execução de tarefas específicas, não sendo crítico o seu relacionamento com outros equipamentos do sistema computacional.

2.4.1 MODELO DE INTERAÇÃO

O processo interativo, como dissemos anteriormente, combina a entrada, o processamento, e a saída de objetos gráficos em tempo real. Um modelo para o desenvolvimento de tarefas de interação estabelece classes lógicas de funções gráficas que podem ser implementadas em diversos dispositivos físicos, utilizando diversas técnicas de interação. Essas funções lógicas são: **keyboard**, **locator**, **valuator**, **button** e **pick**.

- o **keyboard** tem como finalidade a entrada de dados textuais na forma de cadeias de caracteres. Sua implementação pode ser feita de forma a possibilitar a edição de linhas e até mesmo de blocos de texto;
- o **locator** tem como finalidade a entrada de dados vetoriais;
- o **valuator** tem como finalidade a entrada de dados escalares, isto é, números reais;
- os **buttons** tem como finalidade a seleção de um ou mais elementos de um conjunto discreto de opções;
- a função **pick** tem como objetivo identificar um objeto numa coleção de objetos apresentada graficamente.

Essas funções implementam o conceito de **dispositivo lógico**, de modo a criar uma definição independente dos dispositivos físicos. Dessa maneira, introduz-se um nível de abstração que isola as funções de interação com o usuário das particularidades de operação dos equipamentos.

2.4.2 REALIDADE VIRTUAL

Chamamos de **realidade virtual** a um ambiente que permite a total interação do usuário com a simulação de um sistema sintético em um espaço tridimensional. Os dispositivos gráficos desse sistema fazem com que o observador se sinta parte integrante do ambiente sintético, podendo interagir com os objetos presentes, regidos pelas leis desse ambiente.

Para se trabalhar com realidade virtual se faz necessário o uso de equipamentos com um processamento rápido aliado a dispositivos gráficos de saída que simulem três dimensões, juntamente com dispositivos de interação que permitam variados graus de liberdade e até a mesmo possam transmitir ao computador informações físicas tais como força, momento, etc.

2.5 ESTAÇÃO GRÁFICA

O nome **estação gráfica** ou **estação de trabalho** tem sido utilizado para representar uma família de equipamentos gráficos que possuem uma característica arquitetônica comum, que consiste em se adicionar a uma arquitetura tradicional de computadores elementos específicos para o processamento gráfico. Desse modo uma estação gráfica consiste de dispositivos de entrada e saída, memória e CPU, conforme mostramos no diagrama da Figura 2.5.

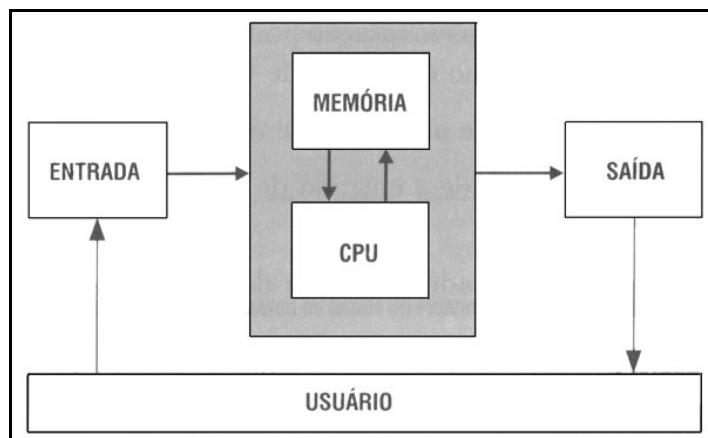


Figura 2.5 – Estação gráfica

Normalmente, as estações gráficas possuem um dispositivo para processamento gráfico que é chamado de **processador gráfico**. Esse dispositivo em geral possui processamento próprio com memória, e sua arquitetura é projetada para realizar várias operações gráficas com um alto grau de eficiência. Essas operações podem ser do tipo vetorial (traçado de segmentos de reta, círculos, etc.) ou matricial (com imagens). É muito comum também se implementar vários tipos de transformações geométricas no processador gráfico.

Um tipo de operação com imagem muito comum nas estações gráficas matriciais é chamada de **operação de “bit block transfer”** comumente chamadas de **operação bitblt**. Essas operações permitem a manipulação de blocos de imagem cuja posição de fronteira na memória está definida a nível de bits (e não de bytes). As operações **bitblt** são essenciais para a implementação eficiente de sistemas de janelas do tipo MS-Windows, ou X-Windows no sistema UNIX.

Para finalizar, deve-se observar que na evolução dos computadores pessoais vem se substituindo as placas gráficas, que são dispositivos de saída matriciais, por processadores gráficos. Esse fato vem fazendo com que a linha divisória entre microcomputadores e estações gráficas diminua a cada dia. Por outro lado, as estações gráficas têm evoluído lançando mão das últimas inovações tecnológicas e se transformando no que se chama de super-estações gráficas.

2.5.1 SISTEMA DE COORDENADAS DO DISPOSITIVO

Cada dispositivo gráfico possui um sistema de coordenadas associado, em relação ao qual os dados são referenciados. Esse sistema de coordenadas é chamado de **sistema de coordenadas do dispositivo**. A especificação dos dados nesse sistema de coordenadas pode ser feita de forma relativa ou absoluta. No caso relativo às coordenadas de posição são referenciadas a partir do ponto corrente, e o dispositivo é chamado de **dispositivo de coordenadas relativas**. No caso absoluto a posição é dada em relação a uma origem, fixa, do sistema e o dispositivo é chamado de **dispositivo de coordenadas absolutas**.

2.6 EQUIPAMENTOS DE ENTRADA GRÁFICA

Do ponto de vista do formato da imagem os dispositivos de entrada gráfica podem ser classificados como vetoriais e matriciais.

Os dispositivos de entrada vetorial são em sua maioria utilizados como componentes de estações gráficas interativas. Um exemplo típico é o “mouse”, componente indispensável em uma estação de trabalho interativa que utilize ambiente de janelas.

Os dispositivos de entrada matricial são tradicionalmente utilizados de modo não interativo devido, principalmente, ao grande volume de dados que devem ser manipulados. Esta situação tende a se modificar com a evolução dos equipamentos de aquisição, exibição e processamento de imagens, que poderão tornar possíveis aplicações em tempo real envolvendo dados matriciais.

2.6.1 DISPOSITIVOS ABSOLUTOS DE ENTRADA VETORIAL

Exemplos de dispositivos cujo sistema de coordenadas é absoluto são: o “*light pen*”, a “*tablet*”, o “*touch panel*”, o “*3D digitizer*” e a “*data glove*”. As características técnicas relevantes dos dispositivos de entrada vetorial são a sua resolução, linearidade, repetibilidade e área de ação.

O ***light pen*** é um dispositivo bidimensional que funciona necessariamente acoplado a um terminal de vídeo. Este equipamento é composto por uma caneta com uma fotocélula na ponta ligada ao circuito de vídeo do terminal. Dessa maneira é possível detectar pontos apresentados na tela e consequentemente sua

localização. Este dispositivo surgiu com os primeiros equipamentos gráficos interativos. Atualmente ele caiu em desuso devido a alguns problemas técnicos apresentados.

O **touch panel** também é um dispositivo bidimensional de entrada que deve ser integrado a um terminal de vídeo. Ele consiste de uma tela transparente, sensível ao toque, que é sobreposta à tela do terminal. Este dispositivo apresenta severas limitações em termos de resolução. Por este motivo, ele é indicado apenas para a seleção de objetos gráficos apresentados na tela. Um exemplo desse tipo de utilização pode ser visto em alguns terminais eletrônicos de banco, e em terminais de informação ao público.

A **tablet** ou **mesa digitalizadora** consiste de uma base plana e um instrumento indicador em forma de caneta ou bloco. No indicador existem um ou mais botões. O equipamento fornece a posição do indicador em relação ao sistema de referência da base, juntamente com o estado dos botões ("on" ou "off"). Com relação aos dados de entrada a tablet é um dispositivo bidimensional. Em alguns modelos é possível especificar a pressão exercida na ponta da caneta e também a orientação (latitude e longitude) da caneta. Estes dados podem ser interpretados de maneira bastante efetiva em um programa de pintura eletrônica, para permitir a simulação de instrumentos de desenho tradicionais como o pincel, o crayon, etc. A Figura 2.6 exibe uma tablet.

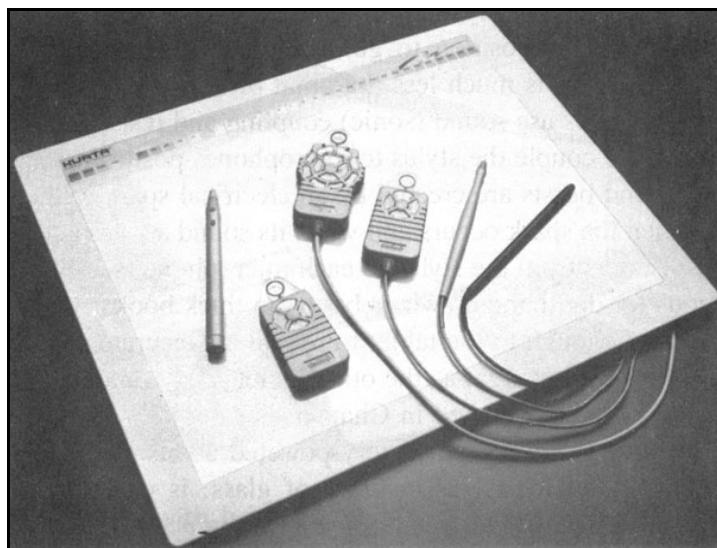


Figura 2.6 – Um modelo de tablet ou mesa digitalizadora

O **3D digitizer** permite captar posição (X , Y , Z) e orientação (α , β , γ) no espaço tridimensional. Este dispositivo é constituído por um emissor magnético e um sensor que, em geral, tem a forma de uma caneta. Esse dispositivo é bastante conveniente para digitalizar diretamente pontos da superfície de um objeto tridimensional.

A **data glove** é uma luva com uma malha de fibras óticas ao longo dos dedos, e um sensor tridimensional semelhante ao 3D digitizer. Além da posição e orientação, ela permite identificar as configurações das articulações dos dedos. O grau de liberdade de entrada de dados desse dispositivo é muito alto, e permite uma interação natural no espaço ambiente tridimensional. Por essa razão, elas têm sido muito utilizadas em aplicações que simulam "Realidades Virtuais". A Figura 2.7 exibe um destes equipamentos.



Figura 2.7 – Equipamento utilizado em aplicações que simulam “Realidades Virtuais”

2.6.2 DISPOSITIVOS RELATIVOS DE ENTRADA VETORIAL

Os dispositivos vetoriais mais comuns que operam com referencial relativo são: o “mouse”, a “trackball”, o “joystick”, os “dials”, a “spaceball”, e os equipamentos do tipo *force-feedback*. Esses dispositivos registram deslocamentos que são transformados em informações de movimento relativo. Isso implica que o programa de controle do dispositivo tem que manter a posição corrente que é atualizada a cada movimento relativo.

O **mouse**, como foi dito anteriormente, é um dos dispositivos de entrada gráfica mais comuns atualmente por estar associado às estações de trabalho que utilizam sistemas de janelas.

A **trackball** é constituída por uma esfera que gira livremente numa base. Os movimentos de rotação em relação a dois eixos ortogonais são transformados em informações de posição de maneira semelhante à do mouse.

O **joystick** é formado por uma haste conectada a uma base. Em geral, o movimento da haste é transformado em um vetor de velocidades que controla a variação dos dados posicionais. Ou seja, na medida em que a haste se afasta do eixo central, a velocidade aumenta proporcionalmente naquela direção. Alguns tipos de joystick possuem um terceiro grau de liberdade, associado à rotação da haste. Esse dispositivo é utilizado com frequência como interface de entrada dos videogames. A Figura 2.8 exibe um joystick com 3 graus de liberdade.

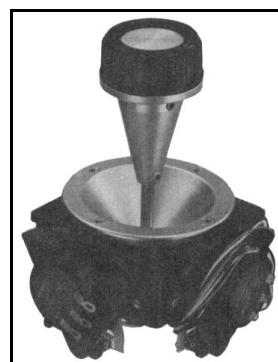


Figura 2.8 – Um modelo de joystick com 3 graus de liberdade (rotação do eixo)

A **spaceball** é uma esfera rígida que tem sensores de esforço. As forças aplicadas na esfera em uma determinada direção são traduzidas em informações de translação e rotação relativas, correspondendo ao movimento infinitesimal da esfera no espaço. Na Figura 2.9 é exibido um modelo de spaceball.



Figura 2.9 – Um modelo de spaceball com 6 graus de liberdade

Os **dials** são potenciômetros tipicamente montados em grupos de seis ou oito. Os potenciômetros fornecem valores escalares e podem ter uma faixa de rotação fixa menor do que 360 graus, ou podem permitir a rotação livre. No primeiro caso o intervalo escalar dos dados é mapeado na faixa de operação do dispositivo, enquanto no segundo caso os dados são especificados de forma relativa. A Figura 2.10 exibe um modelo de dial.

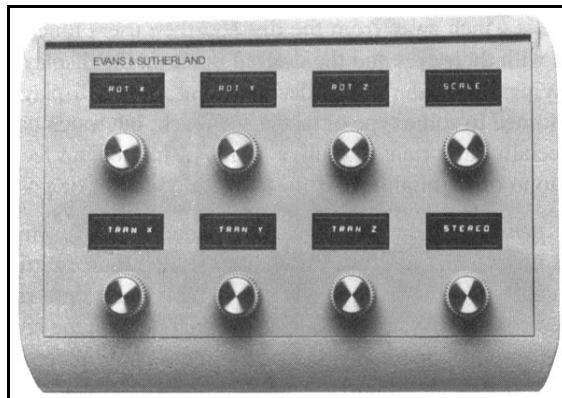


Figura 2.10 – Um modelo de dial composto por oito potenciômetros

Recentemente, tem aumentado o número de equipamentos do tipo “force feedback”. Este tipo de dispositivo além de fornecer dados de posição e orientação, oferece resistência em resposta às ações do usuário.

2.6.3 DISPOSITIVOS DE ENTRADA MATRICIAL

A estrutura dos dispositivos de entrada do tipo matricial consiste de um sensor que capta sinais no espaço ambiente e um circuito analógico-digital que

converte esses sinais analógicos para o formato matricial. (Ver Figura 2.11)

O processo de conversão de uma imagem para uma imagem digital é conhecido como **digitalização**. Os dispositivos de entrada matricial são em sua maioria destinados à digitalização de imagens. Dependendo do meio no qual se encontra a imagem a ser digitalizada temos o “*frame grabber*”, o “*scanner*”, o “*film scanner*” e o “*depth scanner*”.

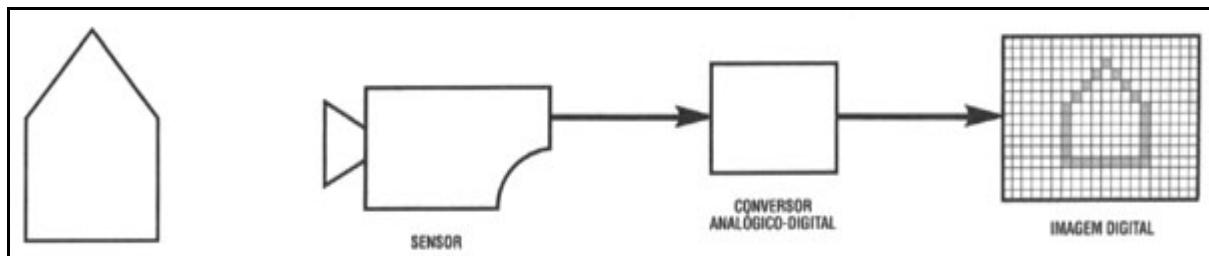


Figura 2.11 – Dispositivo de entrada matricial

O ***frame grabber*** faz a digitalização a partir de um sinal analógico de vídeo. O sinal de vídeo pode ser gerado diretamente por uma câmera ou por um equipamento de reprodução de vídeo. A resolução geométrica da imagem digitalizada é a resolução de vídeo, que é aproximadamente de 512x512 pixels. Os dispositivos mais sofisticados digitalizam com uma resolução de cor de 24 bits.

O ***scanner*** digitaliza a partir de imagens em papel. A imagem é colocada sobre uma superfície transparente, em geral plana ou cilíndrica, que se move numa direção ortogonal a um elemento de digitalização de linha. Esse elemento se compõe de uma fonte de luz e de um sensor que mede a luz refletida linha por linha, em sincronismo com o deslocamento da imagem. A resolução deste dispositivo está situada entre 200 e 4000 pontos por polegada. Um dispositivo semelhante é o ***film scanner***, que obtém uma imagem digital a partir de imagens em transparências, utilizando o laser para maior resolução. Este dispositivo também atinge resoluções superiores a 2000 dpi (“*dots per inch*”).

O ***depth scanner*** ao invés de digitalizar uma imagem, ele captura informações de uma cena tridimensional, produzindo uma matriz de coordenadas com a profundidade de cada ponto da cena. A estrutura dessa matriz depende do processo de varredura utilizado. Os tipos mais comuns possuem varredura plana que fornece informações sobre seções transversais, ou cilíndrica que dá informações sobre seções longitudinais.

2.7 EQUIPAMENTOS DE SAÍDA GRÁFICA

Analogamente aos dispositivos de entrada, podemos subdividir os equipamentos de saída gráfica em vetoriais ou matriciais, de acordo com o tipo de dado gráfico por eles manipulado. Dentre todos os equipamentos gráficos de saída, os ***dispositivos de exibição de vídeo*** são, sem dúvida alguma, os mais importantes e mais comuns. A tecnologia de vídeo implica em uma série de características comuns aos equipamentos vetoriais e matriciais. Por esse motivo, iniciaremos esta seção analisando a estrutura básica dos dispositivos de exibição de vídeo. Em seguida discutiremos os detalhes específicos dos diversos tipos de equipamentos de vídeo no contexto dos dispositivos vetoriais e matriciais.

2.7.1 MONITORES DE VÍDEO

Monitor de vídeo é o nome genérico dado ao dispositivo de saída gráfica que se utiliza de um tubo CRT (*Cathode Ray Tube*), plasma, LCD (*Liquid Crystal Display*), OLED (*Organic Light-Emitting Diode*) ou AMOLED (*Active Matrix Organic Light-Emitting Diode*). Na realidade essa classe de dispositivos de exibição constitui-se em quatro elementos: um monitor, um controlador de vídeo, uma **memória de exibição** e um **conversor digital analógico** (ver Figura 2.12).

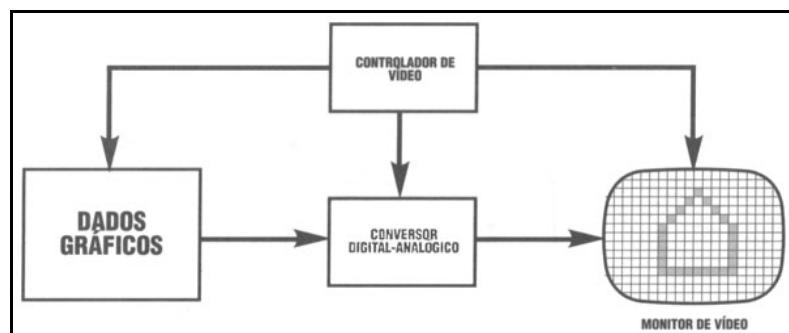


Figura 2.12 – Dispositivo de vídeo

- **MONITORES CRT:**

O monitor CRT (Figura 2.13-a) consiste de um tubo de raios catódicos com uma tela e um canhão. Este produz um ou mais feixes de elétrons, cuja intensidade é variável, e são controlados por um sistema de focalização e exploração. Em cada ponto da tela se coloca uma ou mais camadas de fósforo, de modo que ao atingir um desses pontos o feixe de elétrons dirigido provoca a emissão de radiação eletromagnética na faixa visível do espectro.

Como a resposta luminosa do fósforo utilizado na tela decai exponencialmente com o tempo a imagem precisa ser redesenhada periodicamente. O número de vezes por segundo que a imagem deve ser exibida na tela, para que seja percebida como um fenômeno contínuo no tempo, é chamado de **frequência crítica de fusão**. Este número é determinado por fatores psicofisiológicos. Em média essa frequência se situa próximo a 50 Hz.

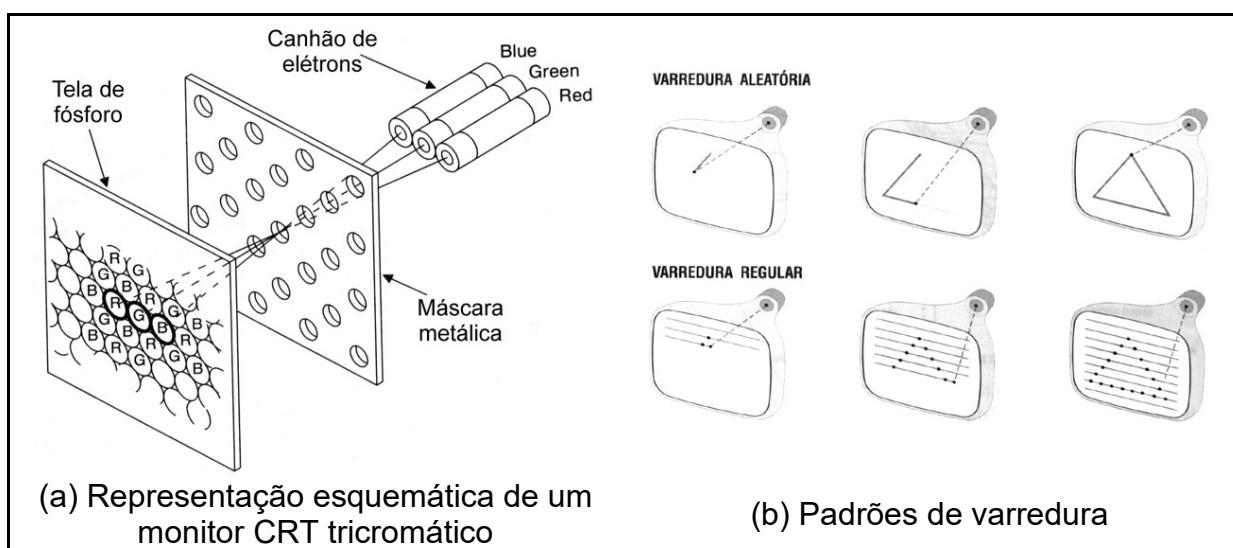


Figura 2.13 – Monitores CRT

Nos monitores CRT o **controlador de vídeo**, também chamado de DPU (*Display Processing Unit*), tem a finalidade de controlar o movimento de exploração na tela do feixe de elétrons, para que a imagem desejada seja produzida. Esse processo, denominado de **varredura** pode ser aleatório ou regular. Na varredura aleatória, o feixe se desloca numa trajetória que segue o desenho das curvas da imagem. Na varredura regular, o feixe se movimenta de acordo com um padrão fixo que percorre toda a tela da esquerda para a direita e de cima para baixo, cobrindo-a por linhas horizontais. Este padrão regular é chamado de **raster**. A Figura 2.13-b ilustra os dois padrões utilizados. Os monitores com tecnologia genuinamente vetorial empregam a varredura aleatória enquanto os monitores de exibição matricial empregam a varredura regular.

- **MONTORES DE PLASMA:**

Nas telas de plasma gases nobres contidos em minúsculas células revestidas com fósforo, nas cores primárias, são ionizadas por um campo elétrico e agem como lâmpadas fluorescentes microscópicas, emitindo luz.

As células de uma tela de plasma são organizadas em uma matriz de milhares de pontos (*pixels*). Cada *pixel* é composto por três células que emitem luz em diferentes comprimentos de onda. O microprocessador associado à tela envia energia elétrica individualmente a cada célula, no mínimo 24 vezes por segundo, para criar a imagem. Variando-se a intensidade da corrente elétrica aplicada a cada célula, varia-se também a intensidade da luz emitida, obtendo até 68 bilhões de cores (36 bits). A Figura 2.14 apresenta a representação esquemática de uma tela de plasma.

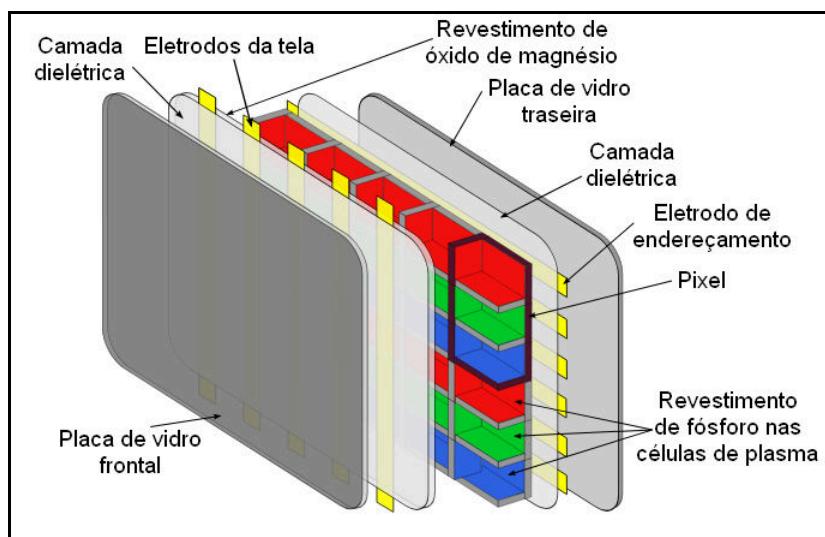


Figura 2.14 – Representação esquemática de uma tela de plasma

Fonte: Adaptado de Laamanen, J. (<https://commons.wikimedia.org/w/index.php?curid=1829066>)

As telas de plasma possuem um ângulo de visão semelhante ao dos televisores (CRT), pois seu princípio básico de funcionamento não depende da polarização da luz, como acontece nas telas LCD.

- **MONTORES DE LCD:**

Este tipo de monitor usa um material que, apesar de não ser líquido, comporta-se como tal. Por isso, ficou conhecido como cristal líquido. Em sua constituição mais simples, as moléculas de cristal líquido são distribuídas entre duas lâminas transparentes polarizadas chamadas substratos. Este processo é orientado

de maneira diferente nas duas lâminas, de forma que estas formem eixos polarizadores perpendiculares, como se formassem um ângulo de 90°. Simplificadamente é como se uma lâmina recebesse polarização horizontal, enquanto a outra polarização vertical, formando um esquema do tipo “linhas e colunas”.

As moléculas de cristal líquido são capazes de orientar a luz. Quando uma imagem é exibida em um monitor LCD, elementos elétricos presentes nas lâminas geram campos magnéticos que induzem o cristal líquido a orientar a luz oriunda da fonte luminosa para formar o conteúdo visual. Sempre que necessário, uma tensão diferente pode ser aplicada, fazendo com que as moléculas de cristal líquido se alterem de maneira a impedir a passagem da luz. A Figura 2.15 ilustra as diferentes camadas de uma tela LCD.

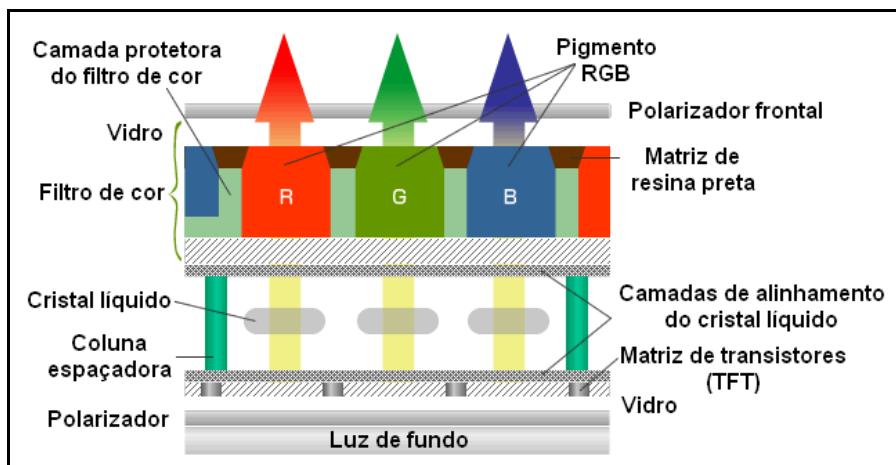


Figura 2.14 – Camadas componentes de uma tela LCD

- **MONITORES OLED E AMOLED:**

OLED e AMOLED são duas tecnologias que guardam semelhanças com o LED (*Light Emitting Diode*), mas diferem em sua composição, pois se trata de um material orgânico (carbono) que emitem luz quando recebem uma carga elétrica. Esses diodos são bastante pequenos; assim, cada pixel da tela pode receber este material de modo a ser iluminado individualmente.

Os OLED são capazes de gerar luz, dispensando a necessidade de retroiluminação presente nas telas LCD. Por isso as telas OLED são muito mais finas que as LCD; tão finas que é possível produzir telas flexíveis. Também são mais eficientes no consumo de energia, geram cores mais nítidas, suportam maior ângulo de visão e oferecem menor tempo de resposta.

A principal diferença entre OLED e AMOLED é que telas compostas com esta última tecnologia são do tipo matriz ativa. As telas OLED com matriz passiva são orientadas com um esquema de transistores organizados em linhas e colunas, assim como acontece com as telas LCD passivas; em telas AMOLED, os transistores são aplicados pixel a pixel usando uma camada de TFT, o que deixa sua fabricação um pouco mais complexa. Mas, neste processo, surgem várias vantagens como telas com tempo de resposta ainda menor e cores mais vivas.

A **memória de exibição** armazena os dados que vão ser utilizados para gerar a imagem. Através do conversor digital analógico os valores armazenados nessa memória são convertidos para uma voltagem que é utilizada pelo canhão para

gerar o feixe de elétrons, no caso de monitores CRT, ou acionar os circuitos responsáveis por iluminar cada pixel, no caso das demais tecnologias. Nos dispositivos matriciais, a memória de exibição, chamada de **frame buffer**, é organizada em uma estrutura matricial de modo a armazenar os valores de cada cor dos pixels da imagem. O tamanho da memória de exibição determina a resolução de cor e a resolução geométrica da imagem. A tabela 2.1 apresenta o número de bits, por pixel, necessários para representar determinadas quantidades de cores.

Tabela 2.1 – Número de bits, por pixel, necessários para representar diferentes quantidades de cores

Número de Cores	Número de Bits	Número de Bytes
2	1	1/8
16	4	½
256	8	1
65536	16	2
16777216	24	3

Nos dispositivos vetoriais a memória de exibição contém instruções de desenho com as coordenadas de tela dos objetos gráficos. Este conjunto de instruções, denominado de **lista de exibição** ("display list"), é executado ininterruptamente pela controladora de vídeo para manter a imagem visível na tela.

2.7.2 DISPOSITIVOS DE SAÍDA VETORIAL

Os dispositivos de saída vetorial produzem imagens traçando segmentos de reta, descritos pelas coordenadas de seus pontos iniciais e finais. Nesta categoria de equipamentos temos: o display caligráfico, o display de armazenamento e os plotadores.

O **display caligráfico** é um dispositivo de exibição de vídeo interativo. O sistema de varredura é aleatório e o fósforo do monitor de baixa persistência, ou seja, o feixe de elétrons se movimenta livremente sobre a tela e a imagem precisa ser regenerada constantemente. Essas características permitem a manipulação em tempo real dos dados.

O **display de armazenamento** ou DVST ("Direct View Storage Tube") dispõe de um monitor de vídeo de alta persistência. Nesse equipamento a imagem traçada é mantida na tela através de um circuito especial, sem necessidade de regeneração. Uma desvantagem desse tipo de monitor é que partes da imagem não podem ser modificadas sem que a imagem inteira seja apagada e redesenhada. Esses monitores têm uma importância histórica em computação gráfica: devido ao baixo custo, pelo fato de não necessitar de uma memória de exibição, ele deu início à grande expansão das aplicações gráficas da computação gráfica nas diversas áreas do conhecimento humano.

Os **plotadores** são equipamentos eletromecânicos para o desenho de linhas sobre papel ou filme. Esse equipamento é constituído por um suporte para superfície de desenho, e um mecanismo de controle do instrumento de traçado. Quanto ao suporte os plotadores podem ser de **mesa** ou de **tambor**. Quanto ao mecanismo de desenho os plotadores podem ser do tipo **contínuo** ou **incremental**.

2.7.3 DISPOSITIVOS DE SAÍDA MATRICIAL

Os dispositivos de saída matricial produzem imagens a partir de uma imagem digital em geral codificada como uma matriz de intensidades. Nesta categoria de equipamentos temos: o *display raster*, o painel de plasma, o display de cristal líquido, as impressoras gráficas e o *film recorder*.

O ***display raster*** consiste de um monitor, um controlador de vídeo e uma memória de exibição que armazena a imagem digital codificada em forma matricial. Projetores de vídeo também podem ser utilizados nos displays raster em substituição aos monitores.

O ***painel de plasma*** é constituído por uma matriz de células microscópicas de neon não precisando, portanto, de memória adicional nem de regeneração de imagem.

O ***display de cristal líquido*** é semelhante ao painel de plasma, mas utiliza células de cristal líquido na matriz de imagem. Essa tecnologia tornou-se padrão de mercado, pois são capazes de apresentar imagens em alta qualidade, com tela plana e baixo consumo de energia.

As ***impressoras de impacto*** são destinadas principalmente para saída alfanumérica. Algumas delas, do tipo "dot matrix" tem capacidade gráfica.

As ***impressoras gráficas*** podem utilizar uma técnica de reprodução eletrostática, térmica ou por jato de tinta. As ***impressoras eletrostáticas*** utilizam um feixe de laser, ou de raio infravermelho, para formar a imagem em um cilindro carregado eletricamente. Partículas de *toner* atraídas para a superfície do cilindro são em seguida transferidas para o papel. As ***impressoras térmicas*** transferem seletivamente pigmentos de tinta para o papel por calor. As ***impressoras de jato de tinta*** possuem uma cabeça de impressão que lança gotículas de tinta no papel por meio de pequenos jatos. As impressoras gráficas podem ser monocromáticas ou coloridas.

O ***film recorder*** registra em película fotográfica imagens geradas por computador. Para reproduzir a cor o filme é exposto três vezes, através de filtros, para as componentes de cor azul, verde e vermelho.

2.7.4 DISPOSITIVOS TRIDIMENSIONAIS

Além dos dispositivos de saída gráfica convencionais, que operam em duas dimensões, existem alguns equipamentos que produzem a ilusão de tridimensionalidade e outros que produzem formas tridimensionais reais. Estes equipamentos podem ser considerados dispositivos gráficos tridimensionais. Dentre estes devemos destacar: o *display estereoscópico*, o "*head-mounted display*", as impressoras tridimensionais e as máquinas ferramenta de controle numérico. Os dois primeiros são utilizados em aplicações científicas e simulações de "realidade virtual", enquanto os dois últimos são utilizados em sistemas de CAD/CAM (*Computer Aided Design/Computer Aided Manufacturing*).

O ***display estereoscópico*** consiste de um dispositivo de exibição de vídeo, acoplado a um sistema que permite mostrar alternadamente, e em rápida sucessão, imagens correspondentes aos olhos direito e esquerdo do observador. O sistema mais comum emprega um obturador eletrônico, colocado na frente da tela do monitor, que polariza a luz sucessivamente em duas direções diferentes. Para perceber o efeito de tridimensionalidade o observador deve usar óculos com lentes polarizadas.

O ***head-mounted display*** consiste de um par de monitores de vídeo montados em um conjunto que é adaptado à cabeça do observador, de modo que a imagem de cada monitor seja vista por apenas um de seus olhos. Em geral, esse equipamento inclui um sensor de posição para possibilitar a geração de imagens correspondendo ao ponto de vista de um observador em movimento.

Os ***dispositivos de impressão tridimensional*** produzem modelos sólidos reais; isto é, verdadeiras esculturas, a partir dos objetos modelados no computador. A geração da escultura se baseia em um processo de varredura espacial, que realiza a forma tridimensional por camadas. Existem dois tipos de tecnologia para essa finalidade: a estereolitografia e a deposição por fusão. O processo de estereolitografia emprega um feixe de laser que solidifica um polímero sensível à luz ultravioleta. O processo de deposição por fusão utiliza um filamento termoplástico que é depositado por um mecanismo de extrusão.

As ***máquinas ferramenta de controle numérico*** são equipamentos tradicionais de fabricação, tais como tornos, fresadeiras e laminadoras que podem ser comandadas por computador. A forma final é esculpida a partir de blocos sólidos de material por instrumentos de corte de precisão.

2.8 EQUIPAMENTOS DE PROCESSAMENTO GRÁFICO

Dois problemas recorrentes no equacionamento da arquitetura dos equipamentos gráficos de processamento estão relacionados com aspectos de funcionalidade e acoplamento. O primeiro aspecto diz respeito ao grau de especialização das funções do processador gráfico. Processadores especializados são mais eficientes e caros, enquanto processadores de propósito geral são mais flexíveis e baratos. O segundo aspecto diz respeito ao canal de comunicação do processador gráfico com o sistema de computação. Um alto acoplamento possibilita um acesso rápido aos dados do sistema, mas implica em grande interdependência. Em contrapartida, um baixo acoplamento permite maior independência entre o processador gráfico e o sistema, mas implica em uma comunicação restrita entre eles.

Estes aspectos estão claramente inter-relacionados. Um processador gráfico especializado necessita de um canal de comunicação de alta capacidade porque muitas das funções gráficas serão realizadas pelo processador principal. Por outro lado, um processador gráfico mais geral dispõe de recursos para executar localmente grande parte das funções de visualização, podendo ter um canal de comunicação limitado com o computador principal.

Existe uma tendência na indústria de equipamentos gráficos que consiste em gradativamente adicionar funcionalidade aos processadores gráficos especializados, até que eles se tornem equivalentes a um computador de uso geral. A única opção que existe então, é completar o ciclo voltando à especialização. Este fenômeno é conhecido na literatura sob a denominação de a “roda da reencarnação”.

2.8.1 DISPOSITIVOS DE PROCESSAMENTO VETORIAL

Os dispositivos do tipo vetorial se destinam principalmente ao processamento de modelos geométricos. Eles atuam, portanto, sobre as coordenadas dos diversos componentes dos modelos, tais como segmentos de reta,

polígonos e etc. Em função dos processadores podemos ter dispositivos do tipo SISD (“*single-instruction, single data stream*”), ou MISD (“*multiple-instruction, single data stream*”).

Os dispositivos do tipo SISD são uniprocessadores que possuem instruções especiais para processamento de dados geométricos, do tipo multiplicação de matrizes por vetores.

Os dispositivos do tipo MISD são *pipelines* compostas de vários processadores organizados sequencialmente. O processamento gráfico é dividido em etapas onde cada processador é especializado numa classe de operações gráficas, como projeção, recorte, etc.

2.8.2 DISPOSITIVOS DE PROCESSAMENTO MATRICIAL

Os dispositivos do tipo matricial são equipamentos multiprocessadores utilizados para o processamento de imagens, para a rasterização e outros algoritmos gráficos paralelizáveis. Podemos ter dispositivos do tipo SIMD (“*single-instruction, multiple data stream*”), ou MIMD (“*multiple-instruction, multiple data stream*”) com diferentes configurações dos processadores.

Os dispositivos do tipo SIMD são utilizados para realizar a mesma operação em vários elementos simultaneamente. Os dispositivos do tipo MIMD são processadores paralelos que se comunicam entre si. A maneira como eles estão interligados define uma topologia de rede, e consequentemente o fluxo de dados.

UNIDADE 3 – MODELAGEM DE SÓLIDOS, CURVAS E SUPERFÍCIES

Chamamos de modelagem às técnicas utilizadas para gerar imagens de cenas tridimensionais a partir de um modelo. Uma das abordagens que veremos aqui consiste em construir objetos a partir de um conjunto de objetos primitivos (esfera, cilindro, cubo, etc.).

Para gerar imagens tridimensionais é necessário representar os objetos de forma conveniente para que eles possam ser processados pelos algoritmos gráficos. Essas informações incluem:

- as medidas do objeto, por exemplo a localização de seus vértices;
- a estrutura ou topologia da cena: como os pontos são agrupados para formar polígonos e esses para formar cada objeto da cena.

3.1 REPRESENTAÇÃO POLIÉDRICA DOS OBJETOS

Uma maneira tradicional de representar os objetos 3D é através de poliedros, ou seja, figuras com todas as faces planas. Essa forma permite a fácil representação de objetos familiares tais como cubos, paralelepípedos, prismas e etc. Aumentando o número de faces do poliedro é possível aproximar figuras curvas.

Um poliedro pode ser modelado pela definição de suas faces. Cada face é um polígono planar de lados, que são segmentos de retas unindo os vértices.

3.1.1 CURVAS E REGIÕES POLIGONAIS

Sejam p_0, p_1, \dots, p_n pontos distintos do plano. Uma curva poligonal é definida como sendo a união dos segmentos $p_0p_1, p_1p_2, p_2p_3, \dots, p_{n-1}p_n$.

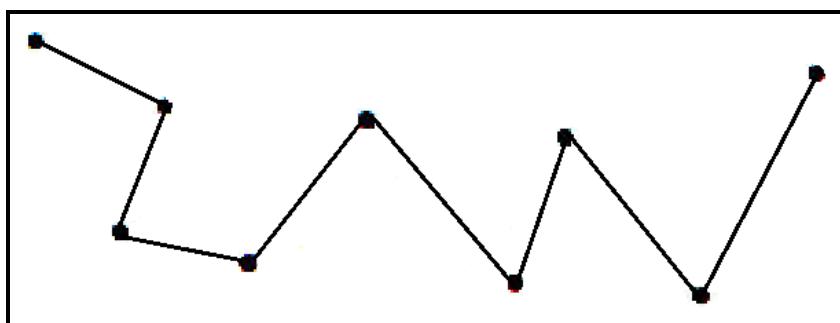


Figura 3.1 – Curva Poligonal

Os pontos p_i são chamados de vértices da curva ou pontos de controle. Os segmentos p_ip_{i+1} são chamados de arestas da curva poligonal.

Curvas planas podem ser aproximadas por curvas poligonais, neste caso são chamadas de retificáveis. Isto significa que dado um erro $\epsilon > 0$, a distância de qualquer ponto da curva poligonal para a curva original c é menor do que ϵ . Para valores suficientemente pequenos de $\epsilon > 0$, isto significa que a curva poligonal contida em uma “vizinhança” de raio ϵ da curva, conforme Figura 3.2.

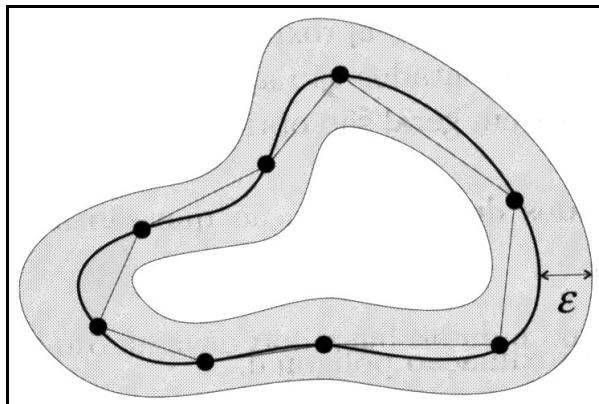


Figura 3.2 – Aproximação de uma curva por uma poligonal

3.2 ARMAZENAMENTO DE POLÍGONOS

Existem basicamente três formas de armazenar polígonos.

3.2.1 VÉRTICES EXPLÍCITOS

Neste modelo, cada polígono é representado por uma lista de coordenadas de vértices.

Para um único polígono esta representação é boa, porém para poliedros, isso representa uma duplicação de coordenadas de vértices compartilhados. Além disso, não há representação explícita dos vértices e arestas compartilhadas.

3.2.2 POLÍGONOS DEFINIDOS POR PONTEIROS PARA LISTA DE VÉRTICES

Nesta representação, cada vértice do conjunto de polígonos é guardado uma vez só numa lista de vértices. Um polígono é definido por uma lista de ponteiros para a lista de vértices.

Nesta representação gasta-se menos memória, e ainda é possível trocar as coordenadas de um vértice, mantendo a conectividade.

O problema desta representação é que ainda é difícil definir os polígonos que compartilham lados, e frequentemente quando se vai desenhar o objeto, cada aresta acaba sendo desenhada duas vezes.

3.2.3 ARESTAS EXPLÍCITAS

Neste caso nós temos 2 listas: a lista de vértices e a lista de arestas. Na lista de arestas, cada aresta aponta para 2 vértices. Cada polígono então é uma lista de ponteiros para a lista de arestas.

Assim, por exemplo, é fácil mostrar o contorno do poliedro, simplesmente varrendo a lista de arestas. Por outro lado, também é fácil gerar um polígono em particular.

Em nenhuma destas representações é fácil saber que arestas incidem num vértice: toda a lista deve ser varrida. Por isso, é usual incluir ponteiros adicionais para obter esta informação, que é necessária em alguns algoritmos de coloração.

3.3 REPRESENTAÇÃO DE SÓLIDOS POR APROXIMAÇÕES

A representação de sólidos pode ser feita através da aproximação dos objetos utilizando-se de técnicas apropriadas. Dentre estas técnicas temos a Geometria Sólido Construtiva (CSG), a Representação de Fronteiras (B-rep), a Modelagem Baseada em Características (Sweep).

3.3.1 MODELAGEM POR GEOMETRIA SÓLIDO CONSTRUTIVA (CSG)

A modelagem CSG tem por base as primitivas geométricas simples como os prismas, cones, cilindros, esferas, etc. (Figura 6.3). É possível criar objetos mais complexos através da combinação e do posicionamento destas primitivas utilizando-se das operações booleanas:

- união (adição);
- diferença (subtração);
- interseção.

A representação dos objetos criados através de modelagem CSG utiliza-se de árvores booleanas (Figura 3.4).

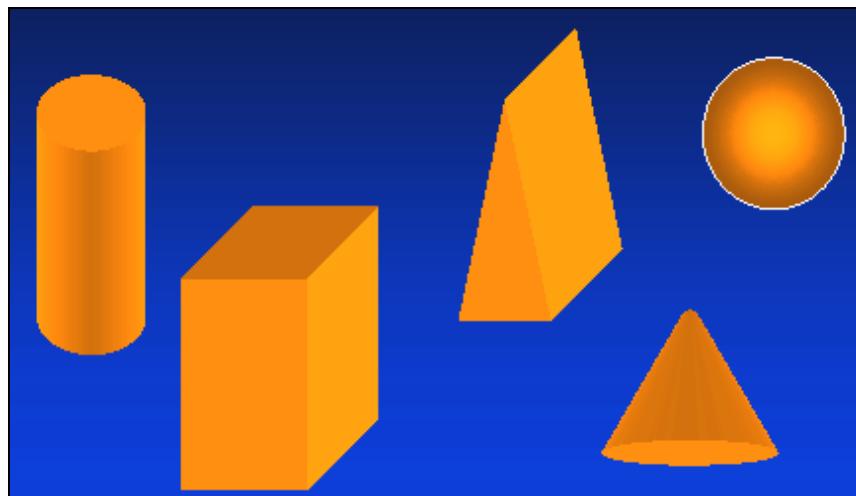


Figura 6.3 – Primitivas básicas para a modelagem CSG

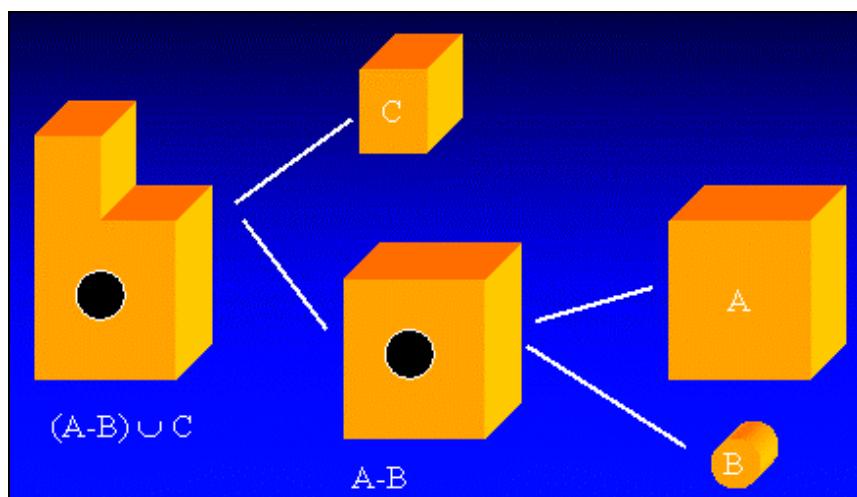


Figura 3.4 – Representação de um objeto criado através de modelagem CSG

3.3.2 MODELAGEM POR REPRESENTAÇÃO DE FRONTEIRAS (B-REP)

Neste tipo de modelagem os sólidos são representados por suas faces, vértices e arestas. Este tipo de modelagem exige que as superfícies dos sólidos sejam fechadas. A Figura 3.5 mostra um exemplo de um sólido modelado através da representação de fronteiras.

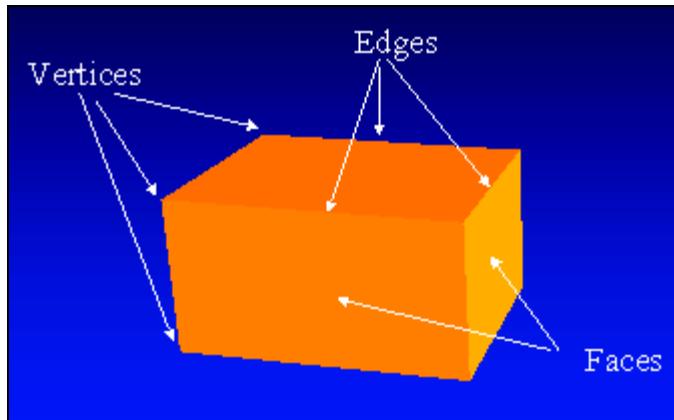


Figura 3.5 – Uma caixa modelada através da representação de fronteiras

O armazenamento e a representação de objetos modelados através da representação de fronteiras utiliza-se dos métodos descritos no item 3.2.

3.3.3 MODELAGEM BASEADA EM CARACTERÍSTICAS (SWEEP)

A modelagem base em características dos objetos é uma técnica que combina aspectos da modelagem CSG e da modelagem B-rep.

Neste processo de modelagem as partes são modeladas pela adição de características a um objeto base. Estas características representam operações de manufatura como furos, nervuras, filetes, chanfros, ranhuras, etc.

Estas operações de manufatura podem tanto adicionar como retirar material do objeto base, de forma similar ao que acontece na modelagem CSG. As características não são limitadas às primitivas simples e podem ser criadas por extrusão, revolução, etc.

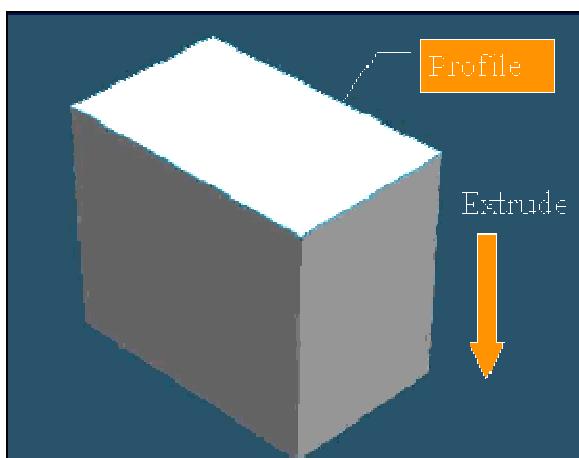


Figura 3.6 – Uma caixa gerada através do processo de extrusão.

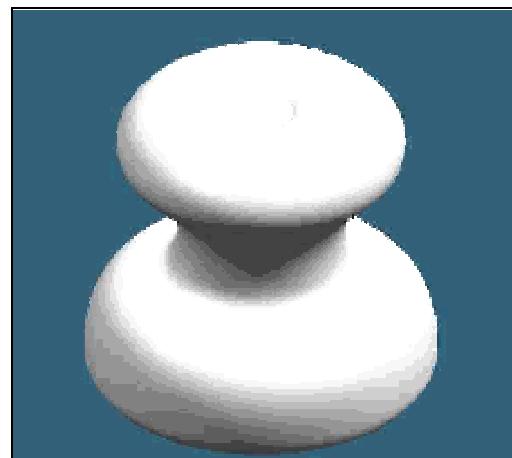


Figura 3.7 – Uma peça gerada através de revolução

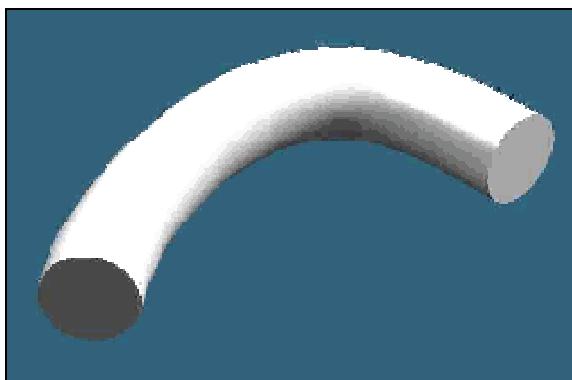


Figura 3.8 – Uma peça gerada por deslizamento

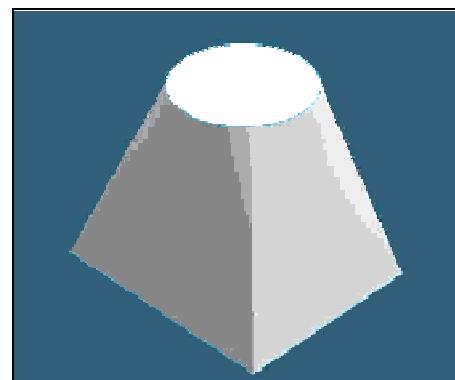


Figura 3.9 – Um objeto gerado por *loft*

A representação de um objeto criado pela modelagem baseada em características utiliza-se de uma árvore similar à árvore booleana utilizada na modelagem CSG. Esta árvore conterá, numa sequência de passos, as características adicionadas ao objeto base.

O processo baseia-se em criar um objeto base e a este objeto adicionar características até a obtenção da forma final desejada.

O processo de adição de características baseia-se na criação de uma forma num plano (2D) que é estendida para o espaço (3D). Os processos utilizados para transformar um objeto 2D em 3D são: extrusão, revolução, deslizamento e *loft*. Estas operações tanto podem adicionar como remover partes do objeto base.

3.3.4 ENUMERAÇÃO DA OCUPAÇÃO ESPACIAL

Neste método um sólido é decomposto em células idênticas e arranjadas num grid regular. Estas células são chamadas de voxels. A Figura 3.10 mostra um objeto representado através da enumeração da ocupação espacial.

Quando representamos um objeto usando enumeração da ocupação espacial controlamos somente a presença ou ausência da célula em cada posição do grid. O objeto pode então ser armazenado numa lista de células onde indicamos quais estão ocupadas ou não.

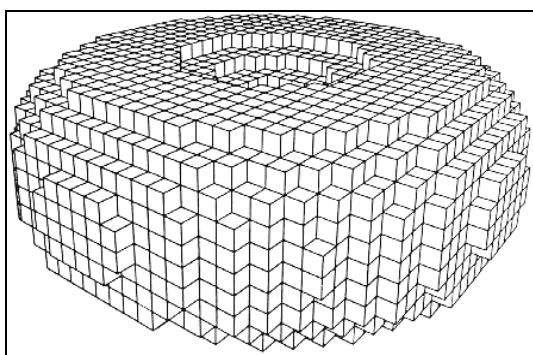


Figura 3.10 – Um objeto representado através da enumeração da ocupação espacial

3.3.5 OCTREES

As Octrees são variantes da enumeração da ocupação espacial, projetadas para serem compatíveis com os requisitos dos dispositivos de

armazenamento. As *Octrees* derivam das *Quadtrees*, uma técnica empregada no armazenamento de imagens. A Figura 3.11 mostra a diferença na representação de uma imagem utilizando a enumeração da ocupação espacial e uma *Quadtree*.

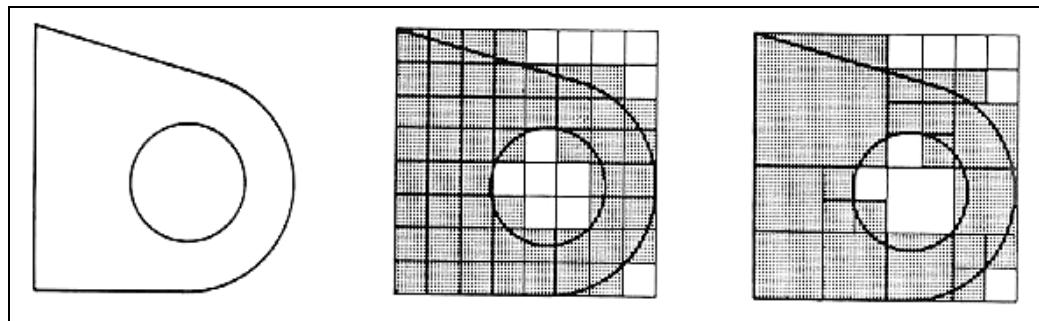


Figura 3.11 –O objeto; sua representação por enumeração da ocupação espacial e sua representação através de *Quadtree*

A representação em memória desta imagem utilizando-se de uma *Quadtree* é apresentada na Figura 3.12.

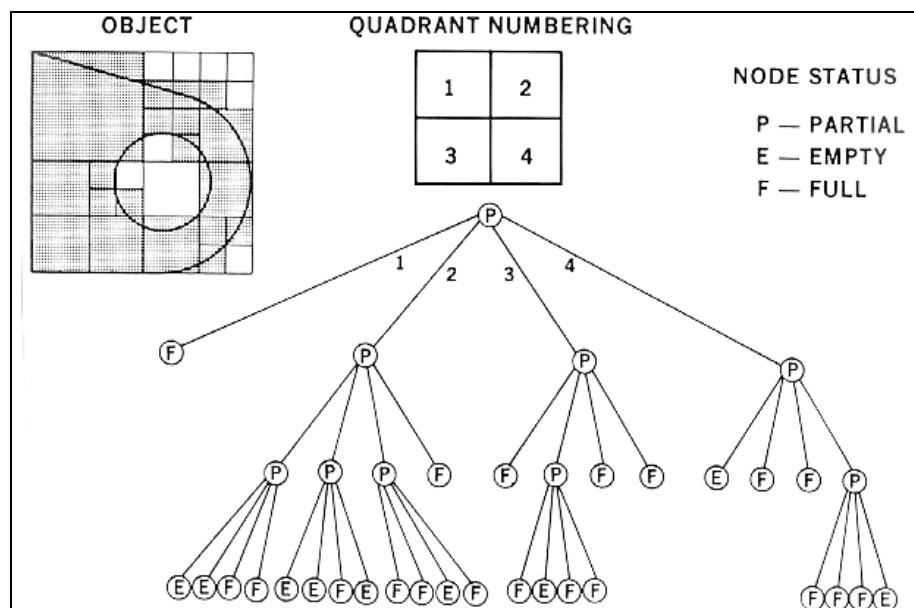


Figura 3.12 – Representação em memória de uma *Quadtree*

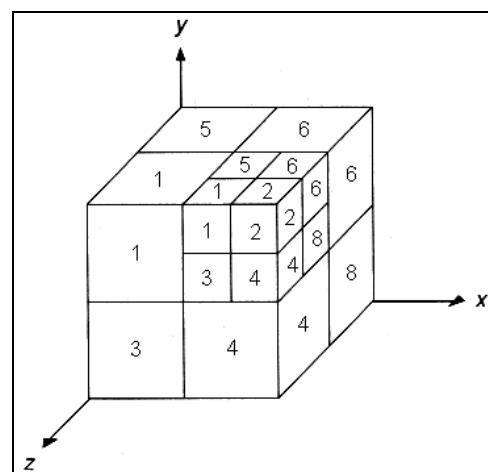


Figura 3.13 – A divisão em octantes de uma *Octree*

As Octrees são similares às Quadtrees, excetuando que as três dimensões são recursivamente divididas em octantes, como mostra a Figura 3.13. Os quadrantes continuam sendo numerados de 1 a 4, enquanto que os octantes são numerados de 1 a 8.

3.4 COMPARAÇÃO DAS REPRESENTAÇÕES

REQUICHA (1980) propôs uma lista de propriedades desejáveis num esquema para representação de sólidos.

O **domínio** da representação deve ser grande o suficiente para permitir a representação de um conjunto útil de objetos físicos. De acordo com este critério, a modelagem baseada em características (*sweep*) é limitada. O método da enumeração espacial e *octrees* pode representar qualquer tipo de sólido, embora frequentemente seja utilizado para representar apenas aproximações deles. Incluindo outros tipos de faces e arestas (não apenas polígonos com linhas retas) a modelagem por representação de fronteiras (B-rep) pode ser utilizada para representar uma ampla gama de objetos.

Uma representação é **única** se ela pode ser usada para codificar qualquer sólido de apenas uma maneira. Somente a enumeração espacial e as *octrees* proporcionam formas únicas de representar sólidos.

A **precisão** permite representar um objeto sem aproximações. A enumeração espacial e a representação por fronteiras (B-rep) permitem representar somente aproximações em muitos casos. As outras técnicas permitem uma melhor representação dos objetos, principalmente quando trabalhamos com altas resoluções.

As técnicas de modelagem devem proporcionar a criação de objetos **válidos**. A modelagem por representação de fronteiras é a mais crítica quanto ao critério validade; podem ocorrer casos de faces, arestas e vértices inconsistentes, bem como interseção entre arestas e faces. As técnicas de *octrees* e CSG permitem facilmente verificar a validade dos objetos representados. A enumeração espacial não necessita de verificação de validade.

Uma representação deve ser **compacta** e **eficiente**. A eficiência está diretamente relacionada com os algoritmos utilizados nas diferentes técnicas. A avaliação da eficiência de um método de representação está relacionada com a interpretação das informações para manipular os sólidos. Se uma técnica tem que processar as informações para manipular o sólido ela perde em eficiência. A modelagem CSG e Octrees tem de processar as informações, enquanto que a representação por fronteiras (B-rep) e a enumeração espacial não necessitam processar as informações. Portanto as últimas podem ser consideradas mais eficientes.

Os critérios anteriores são relativos. Por exemplo, detectar se um ponto é interno a um sólido é um processo mais simples em CSG do que em B-rep.

3.5 MODELAGEM DE CURVAS

Uma curva $Q(t) = [x(t) \quad y(t) \quad z(t)]$, com $0 \leq t \leq 1$, pode ser escrita como:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

Chamando $T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$ e a matriz $C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$, podemos

reescrever $Q(t)$ como:

$$Q(t) = T \cdot C$$

Derivando a equação acima temos os vetores tangentes à curva $Q(t)$:

$$\begin{aligned} \frac{d}{dt} Q(t) &= \frac{d}{dt} T \cdot C = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot C \\ &= \begin{bmatrix} 3a_x t^2 + 2b_x t + c_x & 3a_y t^2 + 2b_y t + c_y & 3a_z t^2 + 2b_z t + c_z \end{bmatrix} \end{aligned}$$

Nestas condições:

- Se duas curvas apresentam um ponto de junção dizemos que esta curva apresenta Continuidade Geométrica C^0 ;
- Se as direções dos vetores tangente no ponto de junção forem iguais, a curva apresenta Continuidade Geométrica C^1 ;
- Se as direções e as magnitudes dos vetores tangentes no ponto de junção forem iguais, a curva é contínua e é dita paramétrica contínua C^2 .

Reescrevendo $C = M \cdot G$ temos:

$$C = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix},$$

onde M é chamada de matriz base e G é o vetor de geometria formado pelas pontos finais e pelos vetores tangentes que definem a curva.

Assim:

$$\begin{aligned} Q(t) &= [x(t) \quad y(t) \quad z(t)] = \\ Q(t) &= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} \end{aligned}$$

considerando apenas x teremos:

$$\begin{aligned}x(t) = & (t^3 m_{11} + t^2 m_{21} + t m_{31} + m_{41}) g_1 x + (t^3 m_{12} + t^2 m_{22} + t m_{32} + m_{42}) g_2 x + \\& + (t^3 m_{13} + t^2 m_{23} + t m_{33} + m_{43}) g_3 x + (t^3 m_{14} + t^2 m_{24} + t m_{34} + m_{44}) g_4 x\end{aligned}$$

expressões similares devem ser construídas para $y(t)$ e $z(t)$.

3.5.1 REPRESENTAÇÃO POR HERMITE

O uso de polinômios de terceira ordem para ajuste de curvas foi extensamente descrito por Hermite. Sua aplicação é básica para o entendimento dos demais polinômios de ajuste de curvas. A geometria proposta por Hermite propõe um interpolador local controlado por 4 fatores a cada intervalo de 2 pontos: os próprios pontos de entrada e saída da curva (P_0, P_1) e os vetores tangentes à curva nestes pontos, definidos como (T_0, T_1) . Estes fatores de controle tem seu peso de participação na composição da geometria de Hermite balancedo conforme apresentado na Figura 3.14.

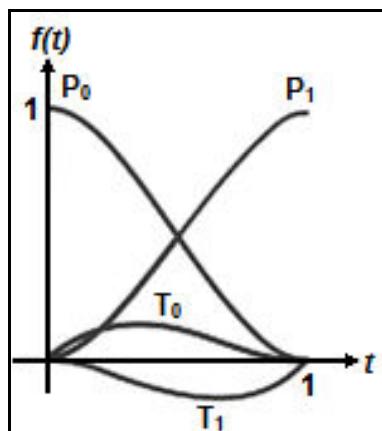


Figura 3.14 – Geometria de composição da função de Hermite

Uma leitura pragmática deste diagrama nos mostra que para um dado instante (t) qualquer, as coordenadas do que geram a curva Hermite são resultados da soma das coordenadas de cada um dos fatores de controle (P_0, P_1, T_0, T_1) ponderados pelas curvas da geometria de Hermite. As curvas de ponderação da geometria de Hermite são definidas por polinômios de terceira ordem com a seguinte forma geral:

$$x(t) = P_x = a_x t^3 + b_x t^2 + c_x t + d_x$$

Assim sendo, tem-se que a solução da geometria de Hermite consiste na determinação dos valores de (a, b, c, d) para cada uma das curvas de ponderação (P_0, P_1, T_0, T_1) . Sabe-se, em função da geometria proposta, que para $(t = 0)$ tem-se:

$$P_{0x} = a_x 0^3 + b_x 0^2 + c_x 0 + d_x$$

ou na forma matricial:

$$M_{Hx} = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$$

Sabe-se, em função da geometria proposta, que para ($t = 1$) tem-se:

$$P_{1x} = a_x 1^3 + b_x 1^2 + c_x 1 + d_x$$

ou na forma matricial:

$$M_{Hx} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

Sabe-se que as tangentes em p_0 e p_1 podem ser obtidas pela derivada da equação da curva no ponto ($x(t) = P_x = a_x t^3 + b_x t^2 + c_x t + d_x$), como se segue:

- Para $t = 0$:

$$T_{0x} = 3a_x 0^2 + 2b_x 0 + c_x + 0$$

ou na forma matricial:

$$M_{Hx} = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}$$

- Para $t = 1$:

$$T_{1x} = 3a_x 1^2 + 2b_x 1 + c_x + 0$$

ou na forma matricial:

$$M_{Hx} = \begin{bmatrix} 3 & 2 & 1 & 0 \end{bmatrix}$$

Para que a geometria de Hermite seja válida é necessário que a relação abaixo seja verdadeira, onde \mathbf{M}_H é a incógnita do problema:

$$\begin{bmatrix} P_0 \\ P_1 \\ T_0 \\ T_1 \end{bmatrix}_x = G_{Hx} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{Hx}$$

Para que esta relação seja válida, também em y e z , é necessário que \mathbf{M}_H seja a matriz inversa da matriz acima, anulando o produto com a matriz identidade resultante, ou seja, \mathbf{M}_H (matriz de Hermite) é definida por:

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

A função geradora da curva de Hermite encontra-se detalhada no código fonte a seguir, sendo este a aplicação direta da matriz M_H .

```
i = 0;

while(i+1 < TotMarks) { //TotMarks = número total de pontos na curva

    RangeX = fabs (X[i+1] - X[i]);
    RangeY = fabs (Y[i+1] - Y[i]);

    if(RangeX > RangeY)
        Step = 1.0/RangeX;
    else
        Step = 1.0/RangeY;

    //Determinação automática das tangentes
    if(i == 0) {
        T1X = X[i+1] - X[i];
        T2X = X[i+2] - X[i];
        T1Y = Y[i+1] - Y[i];
        T2Y = Y[i+2] - Y[i];
    }
    else if (i != 0 && i != TotMarks-2) {
        T1X = X[i+1] - X[i-1];
        T2X = X[i+2] - X[i];
        T1Y = Y[i+1] - Y[i-1];
        T2Y = Y[i+2] - Y[i];
    }
    else {
        T1X = X[i+1] - X[i-1];
        T2X = X[i+1] - X[i];
        T1Y = Y[i+1] - Y[i-1];
        T2Y = Y[i+1] - Y[i];
    }

    WG = 0.5;

    for(t = 0; t <= 1; t += Step) {
        x = 0.5 + (( 2*pow(t,3) -3*pow(t,2) +0*t +1) * X[i] +
                    (-2*pow(t,3) +3*pow(t,2) +0*t +0) * X[i+1] +
                    ( 1*pow(t,3) -2*pow(t,2) +1*t +0) * WG*T1X +
                    ( 1*pow(t,3) -1*pow(t,2) +0*t +0) * WG*T2X);

        y = 0.5 + (( 2*pow(t,3) -3*pow(t,2) +0*t +1) * Y[i] +
                    (-2*pow(t,3) +3*pow(t,2) +0*t +0) * Y[i+1] +
                    ( 1*pow(t,3) -2*pow(t,2) +1*t +0) * WG*T1Y +
                    ( 1*pow(t,3) -1*pow(t,2) +0*t +0) * WG*T2Y);

        if(t == 0)
            MoveTo (hdc, x, y);
        else
            LineTo (hdc, x, y);
    }
    LineTo (hdc, X[i+1], Y[i+1]);
    i++;
}
}
```

O controle preciso das tangentes de entrada e de saída da curva é essencial para a "suavização" da curva total. Diz-se que uma curva é contínua à outra se as suas respectivas tangentes de saída e entrada possuem a mesma

direção conforme ilustra a Figura 3.15.

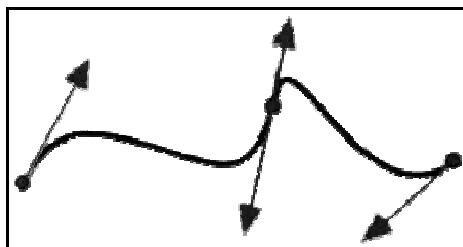


Figura 3.15 - Continuidade dos segmentos em função da direção da tangente

Variações na direção de apenas umas das tangentes produzem resultados interessantes na geração das curvas, conforme apresentado na Figura 3.16.

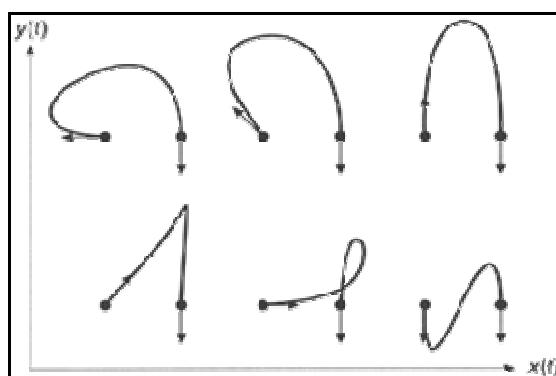


Figura 3.16 - Efeitos da variação na direção da tangente na função de Hermite

Variações no comprimento do vetor tangente afetam a "agressividade" da curva. O módulo do vetor tangente pode ser interpretado como a velocidade no ponto tangente. Os efeitos da variação no módulo da tangente na função de Hermite são apresentados na Figura 3.17.

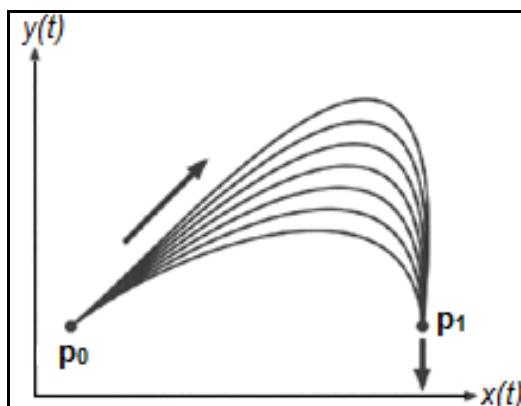


Figura 3.17 - Efeitos da variação no módulo da tangente na função de Hermite

O controle individual das retas tangentes pode não ser interessante quando uma interação manual é desejada. Uma maneira "automática" de controle das retas tangentes, necessárias ao cálculo da função de Hermite, é desejável. Dados 4 pontos definidos por $p_0(x_0, y_0)$, $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ e $p_3(x_3, y_3)$ são geradas 4 tangentes necessárias à geração de 3 segmentos de curva contínuos, como descrito a seguir.

a) Tangente em p_0 :

$$\text{Tangente em } p_0(x_0, y_0) = (T_{0x}, T_{0y})$$

Onde:

$$\begin{aligned} T_{0x} &= x_1 - x_0 \\ T_{0y} &= y_1 - y_0 \end{aligned}$$

b) Tangente em p_1 :

$$\text{Tangente em } p_1(x_1, y_1) = (T_{1x}, T_{1y})$$

Onde:

$$\begin{aligned} T_{1x} &= x_2 - x_0 \\ T_{1y} &= y_2 - y_0 \end{aligned}$$

c) Tangente em p_2 :

$$\text{Tangente em } p_2(x_2, y_2) = (T_{2x}, T_{2y})$$

Onde:

$$\begin{aligned} T_{2x} &= x_3 - x_1 \\ T_{2y} &= y_3 - y_1 \end{aligned}$$

d) Tangente em p_3 :

$$\text{Tangente em } p_3(x_3, y_3) = (T_{3x}, T_{3y})$$

Onde:

$$\begin{aligned} T_{3x} &= x_3 - x_2 \\ T_{3y} &= y_3 - y_2 \end{aligned}$$

Um fator de correção para o "peso" da tangente na composição da curva pode ser aplicado para tornar a curva mais "agressiva". A curva a seguir foi gerada com peso de 50% para as tangentes, conforme ilustrado na Figura 3.18.

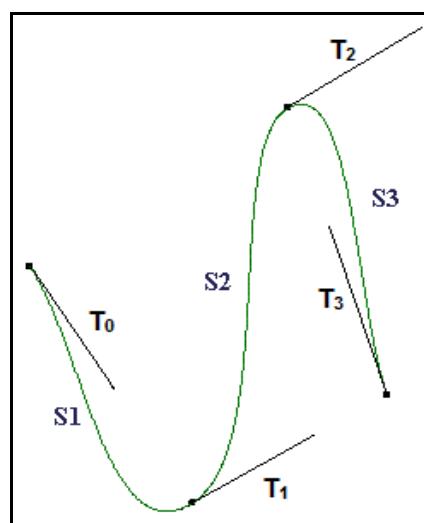


Figura 3.18 - Cálculo automático da tangente para composição de Hermite

3.5.2 REPRESENTAÇÕES POR SPLINE

Uma *spline* é uma linha flexível usada para produzir uma curva suavizada ao longo de uma série de pontos de controle. Existem várias formas de se

representar *splines*, e a sua amostragem varia de acordo com a fórmula matemática utilizada na sua construção.

Existem duas representações principais para os *splines*: as interpoladas onde a curva passa exatamente sobre os pontos, e as aproximadas, onde a curva não, necessariamente, passa por estes pontos, como mostra a Figura 3.19.

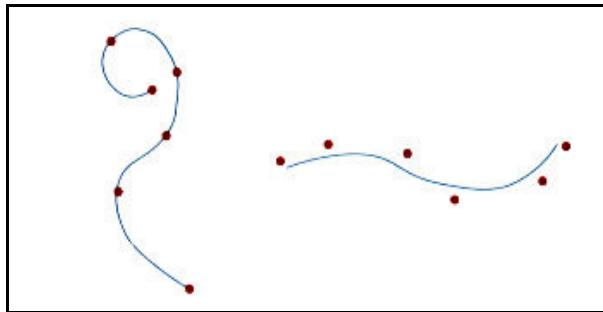


Figura 3.19 – Curva interpolada e curva aproximada

As formas que as *splines* adquirem dependem de como os pontos, ou vértices, são dispostos. Manipulando estes pontos, consequentemente altera-se a curvatura do *spline*. Um *spline* pode ser representado através do polinômio:

$$x(t) = a_xt^3 + b_xt^2 + c_xt + d_x \quad \text{onde } 0 \leq t \leq 1.$$

A matriz para curvas B-Spline (*spline* cúbico), que descreve as curvas de mistura de sua geometria, é definida a seguir:

$$M_{BS} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

A função geradora da curva B-Spline, utilizando 4 pontos de controle e fator de continuidade 4, encontra-se detalhada no código fonte a seguir, sendo esta a aplicação direta da matriz M_{BS} .

```
i = 0;

while(i+3 < TotMarks) { //TotMarks = número total de pontos na curva
    RangeX = fabs (X[i+2] - X[i+1]);
    RangeY = fabs (Y[i+2] - Y[i+1]);

    if(RangeX > RangeY)
        Step = 1.0/RangeX;
    else
        Step = 1.0/RangeY;

    for(t = 0; t <= 1; t += Step) {
        x = (((-1*pow(t,3) + 3*pow(t,2) - 3*t + 1)*X[i] +
               ( 3*pow(t,3) - 6*pow(t,2) + 0*t + 4)*X[i+1] +
               (-3*pow(t,3) + 3*pow(t,2) + 3*t + 1)*X[i+2] +
               ( 1*pow(t,3) + 0*pow(t,2) + 0*t + 0)*X[i+3])/6);

        y = (((-1*pow(t,3) + 3*pow(t,2) - 3*t + 1)*Y[i] +
               ( 3*pow(t,3) - 6*pow(t,2) + 0*t + 4)*Y[i+1] +
               (-3*pow(t,3) + 3*pow(t,2) + 3*t + 1)*Y[i+2] +
               ( 1*pow(t,3) + 0*pow(t,2) + 0*t + 0)*Y[i+3])/6);
    }
}
```

```

        ( 3*pow(t,3) -6*pow(t,2) +0*t +4)*Y[i+1] +
        (-3*pow(t,3) +3*pow(t,2) +3*t +1)*Y[i+2] +
        ( 1*pow(t,3) +0*pow(t,2) +0*t +0)*Y[i+3])/6);

    if(t == 0)
        MoveTo (hdc, x, y);
    else
        LineTo (hdc, x, y);
}
i++;
}

```

Os ajustadores de B-Spline abrigam outra característica interessante. Para gerar-se uma curva convexa, com o mesmo grau de continuidade do restante dos segmentos, basta repetir os 3 primeiros pontos ao final da sequência de pontos de controle.

3.5.3 CURVAS BÉZIER E SUPERFÍCIES BÉZIER

Este tipo de *spline* foi desenvolvido pelo engenheiro francês Pierre Bézier, para desenhar carros da *Renault*. Por serem versáteis, foram aos poucos sendo incorporadas aos programas de desenho, inclusive em programas de modelagem de sólidos.

As curvas Bézier são definidas por dois pontos extremos (p_0 e p_1) e outros dois pontos que controlam os extremos dos vetores tangentes (T_0 e T_1). T_0 e T_1 são definidos a seguir:

a) Tangente em p_0 :

$$\text{Tangente em } p_0(x_0, y_0) = (T_{0x}, T_{0y})$$

Onde:

$$\begin{aligned} T_{0x} &= 3 * (x_1 - x_0) \\ T_{0y} &= 3 * (y_1 - y_0) \end{aligned}$$

b) Tangente em p_1 :

$$\text{Tangente em } p_3(x_3, y_3) = (T_{1x}, T_{1y})$$

Onde:

$$\begin{aligned} T_{1x} &= 3 * (x_2 - x_3) \\ T_{1y} &= 3 * (y_2 - y_3) \end{aligned}$$

A matriz de Bézier, que descreve as curvas de mistura de sua geometria, é definida a seguir:

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

A função geradora da curva de Bézier, para intervalos de 4 pontos,

encontra-se detalhada no código fonte a seguir, sendo este a aplicação direta da matriz M_B .

```
i = 0;

while(i+3 < TotMarks) { //TotMarks = número total de pontos na curva
    RangeX = fabs (X[i+3] - X[i]);
    RangeY = fabs (Y[i+3] - Y[i]);

    if(RangeX > RangeY)
        Step = 1.0/RangeX;
    else
        Step = 1.0/RangeY;

    for (t = 0; t <= 1; t += Step) {
        x = ((-1*pow(t,3) + 3*pow(t,2) - 3*t + 1)*X[i] +
              ( 3*pow(t,3) - 6*pow(t,2) + 3*t + 0)*X[i+1] +
              (-3*pow(t,3) + 3*pow(t,2) + 0*t + 0)*X[i+2] +
              ( 1*pow(t,3) + 0*pow(t,2) + 0*t + 0)*X[i+3]);

        y = ((-1*pow(t,3) + 3*pow(t,2) - 3*t + 1)*Y[i] +
              ( 3*pow(t,3) - 6*pow(t,2) + 3*t + 0)*Y[i+1] +
              (-3*pow(t,3) + 3*pow(t,2) + 0*t + 0)*Y[i+2] +
              ( 1*pow(t,3) + 0*pow(t,2) + 0*t + 0)*Y[i+3]);

        if(t == 0)
            MoveTo (hdc, x, y);
        else
            LineTo (hdc, x, y);
    }
    i += 3;
}
```

Seguem-se alguns exemplos de curvas geradas pelo método de Bézier (Figura 3.20). Note-se que a curva sempre está contida no polígono convexo determinado por seus pontos de controle.

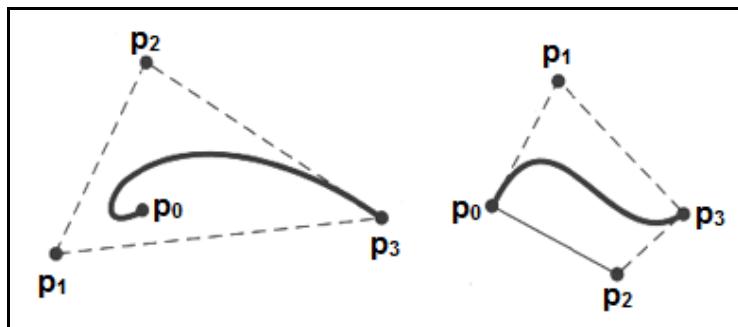


Figura 3.20 – Exemplos de curvas Bézier

Duas curvas Bézier podem ser utilizadas para formar uma superfície Bézier. Esta superfície é formada pelo produto cartesiano das duas funções:

$$P(u, v) = \sum_{j=0}^m \sum_{k=0}^n p_{j,k} BEZ_{j,m}(v) BEZ_{k,n}(u)$$

$$BEZ_{j,m}(v) = C(m, j)v^j(1-v)^{m-j}$$

$$BEZ_{k,n}(u) = C(n, k)u^k(1-u)^{n-k}$$

$$C(m, j) = \frac{m!}{j!(m-j)!}$$

$$C(n, k) = \frac{n!}{k!(n-k)!}$$

Com $p_{j,k}$ especificando a localização de $(m + 1)$ por $(n + 1)$ pontos de controle. A Figura 3.21 demonstra dois exemplos de superfícies geradas com $m = 3$, $n = 3$ e $m = 4$, $n = 4$ pontos de controle.

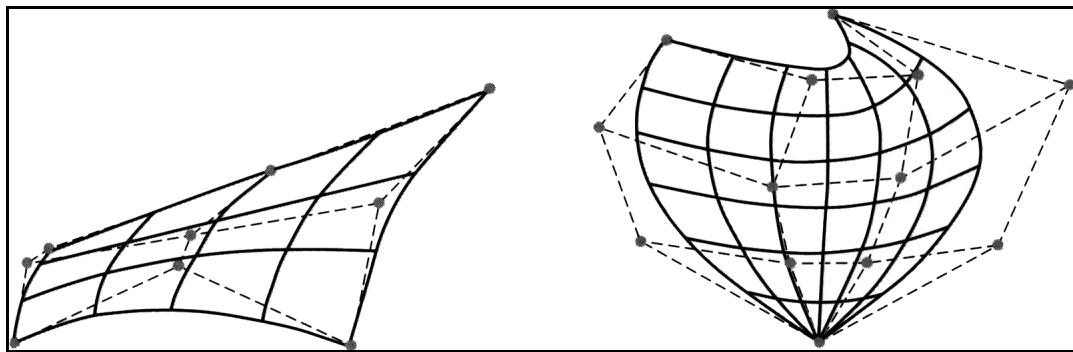


Figura 3.21 – Dois exemplos de superfícies Bézier

UNIDADE 4 – TRANSFORMAÇÕES GEOMÉTRICAS

Transformações Geométricas (TG) são a base de inúmeras aplicações gráficas, podendo estar desde em simples programa 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.

4.1 TRANSFORMAÇÕES 2D

4.1.1 TRANSLAÇÃO

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 dx unidades em relação ao eixo x , e por dy unidades em relação ao eixo y . Logo, o ponto $P'(x', y')$, pode ser escrito como:

$$x' = x + dx \quad e \quad y' = y + dy$$

E se definimos os vetores colunas:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}, \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad T = \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Então podemos escrever:

$$P' = P + T$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

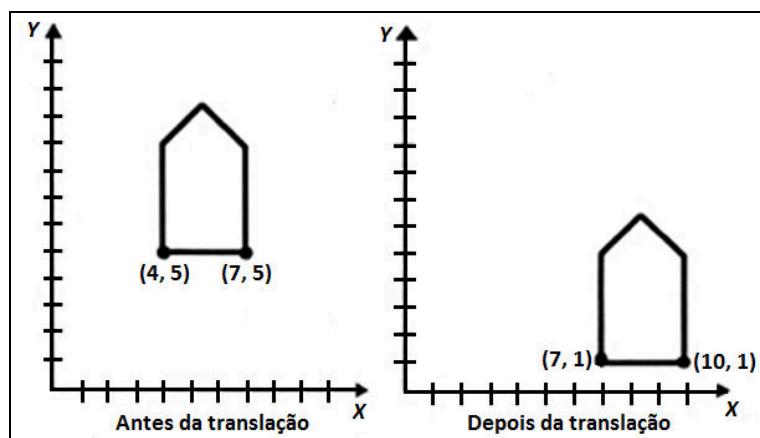


Figura 4.1 – Translação de um objeto

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 4.1 é mostrado o efeito de uma translação de um objeto por (3, -4).

4.1.2 ESCALA

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

$$x' = S_x \cdot x \quad e \quad y' = S_y \cdot y$$

Ou em forma matricial:

$$P' = S \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Na Figura 4.2 a casa sofre uma escala de $\frac{1}{2}$ em x e de $\frac{1}{4}$ em y.

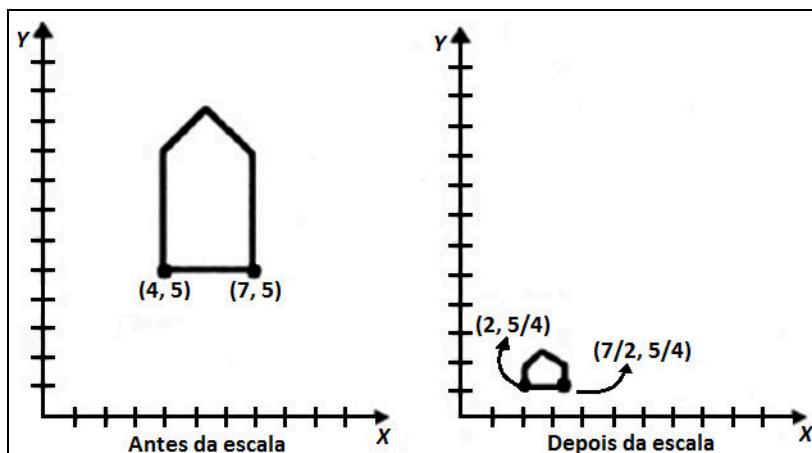


Figura 4.2 – Aplicação de escala de um objeto

Observe que a escala é feita em relação à origem. Assim a casa fica menor e mais próxima da origem. Também as proporções da casa são alteradas, pois a escala utilizada em x (S_x) é diferente da escala em y (S_y). Porém ao se utilizarem escalas uniformes ($S_x = S_y$) as proporções não são afetadas.

4.1.3 ROTAÇÃO

A Rotação de pontos através de um ângulo θ também é feita a partir da origem.

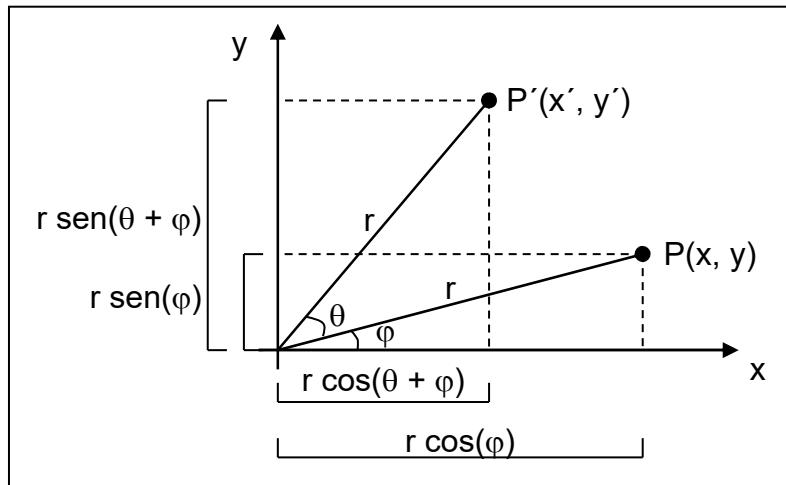


Figura 4.3 - Derivando a equação de rotação

Da Figura 4.3 obtemos as seguintes relações:

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

Assim,

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

Realizando as devidas substituições obtemos que a rotação é definida matematicamente por:

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

Em forma matricial:

$$P' = R \cdot P$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

Da mesma forma que para a escala, a rotação também é feita em relação à origem. 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.

A Figura 4.4 mostra a transformação de rotação (45°) aplicada a um objeto.

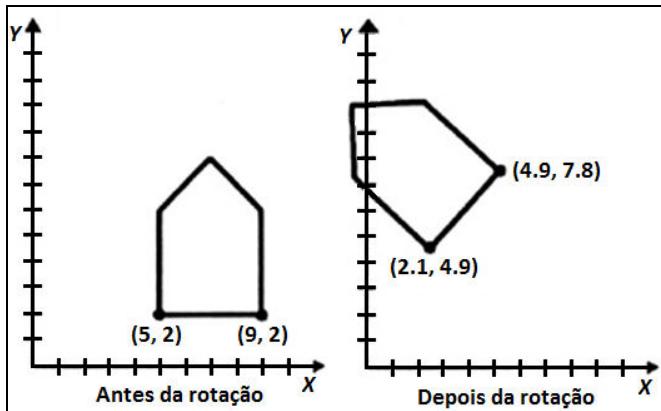


Figura 4.4 – Aplicação de rotação a um objeto

4.2 COORDENADAS HOMOGÊNEAS E MATRIZES DE TRANSFORMAÇÃO

Vimos acima que as operações de translação, rotação e escala podem ser facilmente executadas com o uso de matrizes.

No entanto, enquanto as operações de rotação e escala são multiplicativas, as operações de translação são aditivas. Com o objetivo de otimizar a aplicação destas operações transformadoras foi concebido um sistema de coordenadas denominado coordenadas homogêneas.

Quando tratamos de representar um ponto no espaço 2D no sistema cartesiano fazemos uso das coordenadas (x, y) que posicionam o ponto em relação ao centro de coordenadas. O sistema de coordenadas homogêneas utiliza-se de 3 valores para expressar um ponto: (x, y, M) .

A transposição do sistema homogêneo para o cartesiano se dá pela seguinte relação: $(x, y) = (x/M, y/M)$. Existem, portanto, infinitas coordenadas homogêneas para um mesmo ponto. Os pontos onde $M = 0$ são, por definição, os pontos fora do espaço dimensional.

Tudo isso poderia parecer muito pouco útil não fosse uma pequena característica das operações matriciais que permite expressar a aplicação da matriz de translação como uma multiplicação quando se expressa o ponto no sistema de coordenadas homogêneo. Define-se todo ponto como $(x, y, 1)$ no sistema homogêneo, fazendo com que sua transposição para o espaço cartesiano seja direta (divisão por 1).

Desta forma, as transformações translação, escala e rotação podem ser realizadas a partir das seguintes operações com matrizes.

Translação:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Escala:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotação:

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

4.3 CONCATENAÇÃO DE MATRIZES

O processo de combinar duas matrizes é chamado “concatenação” e é executado multiplicando as duas matrizes multiplicadoras antes de aplicá-la aos multiplicandos. Este processo é especialmente produtivo quando se deseja aplicar certa escala e translação a um conjunto de pontos. Deve-se, no entanto, ter em mente que a ordem de aplicação das transformações afeta o produto afinal, como bem se ilustra a seguir, onde foram aplicados os mesmos fatores de escala e rotação, em ordens inversas, obtendo-se resultados completamente diferentes.

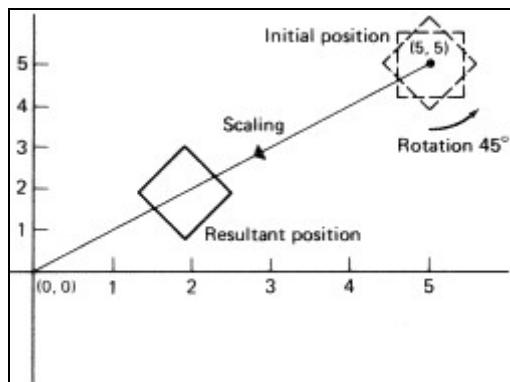


Figura 4.5 – Aplicação de rotação e depois escala.

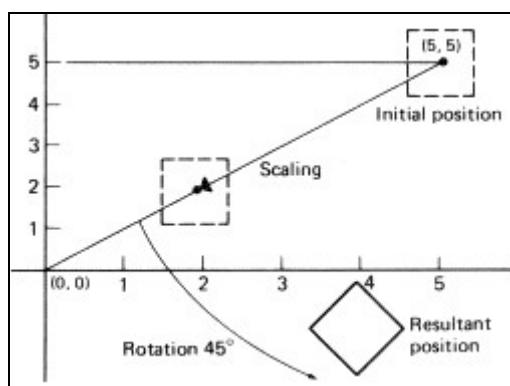


Figura 4.6 – Aplicação de escala e depois rotação.

Considerando a rotação de um objeto em torno de um ponto arbitrário P1. 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 P1, pode-se usar a seguinte sequência de transformações fundamentais:

- Efetuar uma translação, levando P1 à origem.
- Efetuar a Rotação desejada.
- Efetuar uma Translação oposta à realizada em a), levando P1 a posição anterior.

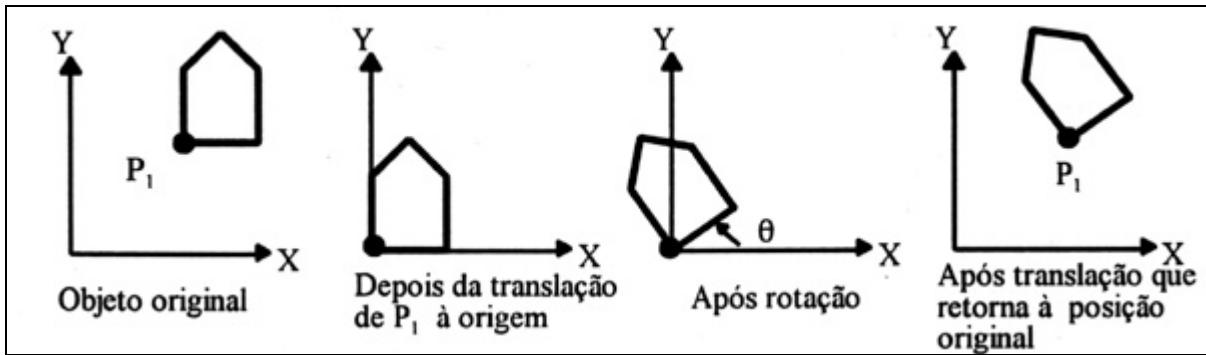


Figura 4.7 – Rotação de um objeto, por um ângulo θ , em relação a um ponto P_1 .

Esta sequência é ilustrada na Figura 4.7, onde o objeto é rotacionado 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:

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

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

4.4 TRANSFORMAÇÕES 2D ADICIONAIS

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

4.4.1 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° 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:

$$E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

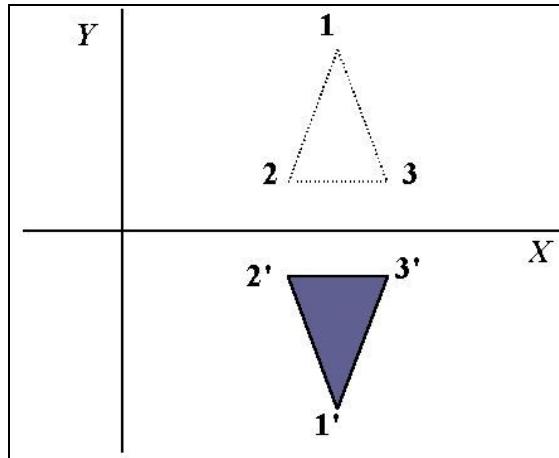


Figura 4.8 – Espelhamento de um objeto em relação ao eixo x.

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

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:

$$E = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

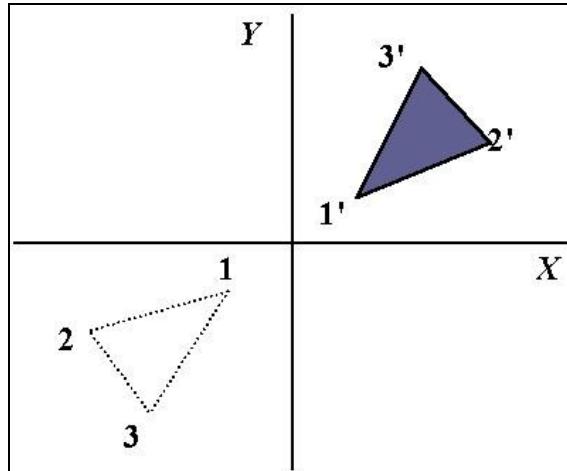


Figura 4.9 – Espelhamento de um objeto em relação ao plano xy

A operação é ilustrada na Figura 4.9. 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° 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° em torno do ponto pivô da reflexão.

4.4.2 CISALHAMENTO

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:

$$SH = \begin{bmatrix} 1 & SH_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Analogamente podemos construir uma matriz para gerar a distorção na direção y. A Figura 4.10 apresenta um exemplo de uma transformação de cisalhamento na direção x.

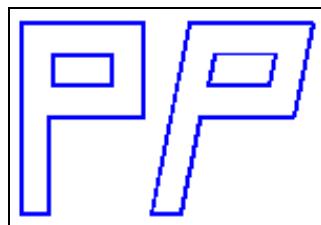


Figura 4.10 – Transformação de cisalhamento na direção x

4.5 TRANSFORMAÇÕES ENTRE SISTEMAS DE COORDENADAS

Já vimos que aplicações gráficas frequentemente 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 (x₀,y₀), com um ângulo de orientação θ 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(-x₀,-y₀)
2. Rotacionar o eixo x' de forma que ele coincida com o eixo x: R(-θ)

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

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

4.7 TRANSFORMAÇÕES 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 3×3 (utilizando coordenadas homogêneas), as transformações geométricas 3D serão representadas por matrizes 4×4 também em coordenadas homogêneas.

Dessa forma, um ponto P de coordenadas (x, y, z) será representado por (x, y, z, W) . Padronizando o ponto teremos, se W for diferente de 0, $(x/W, y/W, z/W, 1)$.

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 poder ser visto na Figura 4.14.

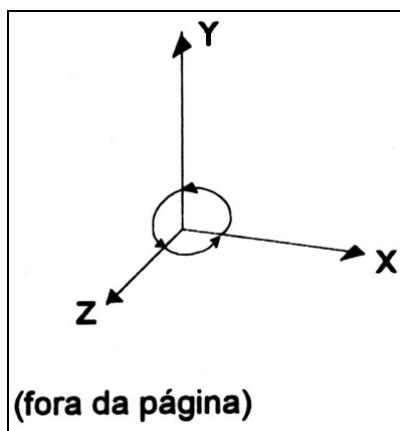


Figura 4.14 - Sistema de coordenadas dado pela regra da mão direita.

O sentido positivo de uma rotação é dado quando, observando-se sobre um eixo positivo em direção à origem, uma rotação de 90° irá levar um eixo positivo em outro positivo. Ou conforme a tabela a seguir:

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

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

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

A transformação acima pode ser representada, também, por:

$$P' = T(dx, dy, dz) \cdot P$$

Similarmente a ESCALA em 3D, fica:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$P' = S(Sx, Sy, Sz) \cdot P$$

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:

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

A matriz de rotação em relação ao eixo x é:

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

A matriz de rotação em relação ao eixo y é:

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

Para a rotação também temos que:

$$P' = R(q) \cdot P$$

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**.

Todas estas matrizes de transformação (T , S e R) possuem inversas. A inversa de T é obtida simplesmente negando-se dx , dy e dz . 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 $B^{-1} = B^T$.

4.8 COMPOSIÇÃO DE TRANSFORMAÇÕES EM 3D

A composição de transformações em 3D pode ser entendida mais facilmente através do exemplo indicado na Figura 4.15. 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.

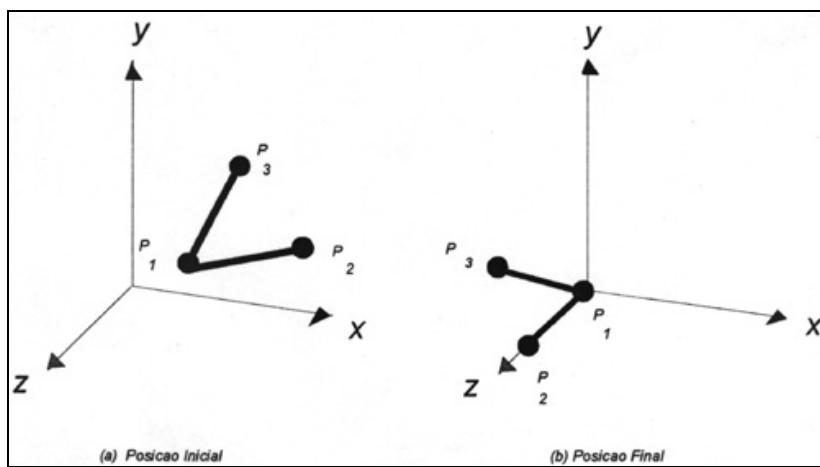


Figura 4.15 - Transformando P_1 , P_2 e P_3 da posição inicial em (a) para a posição final em (b)

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:

- Transladar P_1 para a origem;
- Rotacionar o segmento P_1P_2 em relação ao eixo y , de forma que ele (P_1P_2) fique no plano yz ;
- Rotacionar o segmento P_1P_2 em relação ao eixo x , de forma que ele (P_1P_2) fique sobre o eixo z ;
- 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:

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

Aplicando T a P_1 , P_2 e P_3 , temos:

$$P'_1 = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$P'_2 = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix}$$

$$P'_3 = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix}$$

- **Segundo Passo: Rotacionar em Relação ao eixo Y**

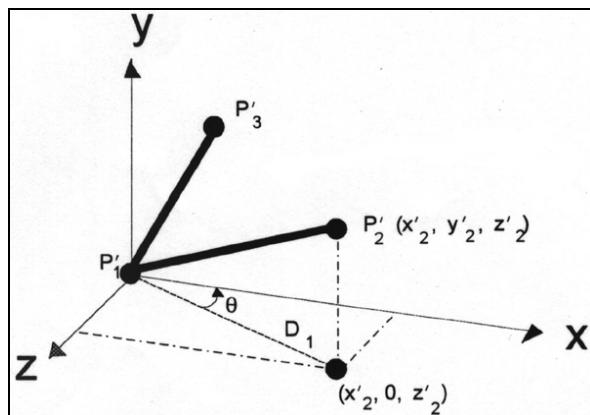


Figura 4.16 - Os pontos após a translação

A Figura 4.16 mostra P_1P_2 após o primeiro passo, bem como a projeção de P_1P_2 no plano x - z (O ângulo θ indica a direção positiva de rotação em relação a y ; o ângulo utilizado na realidade é $-(90-\theta)$). Então:

$$\sin(\theta - 90) = -\cos(\theta) = -\frac{x'_2}{D_1} = \frac{x_2 - x_1}{D_1}$$

$$\cos(\theta - 90) = \sin(\theta) = \frac{z'_2}{D_1} = \frac{z_2 - z_1}{D_1}, \text{ onde:}$$

$$D_1 = \sqrt{(z_2)^2 + (x_2)^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}$$

Substituindo estes valores na matriz de rotação em y temos:

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

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**

A Figura 4.17 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 ϕ , e $\cos \phi = \frac{z''_2}{D_2}$, $\sin \phi = \frac{y''_2}{D_2}$, onde $D_2 = |P''_1P''_2|$ é o **comprimento** do segmento de reta $P''_1P''_2$. Como as translações e rotações preservam o comprimento, o comprimento de $P''_1P''_2$ é o mesmo de P_1P_2 , dessa forma:

$$D_2 = |P_1P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

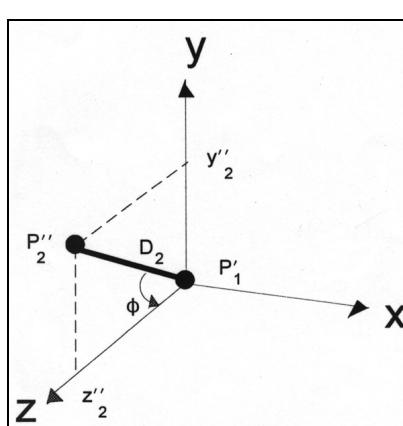


Figura 4.17 - Rotação em relação ao eixo x. P_1 e P_2 de comprimento D_2 é rotacionado em direção ao eixo z, pelo ângulo positivo ϕ .

O resultado da rotação no terceiro passo é:

$$P'''_2 = R_x(\phi) \cdot P''_2 = R_x(\phi) \cdot R_y(\theta - 90) \cdot P'_2 = R_x(\theta) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = \begin{bmatrix} 0 \\ 0 \\ |P_1P_2| \\ 1 \end{bmatrix}$$

Dessa forma, agora P_1P_2 agora está sobre (coincidindo) o eixo z positivo.

- **Quarto Passo: Rotacionar em Relação ao eixo Z**

A Figura 4.18 mostra P_1P_2 e P_1P_3 após o terceiro passo, com P_2''' sobre o eixo z e P_3''' no plano y-z

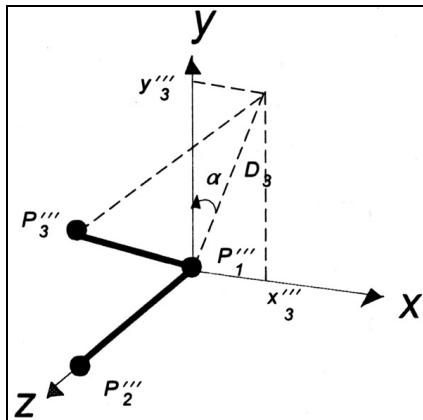


Figura 4.18 - Rotação em relação ao eixo z

$$P_3''' = \begin{bmatrix} x_3''' \\ y_3''' \\ z_3''' \\ 1 \end{bmatrix} = R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) \cdot P_3$$

A rotação dada pelo ângulo positivo α , é

$$\cos \alpha = \frac{y_3'''}{D_3}, \sin \alpha = \frac{x_3'''}{D_3}, D_3 = \sqrt{(x_3''')^2 + (y_3''')^2}$$

trazendo $P_1.P_3'$ para o plano y-z.

Dessa forma, após o quarto passo o resultado é alcançado como visto na Figura 4.15 (b).

A matriz de composição M é a transformação necessária à composição solicitada na Figura 4.15.

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

UNIDADE 5 – VISUALIZAÇÃO EM 3D

5.1 PIPELINE DE VISUALIZAÇÃO

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 câmara fotográfica: imaginamos um observador que, posicionado em um ponto de observação, vê a cena através das lentes de uma câmara virtual que pode ser posicionada de forma a obter a imagem desejada da cena (o “fotógrafo” pode definir a posição da câmara, sua orientação e ponto focal).

A fotografia que se obtém com uma câmara real é uma projeção da cena em um plano de imagem 2D (o filme na câmara). 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 câmara, 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 câmara definem, respectivamente, onde a câmara está e para onde ela está apontando. O vetor que vai da posição da câmara 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 (ver Figura 5.1).

A orientação da câmara é controlada pela sua posição, seu ponto focal e por um vetor denominado *view up*. Estes três parâmetros definem completamente a câmara.

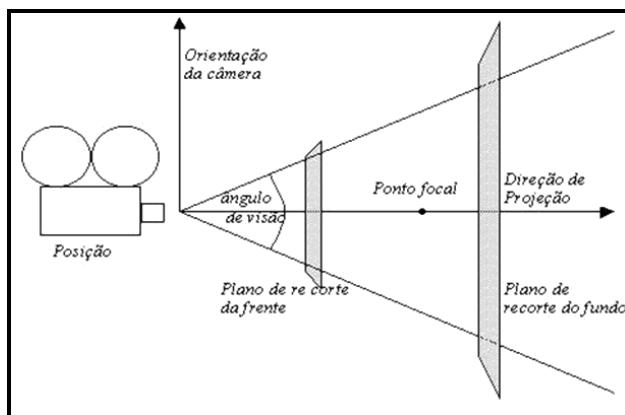


Figura 5.1 – Atributos da câmara

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 câmara 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 câmara.

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 câmara, 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 câmara: a localização dos planos é dada a partir da posição da câmara, 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 câmara 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 5.2 e 5.3.

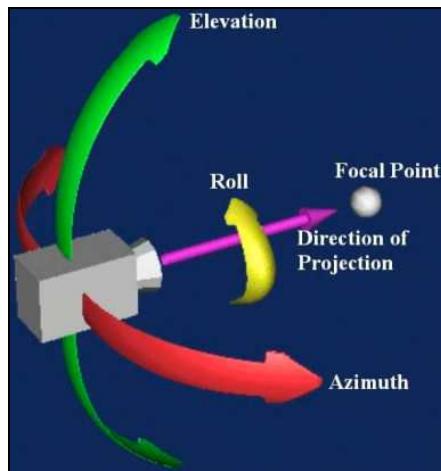


Figura 5.2 - Movimentos de câmara

- **Azimuth** - rotaciona a posição da câmara ao redor do seu vetor *view up*, com centro no ponto focal
- **Elevation** - rotaciona a posição da câmara 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.

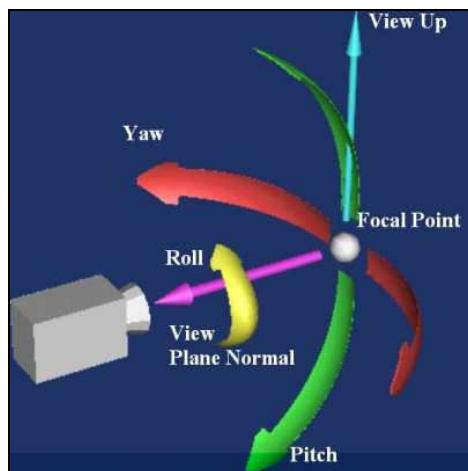


Figura 5.3 – Movimentos de câmara

- **Yaw** - rotaciona o ponto focal da câmara em torno do vetor *view up*, com centro

na posição da câmara.

- **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 câmara.
- **Dolly** - (in, out) move a posição da câmara ao longo da direção de projeção (mais próximo ou mais distante do ponto focal).
- **Zoom** - altera o ângulo de visão da câmara, de modo que uma região maior ou menor da cena fique na região que é potencialmente visível.

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 câmara. A região da cena que aparece no visor da máquina é a que será projetada na superfície do filme. O tipo e o tamanho da lente da câmara determinam quais partes da cena aparecerão na foto final.

5.2 COORDENADAS DE CÂMERA

5.2.1 ESPECIFICAÇÃO DO SISTEMA DE REFERÊNCIA DA CÂMERA

Ao escolhemos uma “vista” específica para uma cena, estabelecemos primeiramente o Sistema de Referência da Câmera (SRC). Um plano de observação, ou plano de projeção, é definido perpendicularmente ao eixo z do SRC (esse plano faz o papel do plano do filme na câmara virtual). Em seguida, é preciso transformar a descrição da cena dada no Sistema de Referência do Universo (SRU) para uma descrição no SRC.

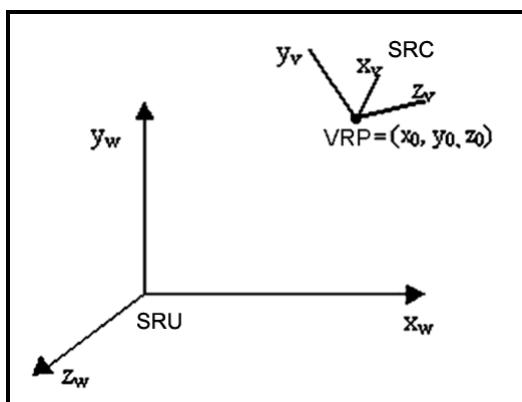


Figura 5.4 – Um SRC baseado na regra da mão direita com x_v , y_v e z_v relativos ao sistema de referência do universo.

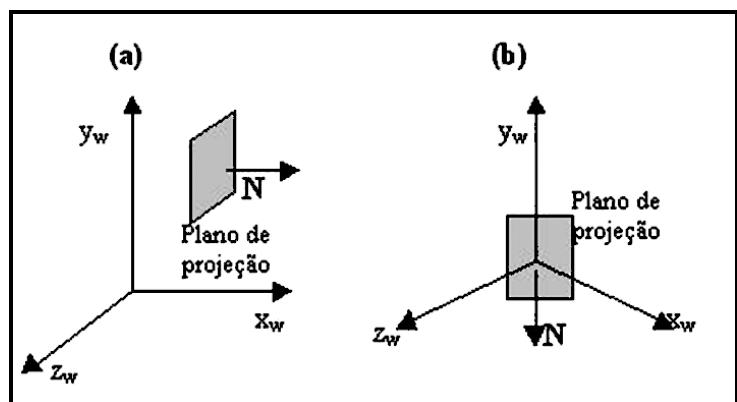


Figura 5.5 – Orientações do plano de observação conforme o vetor normal. O vetor $(1, 0, 0)$ dá a orientação em (a) e $(0, 1, 0)$, a orientação em (b).

Mas, como determinamos o SRC? Inicialmente, estabelecemos a origem do sistema. Para isso, usamos um ponto inicialmente definido no Universo, o *View Reference Point* (VRP). Podemos pensar nesse ponto como sendo a posição da câmara.

Em seguida, selecionamos a direção positiva para o eixo z do SRC (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 SRC (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 SRC (Figuras 5.4 e 5.5).

5.2.2 TRANSFORMAÇÃO DO SISTEMA DE REFERÊNCIA DO UNIVERSO PARA O SISTEMA DE REFERÊNCIA DA CÂMERA

Antes que as descrições do objeto (em SRU) possam ser projetadas no plano de observação, elas precisam ser transformadas para o SRC. A conversão das coordenadas de SRU para SRC é equivalente a uma transformação que superpõe o SRC ao SRU, usando as transformações geométricas já vistas. A sequência de transformações é dada por:

1. translada o VRP para a origem do SRU;
2. aplica rotações para alinhar os eixos do SRC (x_v , y_v , z_v) com os eixos do SRU (x_w , y_w , z_w), respectivamente.

Se as coordenadas do ponto VRP em SRU são dadas por (x_0, y_0, z_0) , a matriz de translação é dada por $T(-x_0, -y_0, -z_0)$. A sequência 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 sequência $R_z.R_y.R_x$. Isto é, primeiro rotacionamos em torno do eixo x_w para trazer o eixo z_v para o plano x_wz_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 SRU para transformá-las para SRC.

Uma forma alternativa de gerar a matriz que descreve as transformações de rotação necessárias é calcular os vetores unitários **u**, **v** e **n** do SRC, e formar a matriz de composição diretamente, uma vez que esses vetores são ortogonais entre si e definem uma matriz ortogonal.

Os vetores unitários **u**, **v** e **n** do SRC (Figura 5.6), podem ser obtidos através dos passos abaixo descritos.

Passo 1:

Obter o vetor **N** e normalizá-lo para obter o vetor **n**. O vetor **n** indica a direção da projeção, estabelecendo o sentido do eixo **z** do Sistema de Referência da Câmera. O vetor **N**, perpendicular ao plano de projeção, pode ser obtido através dos pontos VRP e P.

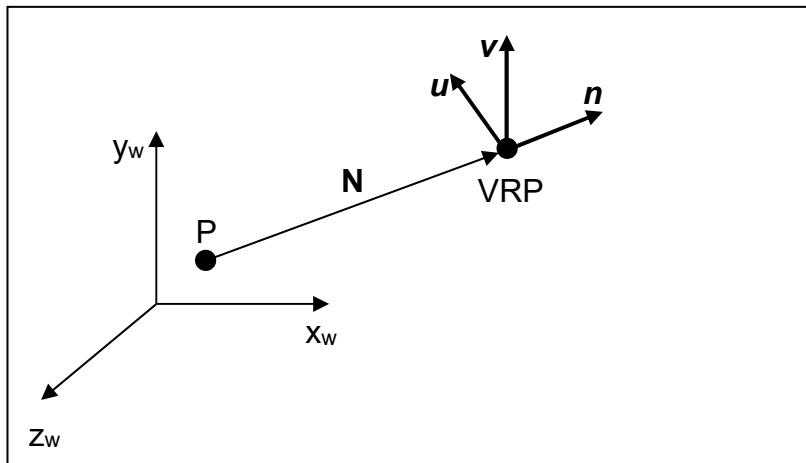


Figura 5.6 – Posicionamento do Sistema de Referência da Câmera

Assim,

$$\vec{N} = VRP - P$$

e

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

Passo 2:

Projetar um vetor **Y** qualquer no plano de projeção para obter o vetor **V**. Normalizando o vetor **V**, obtém-se o vetor unitário **v**. Por conveniência adota-se um vetor **Y** mesmo sentido e direção que o eixo **y** do SRU. Por exemplo, **Y** = (0, 1, 0). A Figura 5.7 auxilia na compreensão deste passo. Caso $|Y| \neq 1$, o vetor **Y** deverá ser normalizado.

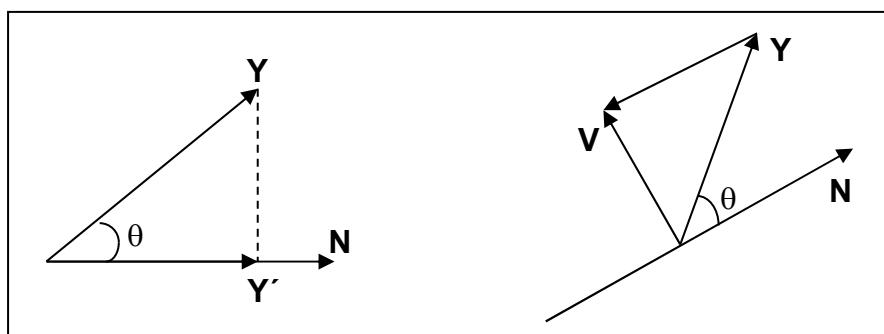


Figura 5.7 – Obtenção do vetor **V**

Algebraicamente, temos:

$$\vec{Y}' = \frac{\vec{N} \cdot \vec{Y}}{|\vec{N}|} \cdot \frac{\vec{N}}{|\vec{N}|} \Rightarrow \vec{Y}' = (\vec{Y} \cdot \hat{n}) \cdot \hat{n}$$

Portanto,

$$\vec{V} = \vec{Y} - (\vec{Y} \cdot \hat{n}) \cdot \hat{n}$$

e

$$\hat{v} = \frac{\vec{V}}{|\vec{V}|} = (v_1, v_2, v_3)$$

Passo 3:

Obter o vetor \mathbf{u} pela normalização do vetor \mathbf{U} . Como \mathbf{V} e \mathbf{N} são vetores ortogonais entre si, o produto vetorial de \mathbf{V} e \mathbf{N} definirá o vetor \mathbf{U} . Assim,

$$\vec{U} = \vec{V} \times \vec{N}$$

$$\hat{u} = \frac{\vec{U}}{|\vec{U}|} = (u_1, u_2, u_3) \quad \text{ou} \quad \hat{u} = \hat{v} \times \hat{n}$$

Portanto, a matriz de rotação composta para a transformação de visualização é dada por:

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

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:

$$M_{SRU,SRC} = R \cdot T = \begin{bmatrix} u_1 & u_2 & u_3 & -VRP \cdot \hat{u} \\ v_1 & v_2 & v_3 & -VRP \cdot \hat{v} \\ n_1 & n_2 & n_3 & -VRP \cdot \hat{n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.3 PROJEÇÕES

Tendo os objetos tridimensionais descritos no sistema de coordenadas de observação pode-se projetá-los 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 5.8 apresenta uma taxonomia dos métodos de projeção.

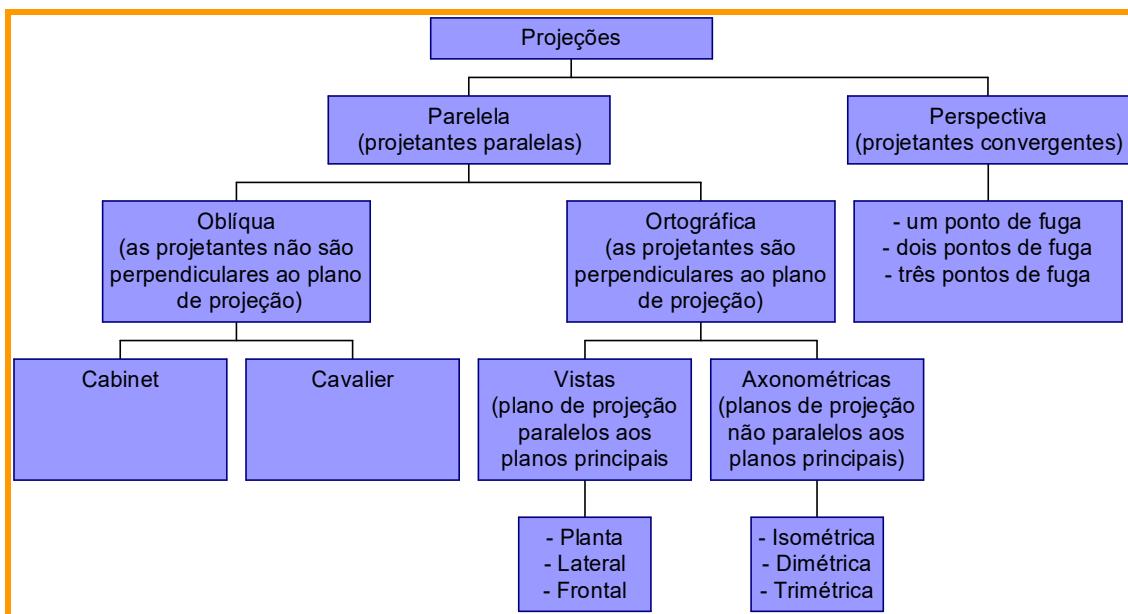


Figura 5.8 – Taxonomia de projeções

5.3.1 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* ou *projetores*. (Figura 5.9)

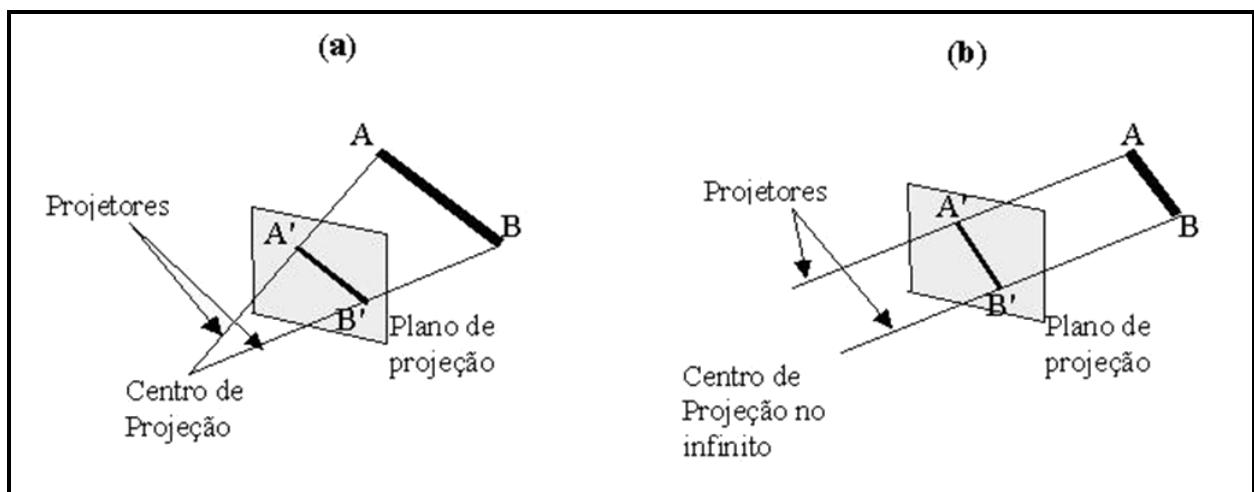


Figura 5.9 - Linha **AB** e sua projeção **A'B'**: (a) perspectiva; (b) ortogonal

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 se dá a aparência de haver uma intersecção entre um conjunto de retas paralelas com um dos eixos principais Ox , Oy ou Oz . 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 os 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 5.10 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!

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 (Figura 5.11). 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 (Figura 5.12).

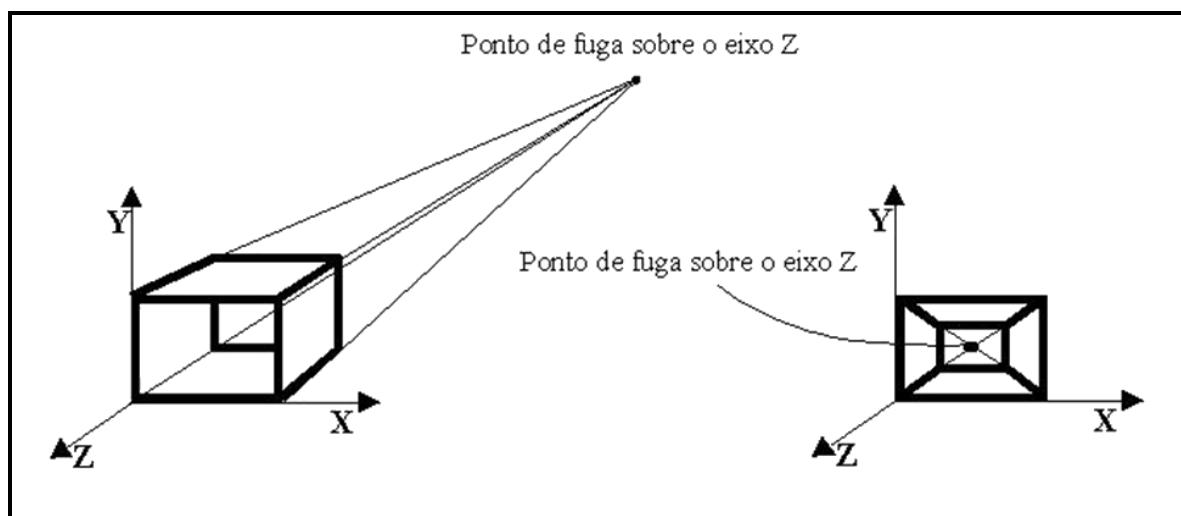


Figura 5.10 – Projeções de um cubo (com 1 ponto de fuga) sobre um plano cortando o eixo Z , apresentando o ponto de fuga.

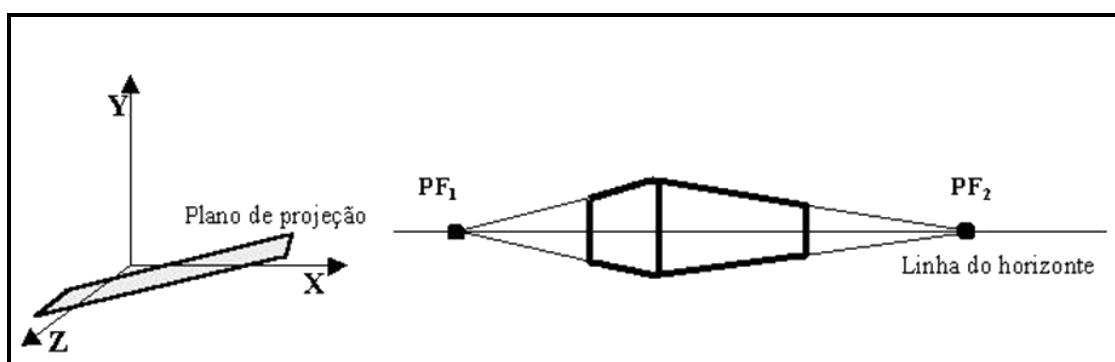


Figura 5.11 – Projeções perspectivas com 2 pontos de fuga (o plano de projeção intercepta 2 eixos (x e z)).

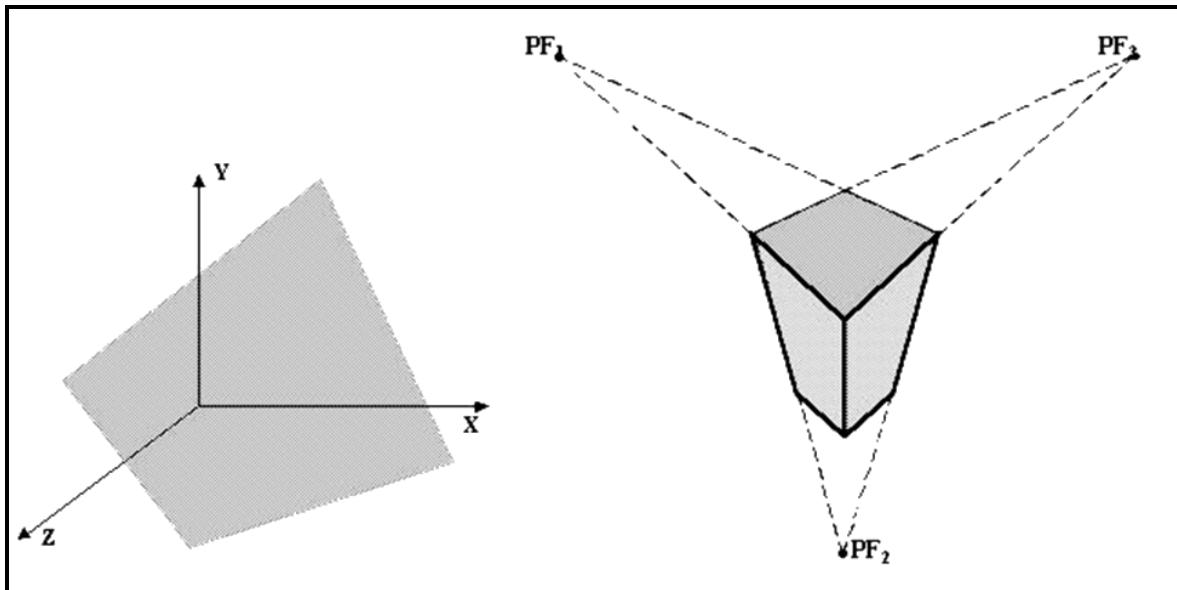


Figura 5.12 – Projeções perspectivas com 3 pontos de fuga
(o plano de projeção intercepta os 3 eixos).

5.3.2 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 objeto.

- 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 5.13);
- Pontos de Fuga:** As projeções de retas não paralelas ao plano de projeção, provocam a ilusão de que se interceptam num ponto do horizonte;
- Confusão Visual:** Os objetos situados atrás do centro de projeção são projetados no plano de projeção de cima para baixo e de trás para frente (Figura 5.14);
- 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. 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.

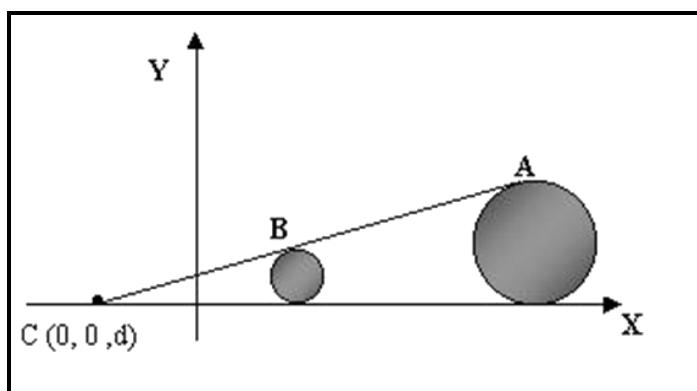


Figura 5.13 – A esfera A é bem maior que a esfera B, porém ambas aparecem com o mesmo tamanho quando projetadas no plano de visão.

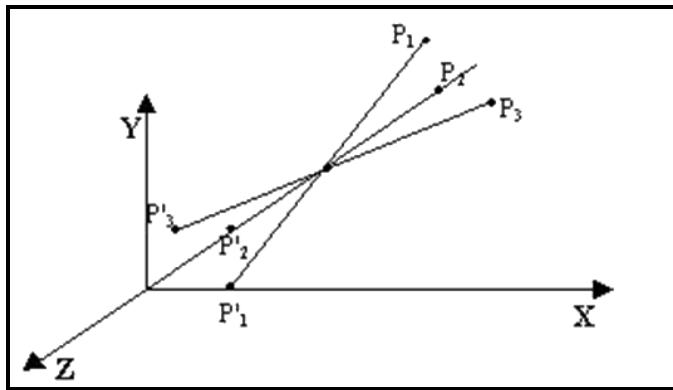


Figura 5.14 – Confusão visual da perspectiva (objeto atrás do centro de projeção)

5.4 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_{pp} , um ponto no eixo Z_v , e que o plano de projeção, normal ao eixo z , está posicionado em z_{vp} , como mostra a Figura 5.15. 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_p, y_p, z_p) de qualquer ponto ao longo da linha de projeção perspectiva como:

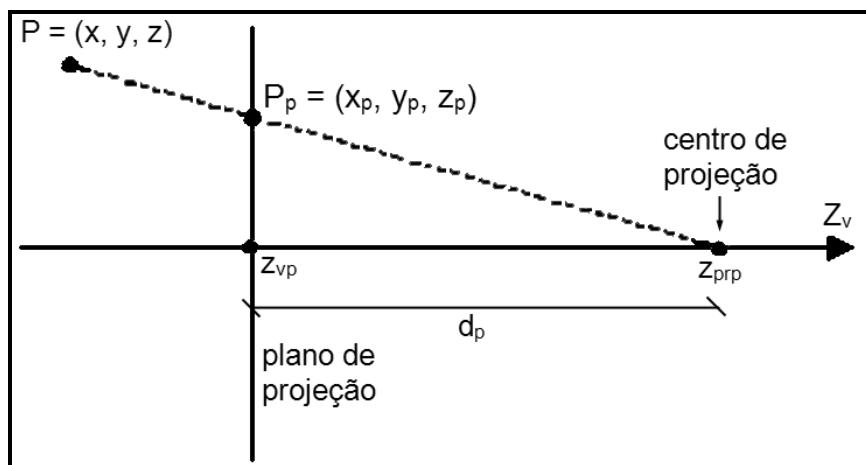


Figura 5.15 – Projeção em perspectiva do ponto $P = (x, y, z)$ na posição (x_p, y_p, z_p) sobre o plano de projeção

$$x_p = x - xu$$

$$y_p = y - yu$$

$$z_p = z - (z - z_{pp}) \cdot u$$

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 $(0, 0, z_{pp})$. As projeções perspectiva e paralela também podem ser definidas através de matrizes 4×4 , assim como as transformações geométricas básicas, o que é interessante para a composição de transformações juntamente com a projeção. No plano de projeção, sabemos que $z_p = z_{vp}$, e podemos resolver a equação de z_p para

obter o valor do parâmetro u nessa posição ao longo da linha de projeção:

$$u = \frac{z_{vp} - z}{z_{prp} - z}$$

Substituindo esse valor de u nas equações de x_p e y_p , obtemos as equações de transformação perspectiva:

$$\begin{aligned} x_p &= x \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) = x \left(\frac{d_p}{z_{prp} - z} \right) \\ y_p &= y \left(\frac{z_{prp} - z_{vp}}{z_{prp} - z} \right) = y \left(\frac{d_p}{z_{prp} - z} \right) \end{aligned}$$

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

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

Nesta representação o fator homogêneo é

$$h = \frac{z_{prp} - z}{d_p}$$

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

$$x_p = \frac{x_h}{h}, \quad y_p = \frac{y_h}{h}$$

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 posicionado 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 , isto é, $z_{vp} = 0$, 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, isto é, nesse

caso $z_{pp} = 0$.

5.5 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 (Figura 5.16):

$$x_p = x, \quad y_p = y$$

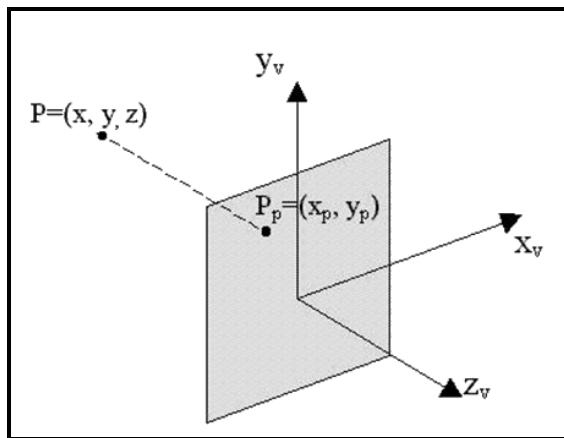


Figura 5.16 – Projeção ortogonal de um ponto no plano de projeção

A matriz de transformação para coordenadas ortográficas é dada por:

$$M_{\text{ortograf}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.6 PROJEÇÃO PERSPECTIVA UTILIZANDO COORDENADAS ESFÉRICAS

O monitor de computador é um objeto bidimensional. Apesar disto, podemos criar a impressão visual de uma terceira dimensão da mesma forma como um artista pode criar a impressão visual de uma cena tridimensional em uma tela bidimensional.

O sistema de coordenadas retangulares (ou Cartesiano) requer três eixos mutuamente perpendiculares. Os eixos X e Y utilizados nos sistemas bidimensionais são agora acrescentados de um eixo Z que passa através da origem e é perpendicular ao plano XY.

Outra forma de representar o espaço é utilizar o *sistema de coordenadas*

esféricas. Neste sistema, cada ponto P é representado por uma tripla ordenada (α, β, χ) de números. A distância do ponto à origem do sistema é α . O ângulo entre o segmento OP e a direção positiva do eixo Z é β , e o ângulo entre a direção positiva do eixo X e a projeção do segmento OP sobre o plano XY é χ e é medido na direção anti-horária, como visto de um ponto sobre o eixo Z positivo (Figura 5.17).

Já que um ponto pode ser identificado em termos de coordenadas cartesianas (X, Y, Z) ou em termos de coordenadas esféricas (α, β, χ) , é necessário que haja algum meio de conversão entre elas. Para isto nós usaremos as seguintes fórmulas, baseadas no triângulo retângulo da trigonometria:

$$X = \alpha \sin \chi \cos \beta$$

$$Y = \alpha \sin \chi \sin \beta$$

$$Z = \alpha \cos \chi$$

$$\alpha = \sqrt{X^2 + Y^2 + Z^2}$$

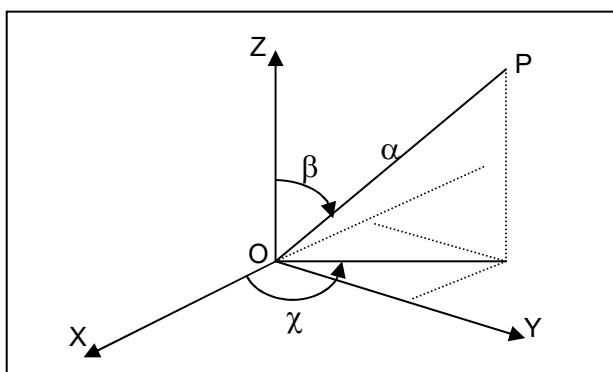


Figura 5.17 – Representação de um ponto P em coordenadas esféricas

Para obtermos uma imagem bidimensional de um objeto tridimensional, é necessário associar um par de coordenadas bidimensionais (ou coordenadas de tela) (SX, SY) com cada tripla de coordenadas retangulares (X, Y, Z) ou esféricas (α, β, χ) . O processo utilizado para se obter as coordenadas de tela é apresentado na Figura 5.18.

A posição do observador é representada pelo ponto P. O plano de projeção (tela) é o plano sobre o qual o objeto está projetado. Assume-se que este plano de projeção é perpendicular ao segmento OP, e a uma distância fixa D do ponto P. Como os pontos do objeto são projetados sobre o plano de projeção, nós podemos assim obter as coordenadas (SX, SY) correspondentes a cada ponto (X, Y, Z) .

Adotaremos a prática de identificar o ponto de vista (localização do olho do observador) em termos de coordenadas esféricas, pois este sistema de coordenadas permite um controle fácil sobre a localização do ponto de vista.

Assim:

- α representará a distância do ponto de vista à origem $(0, 0, 0)$. A sua variação moverá o ponto de vista para mais perto ou para mais longe da origem. Isto fará com que o objeto pareça maior ou menor, de acordo com o valor de α , afetando com isto a perspectiva do objeto;
- β e χ identificam a direção na qual o observador verá o objeto.

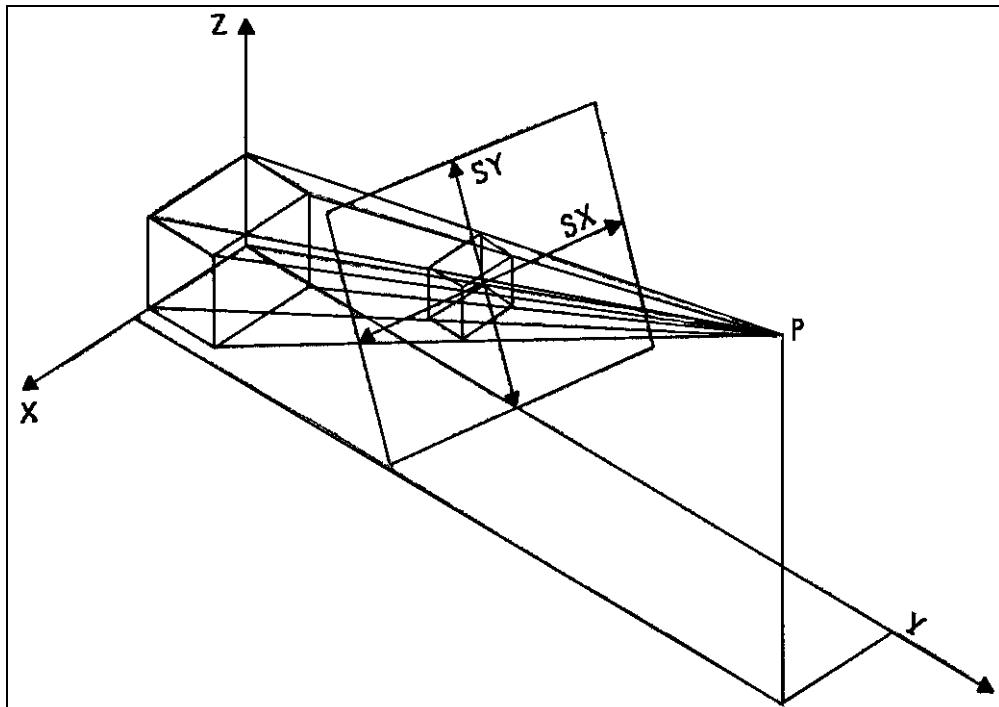


Figura 5.18 – Projeção de um objeto tridimensional em um plano bidimensional

A Figura 5.18 sugere que o plano de projeção deve estar localizado entre o observador e o objeto, mas não é necessário que isto aconteça sempre. O plano de projeção também pode estar localizado atrás do objeto. Isto resulta em uma imagem projetada que é maior que a imagem original.

Os parâmetros α , β , γ e D controlam a localização e a direção da visualização e o tamanho da imagem. Vejamos agora como obter um método para projetar cada ponto de um objeto tridimensional sobre um plano.

Observe a Figura 5.18. A localização do olho do observador (ponto P) está identificada e tem coordenada retangular (X , Y , Z) e coordenada esférica (α , β , γ). Nós posicionamos o plano de projeção (tela) a uma distância D do olho do observador, de tal forma que ela é perpendicular à linha que conecta o observador com a origem. Imagine agora um segundo sistema de eixos retangulares localizado no ponto P. Este é o sistema de coordenadas do observador, e suas coordenadas são denotadas por (X_e , Y_e , Z_e). A Figura 5.19 mostra o sistema de coordenadas do observador.

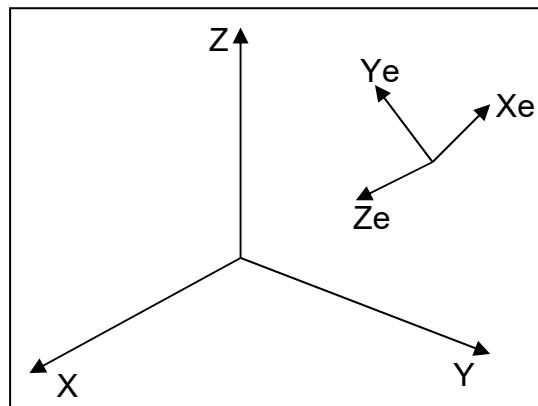


Figura 5.19 – O sistema de coordenadas do observador

Podemos dividir a tarefa de obter uma projeção em perspectiva em duas partes:

1. A identificação das coordenadas do observador (X_e , Y_e , Z_e) associadas com um ponto que tem coordenadas padrão (X , Y , Z);
2. A identificação das coordenadas da tela (SX , SY) associadas com um conjunto específico de coordenadas do observador (X_e , Y_e , Z_e).

A segunda fase é a mais fácil das duas. Com as coordenadas do observador no lugar, nós posicionamos a tela a uma distância D da origem deste sistema (Figura 5.20).

O ponto Q é dado em coordenadas do observador (X_e , Y_e , Z_e). Para obter as coordenadas de tela (SX , SY) devemos utilizar a semelhança de triângulos. Os triângulos retângulos PBA e PGC (Figura 5.21) são triângulos semelhantes, com:

$$\frac{GC}{PG} = \frac{BA}{PB}$$

Assim,

$$\frac{SY}{D} = \frac{Ye}{Ze}$$

Então, temos que:

$$SY = D \cdot \frac{Ye}{Ze}$$

Do mesmo modo, dos triângulos semelhantes PBF e PGE , nós temos:

$$\frac{GE}{PG} = \frac{BF}{PB} \quad e \quad \frac{SX}{D} = \frac{Xe}{Ze}$$

Assim,

$$SX = D \cdot \frac{Xe}{Ze}$$

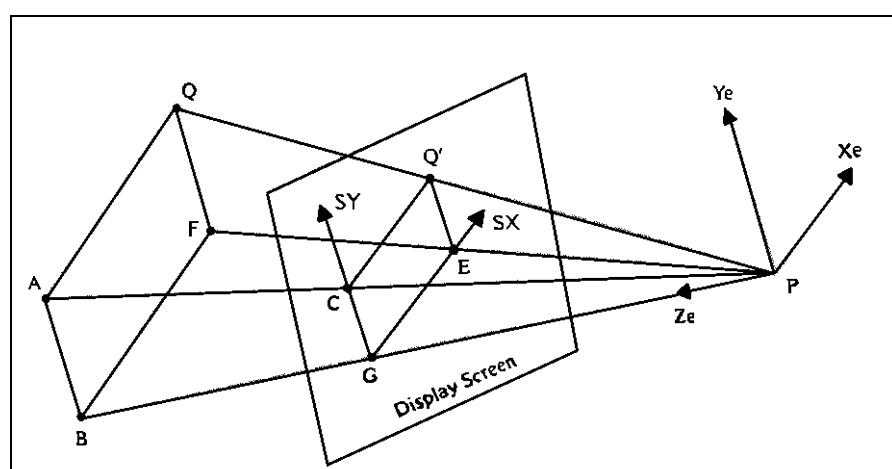


Figura 5.20 – Projeção perspectiva com o posicionamento do ponto de vista do observador

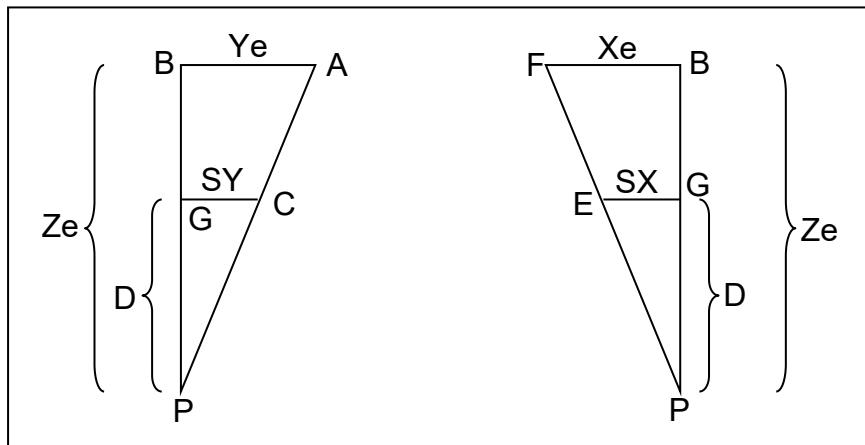


Figura 5.21 – Semelhança dos triângulos PBA/PGC e PBF/PGE

Feito isso, passemos agora a primeira etapa: estabelecer a associação entre as coordenadas padrão e as coordenadas do observador.

Isto é feito através de uma sequência de quatro transformações, cada uma das quais irá transformar parcialmente o sistema de eixos (X, Y, Z) para o sistema (X_e, Y_e, Z_e).

O efeito das transformações pode ser visto como o estabelecimento de uma sequência de novos sistemas de coordenadas, tendo como resultado final o sistema de coordenadas do observador.

Para a finalidade de executar estas transformações, o ponto P, localizado no olho do observador será identificado pelas coordenadas retangulares (H, K, L) ou pelas coordenadas esféricas (α , β , χ), medidas com relação ao sistema de eixos de coordenadas padrão. Lembre que:

$$H = \alpha \sin \chi \cos \beta$$

$$K = \alpha \sin \chi \sin \beta$$

$$L = \alpha \cos \chi$$

A seguir apresentamos a sequência de passos apresentados por Myers:

- Passo 1: Estabelecer a origem em $(H, K, L) = (\alpha \sin \chi \cos \beta, \alpha \sin \chi \sin \beta, \alpha \cos \chi)$. A matriz de transformação que tem este efeito é:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -H & -K & -L & 1 \end{bmatrix}$$

Apesar de parecer inconveniente, a representação da origem em coordenadas esféricas simplifica muitos os cálculos.

- Passo 2: Rotacionar o sistema de eixos sobre o eixo Z', em 90° no sentido anti-horário. O resultado desta operação faz com que o eixo Y' negativo intercepte o eixo Z. A matriz de transformação desta operação é:

$$B = \begin{bmatrix} \sin \beta & \cos \beta & 0 & 0 \\ -\cos \beta & \sin \beta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Passo 3: Rotacionar o sistema de eixos sobre o eixo X' de um ângulo de $(180^\circ - \chi)$. Como resultado, o eixo Z' deve ser direcionado através da origem do sistema de eixos X, Y, Z. Sua matriz de transformação é:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -\cos \chi & -\sin \chi & 0 \\ 0 & \sin \chi & -\cos \chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Passo 4: Mudar para um sistema de coordenadas orientado pela regra da mão esquerda, invertendo o eixo X' utilizando a matriz:

$$D = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se as coordenadas padrão (X, Y, Z) de um ponto são conhecidas, as coordenadas do observador (Xe, Ye, Ze) podem ser obtidas através da aplicação de uma única matriz de transformação T, composta das matrizes de transformação vistas anteriormente, da seguinte forma:

$$T = A \cdot B \cdot C \cdot D$$

Assim, aplicando a matriz de transformação as coordenadas homogêneas (X, Y, Z, 1), temos:

$$(Xe, Ye, Ze, 1) = (X, Y, Z, 1) \cdot \begin{bmatrix} -\sin \beta & -\cos \beta \cos \chi & -\cos \beta \sin \chi & 0 \\ \cos \beta & -\sin \beta \cos \chi & -\sin \beta \sin \chi & 0 \\ 0 & \sin \chi & -\cos \chi & 0 \\ 0 & 0 & \alpha & 1 \end{bmatrix}$$

de onde podemos extrair as seguintes equações de visualização e projeção em perspectiva:

$$Xe = -X \sin \beta + Y \cos \beta$$

$$Ye = -X \cos \beta \cos \chi - Y \sin \beta \cos \chi + Z \sin \chi$$

$$Ze = -X \sin \chi \cos \beta - Y \sin \beta \sin \chi - Z \cos \chi + \alpha$$

e:

$$SX = D \cdot \frac{Xe}{Ze}$$

$$SY = D \cdot \frac{Ye}{Ze}$$

5.6.1 PONTO DE VISTA E PERSPECTIVA

Como vimos anteriormente, os parâmetros α , β , χ e D afetam a imagem que é criada na tela.

Alterando os valores de β e χ podemos visualizar o objeto de diferentes direções.

Alterando o valor de α (distância do observador à origem) aproximamos ou afastamos o observador do objeto. Esta mudança se refletirá na mudança de tamanho do objeto observado.

Além disso, há ainda outra possibilidade de controlar o tamanho da imagem na tela, alterando o valor de D . Quando um ponto de vista é especificado, as mudanças nos valores de D irão afetar o tamanho da imagem sobre o plano de projeção.

5.7 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 ajuste ao seu programa de aplicação (angstrons, 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 5.22.

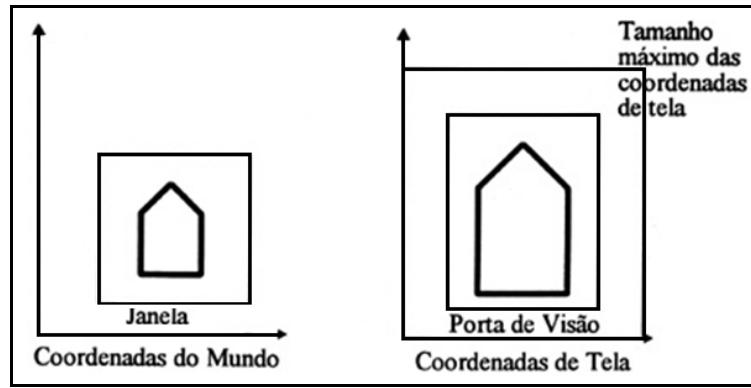


Figura 5.22 - Janela em coordenadas do mundo e porta de visão em coordenadas de tela.

Como pode ser visto pela Figura 5.22, 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, então as novas primitivas de saída desenhadas sobre a tela precisam ser também atualizadas.

Outro ponto é que se podem ter múltiplas portas de visão de uma mesma janela, o que é mostrado na Figura 5.23.

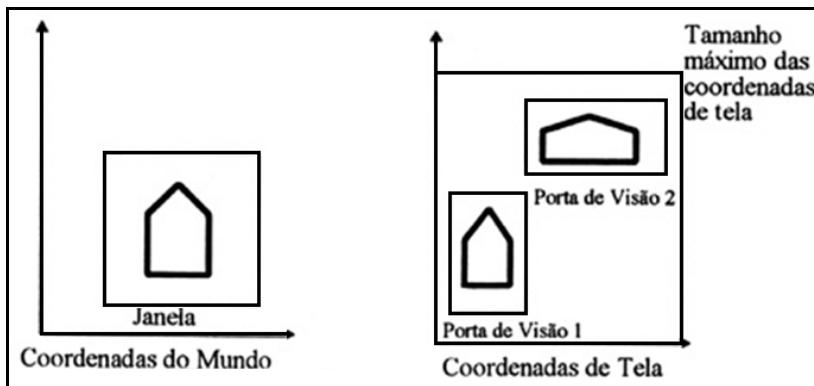


Figura 5.23 - Duas portas de visão associadas a uma mesma janela

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 5.24.

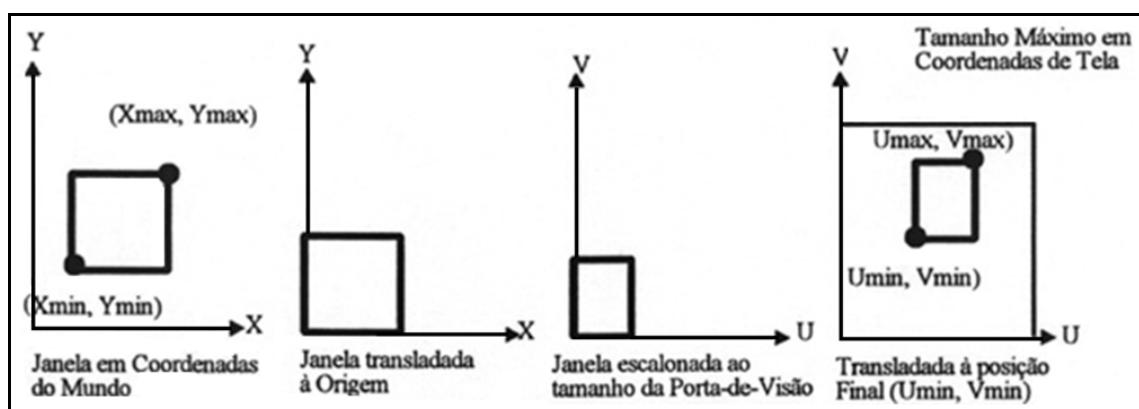


Figura 5.24 - Os passos da transformação janela - porta de visão

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:

$$M_{jp} = T(u_{\min}, v_{\min}, 0) \cdot S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}, 1\right) \cdot T(-x_{\min}, -y_{\min}, 0)$$

$$M_{jp} = \begin{bmatrix} 1 & 0 & 0 & u_{\min} \\ 0 & 1 & 0 & v_{\min} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -x_{\min} \\ 0 & 1 & 0 & -y_{\min} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_{jp} = \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 & -x_{\min} \cdot \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 & -y_{\min} \cdot \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.7.1 EIXO VERTICAL DA VIEWPORT EM SENTIDO OPOSTO A WINDOW

O sistema de coordenadas associado ao monitor do computador (SRT) tem seu eixo vertical orientado em sentido oposto aquele que normalmente usamos no *window* (SRU). Geralmente a *viewport* corresponde a uma porção do monitor e, por compartilhar o mesmo sistema de coordenadas, também tem seu eixo v em sentido oposto ao SRU, conforme ilustrado na Figura 5.25.

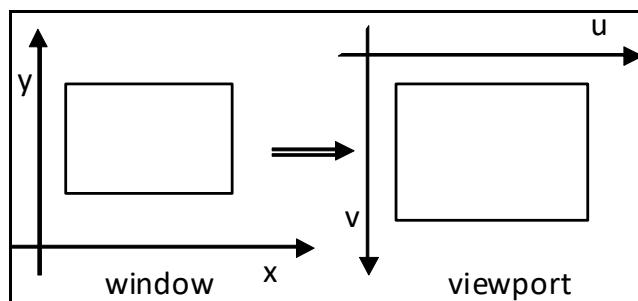


Figura 5.25 – *Window* e *viewport* com eixos verticais em sentidos opostos

Neste caso a transformação janela–porta de visão (*window-to-viewport*) deverá, em algum momento espelhar o objeto em relação ao eixo horizontal (O_x). A Figura 5.26 mostra a sequência de TGs necessária para realizar a transformação.

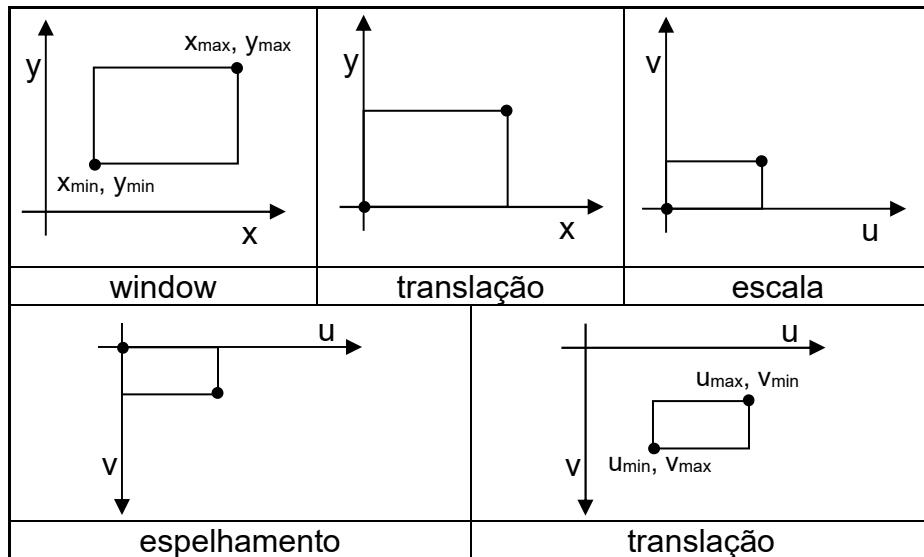


Figura 5.26 – Sequência de TGs envolvidas na transformação janela–porta de visão quando os eixos verticais da *window* e da *viewport* estão em sentidos opostos

Concatenando a quatro TGs temos:

$$M_{jp} = T(u_{\min}, v_{\max}, 0) \cdot E(\overrightarrow{Ox}) \cdot S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}, 1\right) \cdot T(-x_{\min}, -y_{\min}, 0)$$

$$M_{jp} = \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 & -x_{\min} \cdot \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\min} - v_{\max}}{y_{\max} - y_{\min}} & 0 & y_{\min} \cdot \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\max} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.8 CONCATENAÇÃO DAS MATRIZES DO PIPELINE

Como visto na Unidade 4, a concatenação de matrizes de transformações geométricas possibilita reduzir o esforço computacional quando processamos os vértices dos objetos.

O *pipeline* de visualização nada mais é do que uma sequência de transformações geométricas que, agora, estão representadas sobre a forma de matrizes 4×4 e podem ser concatenadas. Assim, a matriz que converte um objeto diretamente do SRU para o SRT passa a ser dada por:

$$M_{SRU,SRT} = M_{jp} \cdot M_{proj} \cdot M_{SRU,SRC}$$

Para transformar os vértices do objeto do SRU para o SRC, fazemos:

$$P' = M_{SRU,SRT} \cdot P_{SRU} \Rightarrow \begin{bmatrix} x_h \\ y_h \\ z' \\ h \end{bmatrix} = M_{SRU,SRT} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Caso o fator homogêneo h seja diferente de 1, devemos fazer o ajuste das coordenadas homogêneas, dividindo as coordenadas x e y por h , como representado abaixo. A coordenada z' não é normalizada, pois está associada à profundidade do objeto em cena e será necessária para determinação da visibilidade.

$$x_{SRT} = \frac{x_h}{h}, y_{SRT} = \frac{y_h}{h}$$

5.9 OCULTAÇÃO DE LINHAS E SUPERFÍCIES

A solução eficiente de problemas de visibilidade é provavelmente o principal passo do processo de obtenção de imagens realistas. Estes problemas lidam frequentemente com a determinação de visibilidade de linhas e superfícies.

Para exibir objetos complexos é necessário descrevê-los matematicamente. Comumente são empregadas duas soluções: uma primeira opção é a representação por um subconjunto de curvas, extraído dos contornos do objeto, de maneira a representá-lo. A descrição matemática destas curvas é normalmente usada para gerar a exibição. Uma segunda solução propõe a representação do sólido através de uma série de faces conectadas apropriadamente. Uma aproximação matemática simples pode ser utilizada para descrever cada face, e a exibição do objeto é produzida como o agregado destas faces.

Existem diversas maneiras de se resolver o problema de superfícies ocultas. Algumas são simples, mas requerem grande quantidade de memória. Outras usam menos memória, mas demandam maior tempo de computação. Outros métodos são rápidos, demandam pouca memória e são simples, porém sua utilidade é limitada a alguns casos especiais. O avanço tecnológico na confecção de chips de memória disponibilizou o armazenamento de quantidades enormes de informação a baixo custo. Assim, as técnicas de determinação de superfícies ocultas baseadas no uso intensivo de memória estão ganhando popularidade.

5.9.1 OCULTAÇÃO DE LINHAS

Uma das aplicações mais comuns da computação gráfica é o desenho de funções contínuas a duas variáveis, como $\mathbf{y} = f(\mathbf{x}, \mathbf{z})$. Funções como estas definem superfícies como as mostradas na Figura 5.27(a). Estas superfícies são um caso especial de problemas de ocultação de superfícies, para os quais existem soluções específicas e eficazes.

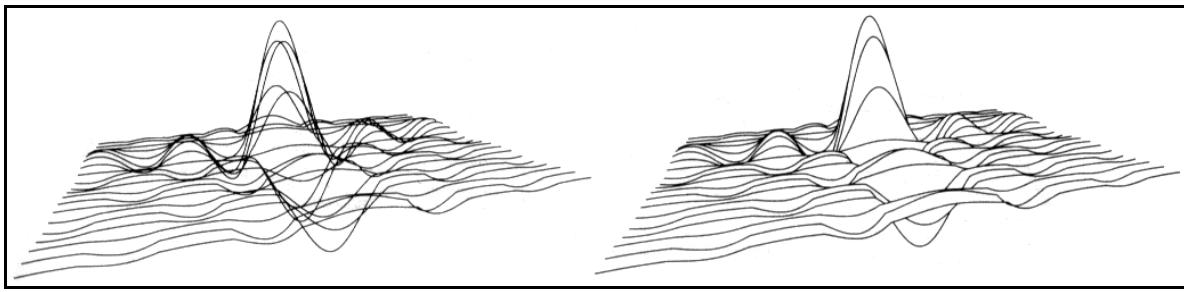


Figura 5.27 – Gráfico de uma função a duas variáveis
(a) sem e (b) com linhas ocultas removidas

Uma função de x e z pode ser aproximada por uma matriz $m \times n$ de valores y . Cada elemento desta matriz pode ser entendido como sendo a altura de um ponto (x, z) de um reticulado que é a base do gráfico. Assumimos que cada coluna do reticulado corresponde a um valor constante de x e que cada linha corresponde a um valor constante de z . Em outras palavras, a matriz está alinhada com os eixos x e z do sistema de coordenadas. Assim, os índices e o valor de um elemento da matriz correspondem a um ponto do espaço 3D. Um desenho em **wireframe** da superfície pode ser feito desenhando-se dois conjuntos: um conjunto de *polylines* usando os pontos de cada linha da matriz (z constante) e o outro conjunto de *polylines*, ortogonal ao primeiro, usando os pontos de cada coluna da matriz (x constante). Um algoritmo de ocultação de linhas deve suprimir todas as partes da superfície que estão atrás de outras partes, como mostrado na Figura 5.27(b). Tanto a Figura 5.27(a) como a 5.27(b) mostram apenas o conjunto de *polylines* desenhado com as linhas da matriz.

Para implementar esta ideia consideraremos inicialmente o problema de desenhar somente as *polylines* com z constantes, assumindo que a *polyline* mais próxima do observador está numa aresta da superfície. Uma solução eficiente para este problema está baseada no fato de que cada *polyline* com z constante aparenta estar num plano paralelo e separado de coordenada z . Uma vez que nenhum destes planos (portanto nenhuma das *polylines*) intercepta outro, cada *polyline* não pode ser ocultada por outra *polyline* de coordenada z mais afastada do observador. Então, desenharemos as *polylines* de coordenada z constante em distâncias crescentes a partir do observador, estabelecendo uma ordem de “frente-para-fundo”. O desenho correto de cada *polyline* de coordenada z constante requer, apenas, que não desenhemos as partes da *polyline* que são ocultadas por partes da superfície desenhadas anteriormente.

Consideremos apenas o limite da silhueta das *polylines* desenhadas no plano de visão, mostrada na Figura 5.28. Quando uma nova *polyline* é desenhada, ela será visível somente onde sua projeção eleva-se acima ou abaixa da silhueta anterior. Uma vez que cada nova *polyline* tem uma coordenada z constante maior do que qualquer uma das *polylines* precedentes, ela não pode atravessar qualquer parte da superfície já desenhada. Então, para determinar quais partes serão desenhadas, precisamos apenas comparar os valores y da *polyline* que está sendo projetada com aqueles correspondentes às partes da silhueta da superfície já desenhada. O algoritmo que utiliza uma informação codificada representando o mínimo e o máximo da coordenada y para cada x da silhueta é chamado de **algoritmo da linha horizontal**.

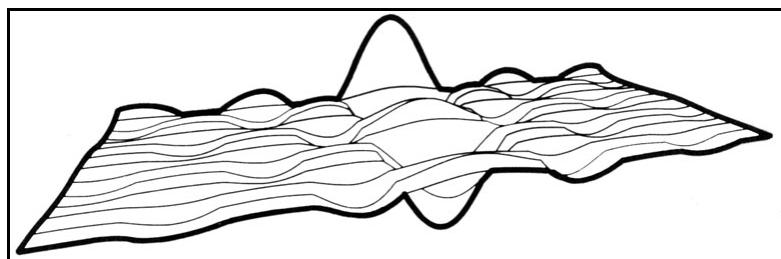


Figura 5.28 – Silhueta das linhas desenhadas

Uma forma de representar esta silhueta utiliza dois vetores (**YMIN**, **YMAX**), para armazenar o menor e o maior valor **y**, na projeção, para um conjunto finito de coordenadas **x**. Estas estruturas de dados são conhecidas como imagem-precisão, pois elas possuem número finito de entradas. **YMIN** e **YMAX** são inicializados, respectivamente, com valores para **y** que estão acima e abaixo de todos os valores de **y** possíveis de serem obtidos na projeção da superfície. Quando uma nova polyline é desenhada, os valores projetados de **y**, para cada par de vértices adjacentes, são comparados com os valores na posição correspondente nos vetores da silhueta. Como mostra a Figura 5.29, um vértice cujo valor está acima daquele na posição correspondente de **YMAX** (A, B, G) ou abaixo de **YMIN** (E, F) está visível; por outro lado, o segmento (C, D) é invisível. Se ambos os vértices são invisíveis, então o segmento de linha é completamente invisível (CD) e os vetores da silhueta permanecem inalterados. Se ambos os vértices são visíveis, com respeito ao mesmo vetor da silhueta (AB, EF), então o segmento de linha é inteiramente visível e pode ser desenhado inteiramente, e este vetor da silhueta deve ser atualizado. As coordenadas **x** de dois vértices adjacentes numa polyline de coordenada **z** constante são, geralmente, mapeados para locais não adjacentes nos vetores de siluetas. Nesta situação, os valores de **y** devem ser interpolados linearmente entre dois **y** adjacentes dos vetores **Y**.

Finalmente, devemos considerar o caso de linhas parcialmente visíveis, nos quais ambos os vértices são não visíveis quando considerado um mesmo vetor da silhueta. Ainda que isto signifique que um dos vértices é visível e o outro invisível (BC, DE), pode acontecer de ambos os vértices serem visíveis, um acima de **YMAX** e outro abaixo de **YMIN** (FG). Os valores de **y** interpolados podem ser comparados com aqueles nas regiões de divisão nos vetores da silhueta para determinar os pontos de interseção. A linha pode não ser visível naqueles locais onde um valor de **y** interpolado adentra a silhueta. Somente as partes visíveis do segmento de linha fora da silhueta devem ser desenhadas, e o vetor da silhueta deve ser atualizado, como mostra a Figura 5.29.

Infelizmente, os vetores imagem-precisão **YMIN** e **YMAX** da silhueta tornam este algoritmo sujeito a problemas de *aliasing*. Se a primitiva linha utilizada possuir resolução maior do que os vetores de silhueta, o *aliasing* pode manifestar-se desenhando segmentos de linhas não visíveis ou ocultando segmentos visíveis. Por exemplo, a Figura 5.30 mostra três polylines desenhadas na ordem frente-fundo. As duas polylines de partes (a) e (b) formam um vazio quando elas se cruzam. A segunda linha pode ser corretamente ocultada pela primeira através da interpolação de valores em **YMAX**, determinando a interseção das duas linhas. Após a segunda linha ter sido desenhada, entretanto, ambos os valores em **YMAX** são iguais, assim a terceira linha, desenhada na parte (c), fica incorretamente ocultada quando ela passa sobre a interseção das duas primeiras linhas. O uso de vetores de silhueta com resolução maiores pode reduzir problemas como este, mas incrementam o tempo de processamento.

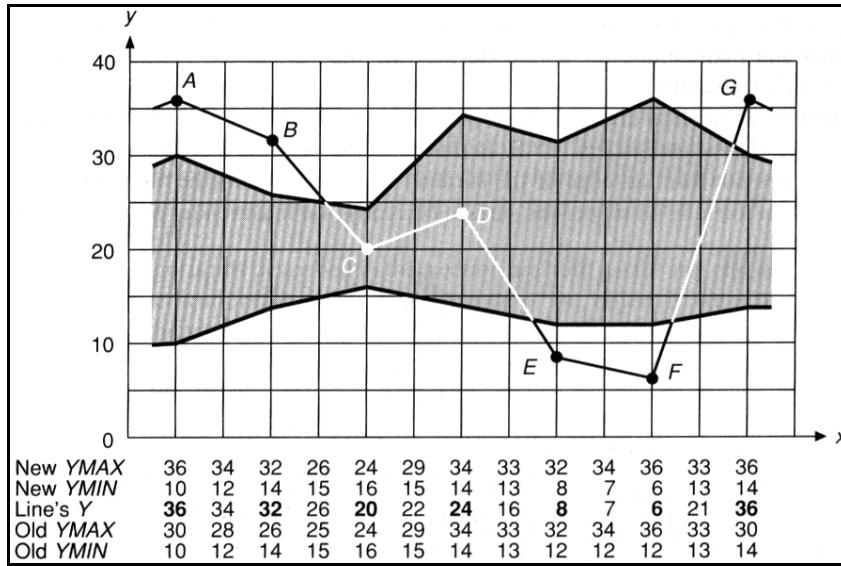


Figura 5.29 – Os valores de y para pontos entre os extremos dos segmentos devem ser interpolados e comparados com aqueles armazenados nos vetores da silhueta

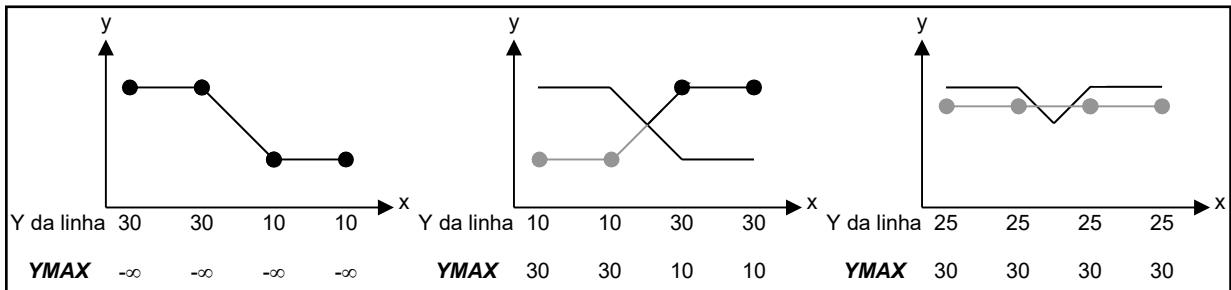


Figura 5.30 – Problema de *aliasing* ocasionados pela silhueta imagem-precisão

Uma alternativa aos vetores imagem-precisão $YMIN$ e $YMAX$ é o uso de estruturas de dados objeto-precisão. Os vetores $YMIN$ e $YMAX$ podem ser substituídos por duas *polylines* objeto-precisão representando as fronteiras da silhueta. Como cada segmento de uma *polyline* de coordenada z constante é verificado, somente as partes projetadas acima da *polyline* $YMAX$ ou abaixo da *polyline* $YMIN$ serão desenhadas. As linhas projetadas representando as partes visíveis são inseridas na *polyline* da silhueta a partir do ponto onde ocorreu a interseção, substituindo as partes subsequentes da *polyline* da silhueta até a próxima interseção. A maior precisão desta alternativa toma tempo de processamento devido às buscas e a manutenção das *polylines* da silhueta. Além disso, o algoritmo é mais complexo sendo mais difícil codificá-lo.

Suponha que desejemos desenhar *polylines* com coordenada x constante, ao invés de *polylines* de coordenada z constante. O resultado produzido é apresentado na Figura 5.31. Neste caso, a *polyline* com x constante mais próxima do observador não forma uma aresta da superfície. É a sétima *polyline* a partir da esquerda. Para desenhar corretamente a superfície, desenham-se as *polylines* à direita da mais próxima da esquerda para a direita, e aquelas à esquerda da mais próxima devem ser desenhadas da direita para a esquerda. Em ambos os casos, as *polylines* são desenhadas na ordem frente-fundo a partir do observador.

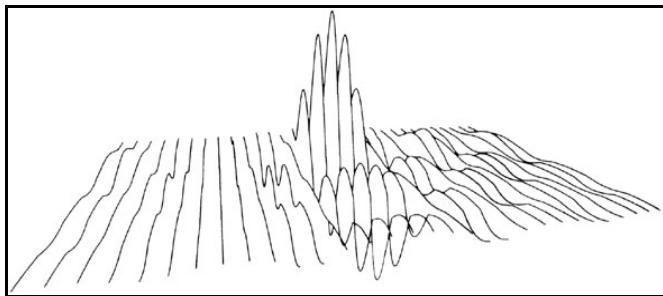


Figura 5.31 – Superfície desenhada a partir de *polylines* com x constante

O algoritmo imagem-precisão pode ser facilmente estendido para plotar linhas com x constante bem como linhas com z constante, como mostra a Figura 5.32.

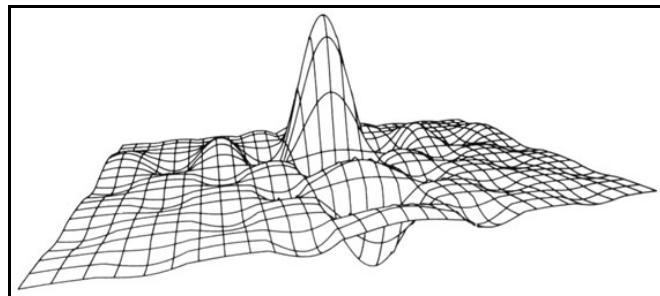


Figura 5.32 – Superfície da Figura 5.31 com linhas de coordenadas x e z constantes

Ainda que tentemos assumir que a superposição das *polylines* com x e z constantes, corretamente desenhadas, funcionará bem, isto não permite que linhas de um conjunto ocultem linhas de outro conjunto, como mostrado na Figura 5.33.

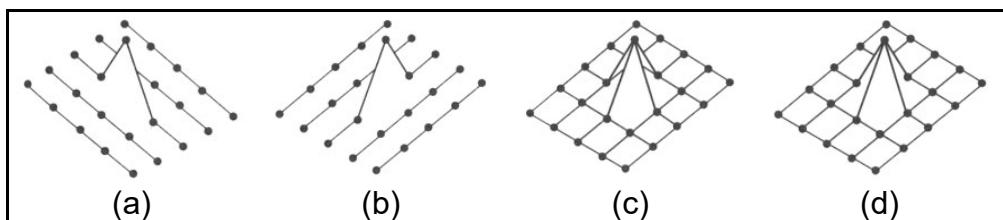


Figura 5.33 – (a) Linhas com z constante. (b) Linhas com x constante.
(c) Superposição das partes (a) e (b). (d) A solução correta.

Para que isso seja possível devemos intercalar o processamento dos dois conjuntos de *polylines*. O conjunto de *polylines* que está mais próximo do paralelismo com o plano de visualização (por exemplo, aquele com z constante) é processado na mesma ordem vista anteriormente. Depois que cada *polyline* com z constante é processada, os segmentos de cada *polyline* com x constante, entre duas *polylines* com z constante e subsequentes, são desenhados. Os segmentos de linhas em x devem ser desenhados utilizando a mesma cópia da estrutura de dados da silhueta usada para desenhar as *polylines* de z constante. Além disso, elas devem ser processadas na ordem frente-fundo.

A Figura 5.34(a) mostra linhas com x constante processadas na ordem correta, da esquerda para a direita neste caso. As linhas com x constante na Figura 5.34(b) foram desenhadas na ordem incorreta, da direita para a esquerda. A ordem incorreta de desenho causa problemas, porque as linhas são sombreadas, no vetor **YMAX**, pela linha desenhada anteriormente.

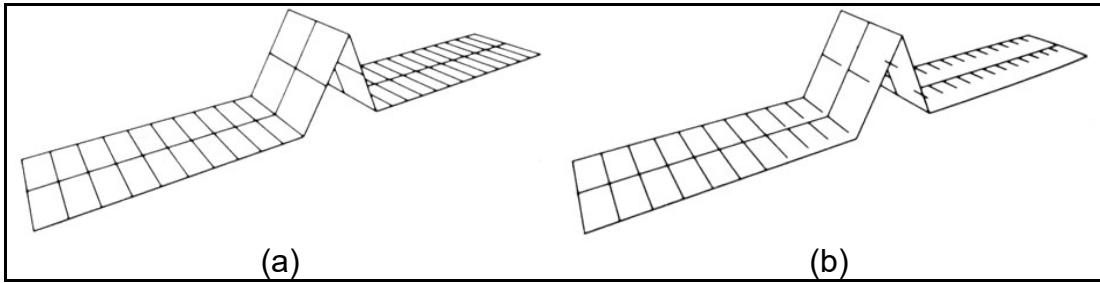


Figura 5.34 – *Polyline*s com x constante, como aquelas com z constante, devem ser processadas na ordem correta.(a) Linhas processadas corretamente. (b) Linhas processadas incorretamente

5.9.2 ALGORITMOS BÁSICOS PARA DETERMINAÇÃO DE SUPERFÍCIES VISÍVEIS

Existem dois algoritmos básicos para determinação de superfícies visíveis.

Algoritmo 1:

```
para cada pixel na imagem faça {
    • determinar o objeto mais próximo do observador que é visível pela
      projeção através do pixel;
    • plotar o pixel na cor apropriada;
}
```

Neste algoritmo vemos que é necessário, para cada pixel p , pesquisar todos os n objetos da imagem para procurar aquele mais próximo do observador.

Algoritmo 2:

```
para cada objeto no mundo faça
{
    • determinar quais partes do objeto cuja visão não está ocultada por
      partes do mesmo ou de outro objeto;
    • plotar estas partes nas cores apropriadas;
}
```

Embora este segundo algoritmo pareça ser de desempenho superior ao primeiro, sua implementação é mais complexa e o tempo consumido é maior.

O algoritmo 1 é conhecido como algoritmo imagem-precisão e o algoritmo 2 conhecido como algoritmo objeto-precisão. O algoritmo imagem-precisão trabalha com a resolução da tela e determina a visibilidade de cada pixel enquanto que o algoritmo objeto-precisão trabalha com a definição dos objetos e determina a visibilidade de cada um deles.

A vantagem do algoritmo objeto-precisão é necessitar executar apenas o último passo do algoritmo quando o tamanho da tela for alterado, pois a definição dos objetos visíveis não depende da resolução do monitor.

A determinação das superfícies visíveis é uma análise que deve ser feita no espaço 3-D, pois as projeções no espaço 2-D destroem as informações de profundidade necessárias para realizar as comparações.

Para podermos comparar a profundidade de dois pontos devemos responder a pergunta:

Um ponto ($P_1 = x_1, y_1, z_1$) oculta o outro ($P_2 = x_2, y_2, z_2$)?

A resposta da pergunta é **sim** se ambos os pontos estão na mesma linha de visada. Aí, então, podemos comparar suas coordenadas z_1 e z_2 .

Em projeções paralelas, se $x_1 = x_2$ e $y_1 = y_2$, então os pontos estão numa mesma linha de visada. Em projeções em perspectiva nós devemos comparar $x_1/z_1 = x_2/z_2$ e $y_1/z_1 = y_2/z_2$. Observadas as igualdades os pontos estão na mesma linha de visada.

Nos casos de projeções em perspectiva, transformamos as coordenadas 3D do objeto para coordenadas 3D do sistema de coordenadas da tela, criamos uma projeção paralela do objeto. Assim eliminamos a necessidade de realizar as divisões e simplesmente comparamos se $x_1 = x_2$ e $y_1 = y_2$.

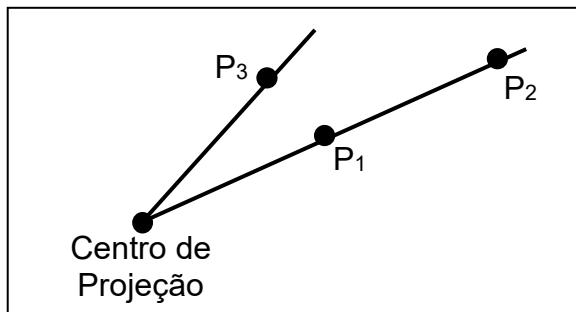


Figura 5.35– Um exemplo de projeção com pontos ocultos

Esta transformação é feita pela aplicação da seguinte matriz de transformação ao objeto projetado em perspectiva canônica (os pontos são expressos com coordenadas variando entre -1 e 1).

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{1+z_{\min}} & \frac{-z_{\min}}{1+z_{\min}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, z_{\min} \neq -1$$

Com $z_{\min} = \frac{n}{f}$, onde n e f são as distâncias aos planos de recorte da frente e do fundo, respectivamente.

5.9.3 ALGORITMOS PARA DETERMINAÇÃO DE SUPERFÍCIES VISÍVEIS

- **VISIBILIDADE POR PRIORIDADE:**

Se um objeto "A" bloqueia a visão de um objeto "B" e ambos os objetos encontram-se na mesma linha de visada do observador então o objeto "B" está mais distante que o objeto "A".

Usando esta linha simplificada de raciocínio pode-se criar um algoritmo que calcule a distância dos objetos ao observador, e priorizar a visualização dos objetos mais próximos do observador.

Neste processo calcula-se a distância de cada superfície ao observador e ordenam-se estas distâncias numa tabela de prioridades. Os objetos mais distantes são então apresentados na tela antes dos objetos mais próximos, por ordem decrescente de distância, de maneira que os objetos mais próximos ao observador são "sobrescritos" nos objetos mais distantes, "bloqueando" a sua visualização.

Existem diversas variações desta técnica em função da maneira pela qual se escolhe ou se calcula a distância de um objeto tridimensional. A aproximação mais simples para este cálculo é usar a distância do centróide ou centro de gravidade da superfície em questão. Esta distância, considerando o observador na origem do sistema de coordenadas, é dada por:

$$D = \sqrt{x^2 + y^2 + z^2}$$

onde x, y, z representam o centróide da face.

No entanto operações envolvendo raízes demandam uma grande quantidade de cálculos para serem solucionados e isso demanda tempo, assim a forma trigonométrica, embora aparentemente mais complexa, é preferível:

$$h = \frac{x}{\cos(\operatorname{arctg}(\frac{y}{x}))} \quad D = \frac{x}{\cos(\operatorname{arctg}(\frac{x}{h}))}$$

A ordenação por centróides apresenta algumas limitações como bem ilustra a figura a seguir:

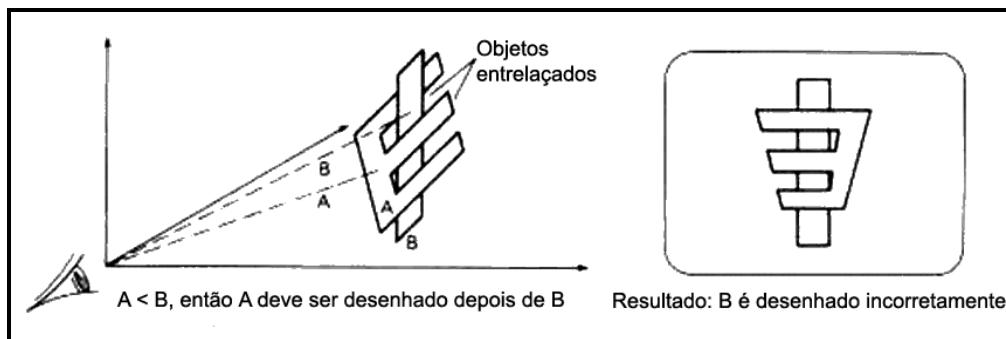


Figura 5.36 – Erro na determinação de visibilidade

A visibilidade por prioridade baseia-se também na falsa suposição de que uma superfície qualquer sempre domina o seu plano de visibilidade, ou seja, se "A" bloqueia "B" então "B" não pode bloquear "A", no entanto isto só é verdadeiro para polígonos convexos.

Polígonos côncavos ou de formatos "exóticos" podem bloquear uns aos outros. A solução para o problema está na execução de cálculos de distância em vários pontos do polígono, dividindo-o efetivamente em muitos polígonos convexos pequenos.

• ELIMINAÇÃO DE FACES OCULTAS PELO CÁLCULO DA NORMAL:

Pode-se também usar outra característica das superfícies, que não a distância ao observador, para determinar a sua visibilidade. Se observarmos uma

superfície diretamente, não podemos observar o seu lado oposto. Apenas uma face pode ser observada de cada vez, não importa qual o ângulo de visada.

Este método é particularmente aplicável aos poliedros convexos fechados. O "lado de trás" ou "de dentro" de uma superfície pode ser determinado pela orientação de uma reta normal ao plano.

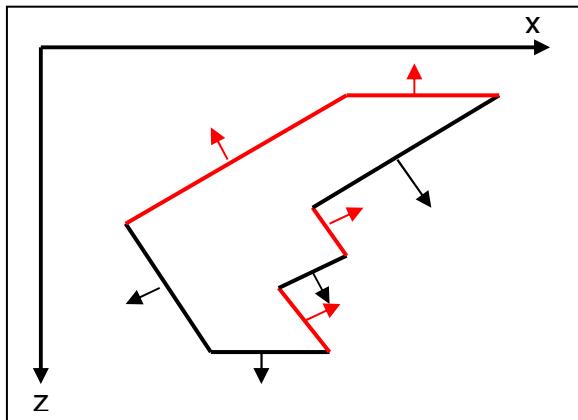


Figura 5.37 – Apresentação dos vetores normais às superfícies

Assumindo-se uma projeção ortogonal, uma superfície é visível quando uma reta normal ao plano aponta para o plano de projeção ou "tela". Uma face é invisível quando aponta na direção oposta.

A direção de uma reta normal ao plano pode ser obtida pela componente Z do vetor. Se a componente Z é positiva, então a reta aponta na direção do plano de projeção sendo, portanto, visível ao observador; se o valor de Z é negativo então a superfície está "de costas" para o observador e, portanto, não é visível. A equação de um plano definido por 3 pontos (P_1, P_2, P_3) pode ser obtida através de:

$$\begin{aligned} a &= (P_3y - P_2y) \cdot (P_1z - P_2z) - (P_1y - P_2y) \cdot (P_3z - P_2z) \\ b &= (P_3z - P_2z) \cdot (P_1x - P_2x) - (P_1z - P_2z) \cdot (P_3x - P_2x) \\ c &= (P_3x - P_2x) \cdot (P_1y - P_2y) - (P_1x - P_2x) \cdot (P_3y - P_2y) \\ d &= -(a \cdot P_2x) - (b \cdot P_2y) - (c \cdot P_2z) \end{aligned}$$

Onde: $ax + by + cz + d = 0$ é a equação do plano.

Aplicando-se a posição do observador, definida por $O[x, y, z]$, à equação do plano da seguinte maneira:

$$D = (a \cdot Ox) + (b \cdot Oy) + (c \cdot Oz) + d$$

tem-se que:

Se $D = 0$, então o observador é parte do plano.

Se $D > 0$, então o plano é visível ao observador.

Se $D < 0$, então o plano é invisível ao observador.

A eliminação de faces ocultas pelo cálculo da normal, por si só, não constitui uma técnica completa para determinação de superfícies ocultas, uma vez que sua aplicação é limitada apenas aos poliedros convexos fechados. Pode parecer que seu uso é pouco recomendável. No entanto, esta técnica tem a capacidade de, em muitos casos, eliminar mais da metade das superfícies a serem

consideradas para cálculo. Como o tempo de processamento de muitas rotinas de determinação de faces ocultas cresce exponencialmente em função do número de faces (principalmente devido às rotinas de ordenação), a aplicação desta técnica como um "pré-filtro" costuma trazer uma redução no tempo de processamento de aproximadamente 75%.

- **ALGORITMO Z-BUFFER:**

O algoritmo de *z-buffer*, desenvolvido por E. Catmull em 1974, é um dos algoritmos de determinação de visibilidade de superfícies mais simples de se implementar em software ou hardware.

Sua implementação é baseada no uso intensivo de memória e requer a alocação de 2 *buffers*, ou matrizes em memória, com dimensões idênticas à tela de apresentação.

Denomina-se o primeiro, *buffer* de Imagem, o qual deve possuir a mesma profundidade de cor da tela de apresentação final. Utiliza-se o *buffer* de Imagem como "rascunho" durante os cálculos de visibilidade dos objetos.

O segundo *buffer* destina-se a armazenar a "profundidade" ou distância de cada pixel da tela rascunho ao plano de projeção, sendo chamado de *z-buffer* propriamente dito.

O *buffer* de profundidade ou *z-buffer* é inicializado com um valor que representa o valor máximo da distância de um ponto ao plano de projeção. O *buffer* de Imagem deve ser inicializado com o valor das cores de fundo da imagem.

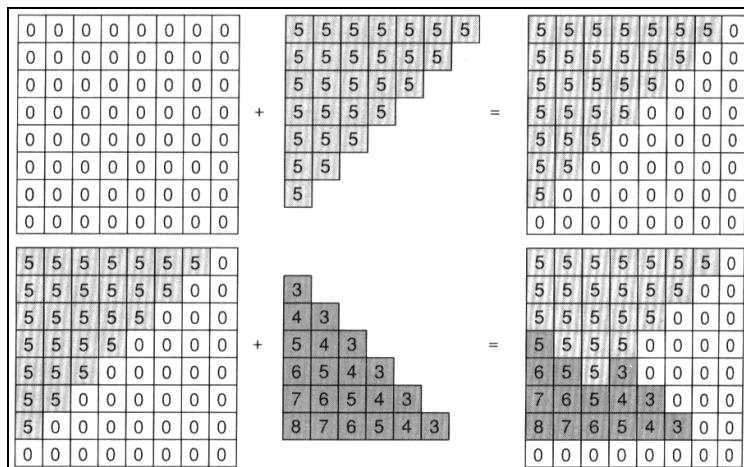


Figura 5.38 – Um exemplo de aplicação de *z-buffer*

Os polígonos que compõem ou representam os objetos no espaço 3D são projetados na tela de rascunho, pixel a pixel. Para cada novo pixel projetado na tela de rascunho é efetuado o cálculo de distância em relação ao plano de projeção; se esta distância for inferior à distância armazenada no *z-buffer*, para aquela coordenada, então os valores de cor deste pixel substituem os valores anteriormente armazenados na tela de rascunho e uma nova profundidade é armazenada no *z-buffer*, para aquela posição. Se os valores de distância são maiores que os presentes no *z-buffer* então este pixel está sendo "bloqueado" por um ponto, previamente calculado, que já se encontra mais próximo do plano de projeção.

A equação geral da equação do plano é dada por:

$$ax + by + cz + d = 0$$

Assumindo-se uma projeção paralela no plano XY, temos que a componente Z (profundidade) de um ponto qualquer no plano pode ser obtida da seguinte maneira:

$$z = \frac{-d - ax - by}{c}$$

Para um ponto P (x_1, y) pertencente ao mesmo plano temos:

$$z_1 = \frac{-d - ax_1 - by}{c}$$

Assim, a distância entre z e z_1 , expressa por Δz é calculada por:

$$\Delta z = z - z_1 = -\frac{a}{c}(\Delta x)$$

A aplicação prática desta conveniente proporção entre as profundidades se aplica diretamente para o preenchimento dos polígonos, limitando o cálculo completo da profundidade apenas ao primeiro ponto da linha de preenchimento (ou *scan line*).

Apenas uma subtração é necessária para o cálculo das demais profundidades da *scan line*, uma vez que a relação $\frac{a}{c}$ é constante.

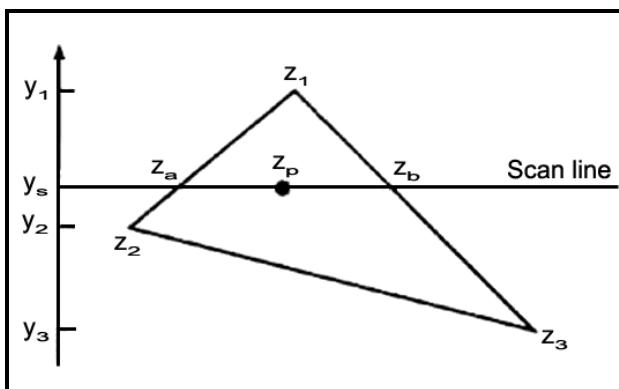


Figura 5.39 – Interpolação de valores de z entre as arestas de um polígono usando *scan lines*

Note-se também que a cor de um pixel não precisa ser calculada se a condicional de visibilidade não for satisfeita, pouparindo cálculos desnecessários. O algoritmo de *z-buffer* também não requer que os objetos sejam polígonos ou impõe qualquer restrição à forma. Nenhum algoritmo explícito de cálculo de interseção precisa ser utilizado.

Os *z-buffers* são geralmente implementados com 16 ou 32-bit inteiros, embora algumas implementações usem valores de ponto-flutuante.

- **RAY TRACING:**

Este método determina a visibilidade de superfícies pelo traçado imaginário de um raio de luz que parte do olho do observador até os objetos na cena. É o algoritmo típico de imagem-precisão. Um centro de projeção (o olho do

observador) e um plano de projeção para o *window* são selecionados. O plano de projeção pode ser dividido numa grade regular, cujos elementos correspondem a pixels numa resolução desejada.

Para cada pixel um raio sai do olho do observador, do centro de projeção, passando através do centro do pixel até atingir a cena. O pixel vai assumir a cor do objeto mais próximo do ponto de interseção, como mostrado na figura abaixo.

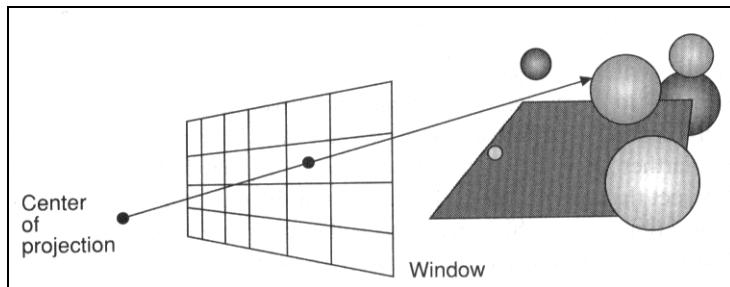


Figura 5.40 – Um raio de luz é direcionado partindo do centro de projeção, passando através de cada pixel da imagem determinando o objeto mais próximo

O núcleo de qualquer *ray tracer* é a tarefa de determinar a interseção do raio com um objeto. Para isso devemos utilizar a representação paramétrica de vetores. Cada ponto (x, y, z) ao longo do raio de (x_0, y_0, z_0) até (x_1, y_1, z_1) é definido por algum valor t tal que:

$$x = x_0 + t(x_1 - x_0) \quad y = y_0 + t(y_1 - y_0) \quad z = z_0 + t(z_1 - z_0)$$

Por conveniência, escrevemos:

$$x = x_0 + t \cdot \Delta x \quad y = y_0 + t \cdot \Delta y \quad z = z_0 + t \cdot \Delta z$$

Para encontrar o ponto de interseção do raio com os objetos da cena devemos ter equações que representem os objetos vistos na cena. Caso o objeto visto na cena é uma esfera, sua equação é dada por:

$$(x - a)^2 + (y - b)^2 + (z - c)^2 = r^2$$

A interseção é encontrada substituindo os valores de x , y e z (vetores paramétricos do raio) na equação da esfera. Assim obtemos:

$$(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + 2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] + (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0$$

Resolvendo a expressão acima podemos obter:

- nenhuma raiz real: o raio não intercepta a esfera;
- uma raiz real: o raio tangencia a esfera;
- duas raízes reais: o raio intercepta a esfera na raiz positiva de menor valor.

Encontrar a interseção com um polígono é um pouco mais difícil. Nós devemos, primeiramente, determinar se o raio intercepta o plano do polígono; depois

verificamos se o vértice de interseção está dentro dos limites do polígono. Dada a equação do plano:

$$Ax + By + Cz + D = 0$$

Substituindo os valores de x, y e z (vetores paramétricos) obtemos:

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)}$$

Se o denominador for nulo, o raio e o plano são paralelos e não se interceptam.

O algoritmo *ray tracing* é computacionalmente dispendioso. Numa tela de 1024 x 1024 pixels, com 100 objetos, serão necessários 100 milhões de cálculos de interseção. Isto representa de 75 a 95% do tempo gasto no processo. Uma solução para isso seria utilizar um pré-filtro com o algoritmo de eliminação de faces ocultas pelo cálculo da normal.

UNIDADE 6 – ALGORITMOS RASTER BÁSICOS PARA DESENHO DE PRIMITIVAS EM 2D

6.1 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 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.

6.2 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. 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.

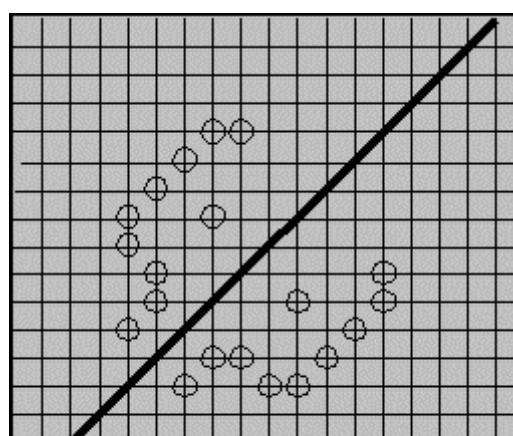


Figura 6.1 - Reflexão de uma imagem em relação a um eixo diagonal

6.3 CARACTERÍSTICAS DESEJÁVEIS PARA OS ALGORITMOS DE CONVERSÃO MATRICIAL DE SEGMENTOS DE RETAS

- a) **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.
- b) **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.
- c) **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.
- d) **Intensidade independente da inclinação:** para segmentos de diferentes inclinações
- e) **Continuidade:** a imagem não apresenta interrupções indesejáveis.
- f) **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 a), 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 b) é problemático apenas quando os pontos extremos dos segmentos são especificados em coordenadas do usuário. Para o requisito c), conta-se o número de pixels traçados, e divide-se pelo comprimento da linha. Os requisitos d) e f) são avaliados através da execução de *benchmarks* em cada algoritmo, para efeito de comparação. O requisito e) 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.

6.3.1 CRITÉRIO ADOTADO

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

Supondo que este segmento esteja no 1º octante, temos que este ponto respeita as seguintes relações:

$$0 < x_1 < x_2$$

$$0 < y_1 < y_2$$

$$y_2 - y_1 < x_2 - x_1$$

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 reticulado 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 2º octante não produz resultados satisfatórios, pois nesse caso o número de verticais que corta o segmento será menor que o de horizontais. A solução para segmentos no 2º octante é utilizar este critério só que trocando x com y e vertical por horizontal.

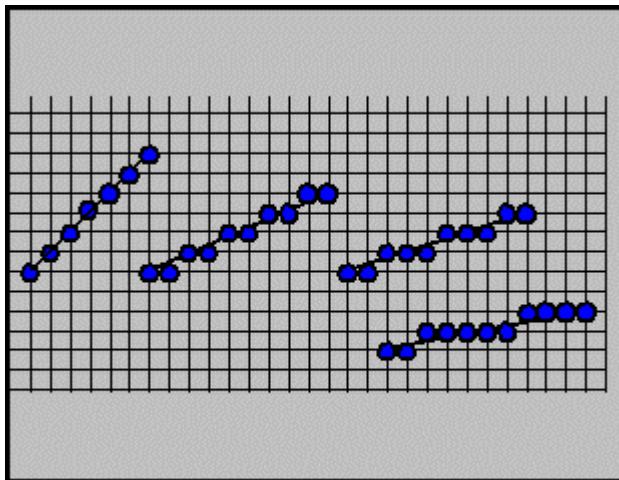


Figura 6.2 - Imagens de Segmentos Convertidos pelo Critério Explicitado Acima

6.4 ALGORITMOS BÁSICOS PARA O TRAÇADO DE RETAS

Equação da reta:

$$y = y_1 + m(x - x_1)$$

onde:

$$m = \frac{(y_2 - y_1)}{(x_2 - x_1)}$$

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 trecho de código abaixo implementa esta ideia. Este algoritmo é pouco eficiente, pois utiliza cálculos em ponto flutuante.

Algoritmo Incremental

```
void Line(          /* Assume -1<=m <=1, x0 < x1*/
    int x0, int y0,      /* Ponto da esquerda*/
    int x1, int y1,      /* Ponto da direita*/
    int value)           /* Cor do ponto*/
{
    int x              /* x varia de x0 a x1*/

    double dy = y1 - y0;
    double dx = x1 - x0;
    double m = dy/dx;
```

```

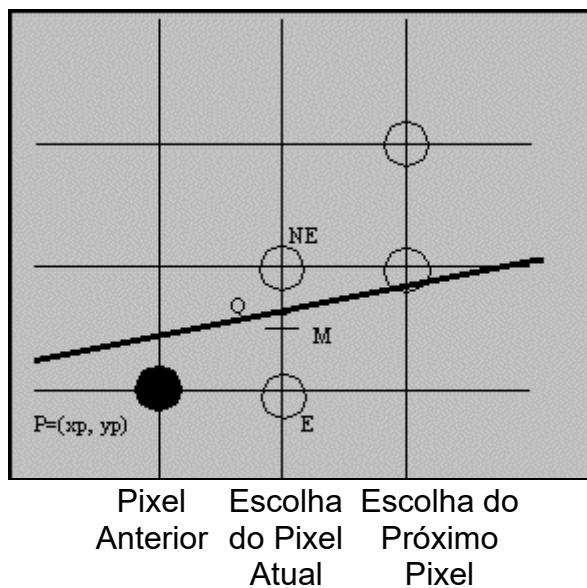
double y = y0;

for (x = x0; x<=x1; x++) {
    WritePixel (x, Round(y), value); /*plota o pixel*/
    y += m;                         /*incrementa y*/
}
}

```

6.4.1 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 inteiras, 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) .



Algoritmo do Ponto Médio

```

void MidpointLine (int x0, int y0, int x1, int y1, int value)
{
    int dx = x1 - x0;
    int dy = y1 - y0;
    int d = 2 * dy - dx;           /*valor inicial de d*/
    int incrE = 2*dy;             /*incremento usado para mover para E*/
    int incrNE = 2*(dy - dx);    /*incremento usado para mover para NE*/
    int x = x0;
    int y = y0;
    WritePixel (x, y, value); /* plota o ponto inicial*/

    while (x < x1) {
        if (d <= 0 ) {      /*escolhe E*/
            d += incrE;
            x++;
        } else {            /*escolhe NE*/
            d += incrNE;
            x++;
            y++;
        }
        WritePixel (x, y, value); /*Plota o ponto final*/
    }
}

```

6.5 CONVERSÃO MATRICIAL DE CIRCUNFERÊNCIAS

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

$$x^2 + y^2 = R^2$$

Circunferências não centradas na origem podem ser transladadas para a origem, e os pixels reais podem ser traçados aplicando-se um deslocamento 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 é utilizar a própria equação da circunferência, isolando o termo y :

$$y = \pm\sqrt{R^2 - x^2}$$

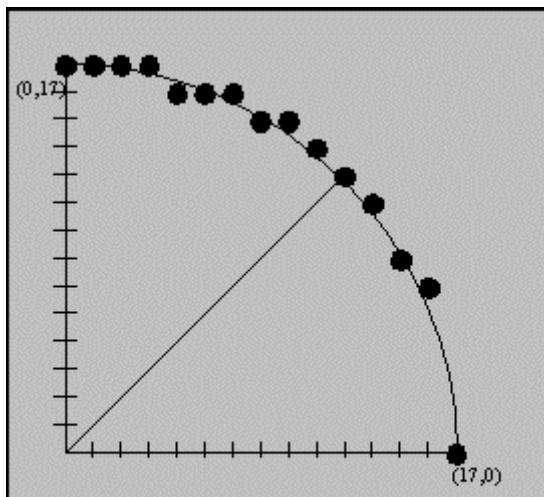


Figura 6.3 - Um arco, de $\frac{1}{4}$ de circunferência, obtido variando-se x em incrementos unitários, calculando e arredondando y.

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 disso, haverá grandes *gaps* nas regiões onde a tangente à circunferência é infinita (valores de x próximos a R, Figura 6.3). Uma maneira de resolver o problema dos gaps é plotar $R.\cos$ e $R.\sin$, para variando de 0 a 90 graus, mas essa solução também é inefficiente, pois precisa de funções caras – seno e cosseno.

6.5.1 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 trivialmente sete outros pontos da circunferência. 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* indicado a seguir.

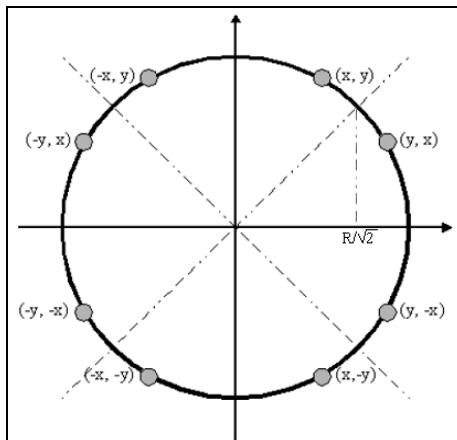


Figura 6.4 - Oito pontos simétricos em uma circunferência

Procedimento CirclePoints

```
void CirclePoints (int x, int y, int value)
{
    WritePixel (x, y, value);
    WritePixel (y, x, value);
    WritePixel (y, -x, value);
    WritePixel (x, -y, value);
    WritePixel (-x, -y, value);
    WritePixel (-y, -x, value);
    WritePixel (-y, x, value);
    WritePixel (-x, y, value);
}
```

6.5.2 ALGORITMOS DO PONTO-MÉDIO PARA CIRCUNFERÊNCIAS

Consideraremos apenas um arco de 45° da circunferência, o 2º octante, de $x = 0$, $y = R$ a $x = y = R/(2)^{1/2}$, 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.

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 ponto 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”.

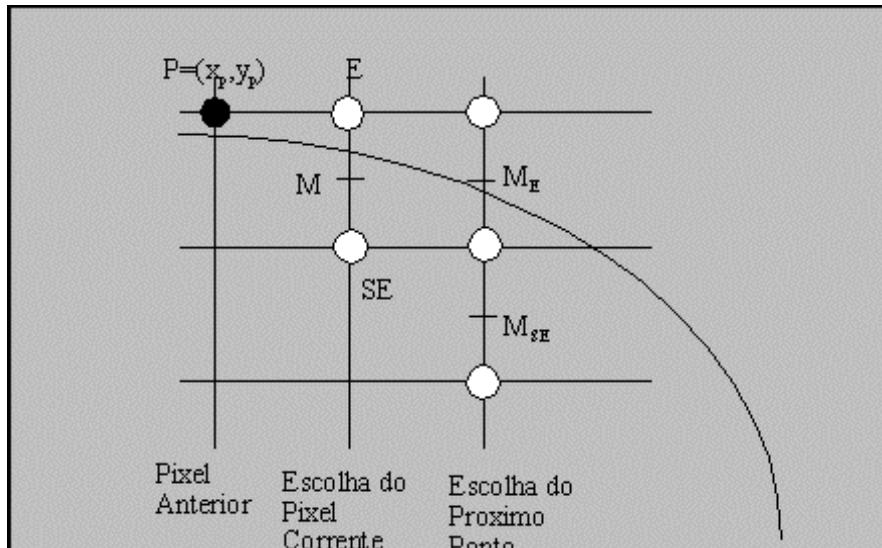


Figura 6.5 - Malha de pixels para o algoritmo do Ponto-médio para circunferências, ilustrando a escolha entre os pixels E e SE.

Algoritmo do Ponto-médio para Circunferências

```
void MidpointCircle (int radius, int value)
/* Assume que o centro do círculo está na origem*/
{
    int x = 0;
    int y = radius;
    int d = 1 - radius;
    CirclePoints (x, y, value);

    While (y > x) {
        If (d < 0 )           /*escolhe E*/
            d += 2 * x + 3;
        else {                 /*escolhe SE*/
            d += 2 * (x - y) + 5;
            y--;
        }

        x++;
        CirclePoints (x, y, value);
    }
}
```

6.6 CONVERSÃO MATRICIAL DE ELIPSES

Consideremos a elipse padrão da Figura 6.6, centrada em $(0, 0)$. Ele é descrita pela equação:

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

onde:

2a - o comprimento do eixo maior (eixo x)
2b - o comprimento do eixo menor (eixo y)

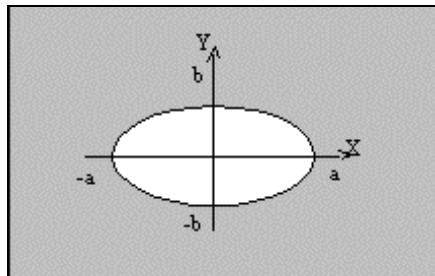


Figura 6.6 - Uma elipse padrão centrada na origem

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

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

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

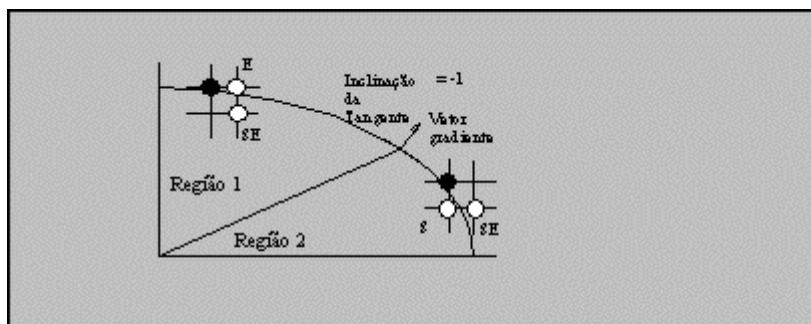


Figura 6.7 - As duas regiões adotadas, definidas pela tangente a 45°

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 i e j são de magnitudes iguais. A componente j do gradiente é maior do que a i na região 1, e vice-versa na região 2. Assim, se o próximo "ponto-médio" é $a^2(y_p - 1/2)b^2(x_p + 1)$, nós mudamos da região 1 para a região 2 (Figura 6.7).

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.

6.7 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.

Algoritmo do Ponto-médio para Elipses

```

void MidpointElipse (int a, int b, int value){
/* Assume que o centro da elipse é a origem. Note que erro de
overflow pode ocorrer para inteiro de 16-bit devido aos quadrados*/

    int a2 = a * a;           int b2 = b * b;
    int twoa2 = 2 * a2;       int twob2 = 2 * b2;
    int x = 0;                int y = b;
    int px = 0;               int py = twoa2 * y;
    int p;

    EllipsePoints (x, y, value);

    p = int(b2 - (a2 * b) + (0.25 * a2) + 0.5);
    while (px < py) {
        x++;
        px += twob2;
        if (p < 0) p += b2 + px
        else {
            y--;
            py -= twoa2;
            p += b2 + px - py;
        }
        EllipsePoints (x, y, value);
    }

    p = int(b2 * (x+0.5)*(x+0.5) + a2 * (y-1)*(y-1) - a2 * b2 + 0.5);
    while (y > 0) {
        y--;
        py -= twoa2;
        if (p > 0) p += a2 - py;
        else {
            x++;
            px += twob2;
            p += a2 - py + px;
        }
        EllipsePoints (x, y, value);
    }
}

```

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 corrigi-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.

6.8 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 denominada *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) (Figura 6.8). 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.

6.8.1 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 certa espessura que cobre uma região da malha de pixels, como ilustrado na Figura 6.9. Consequentemente, 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.

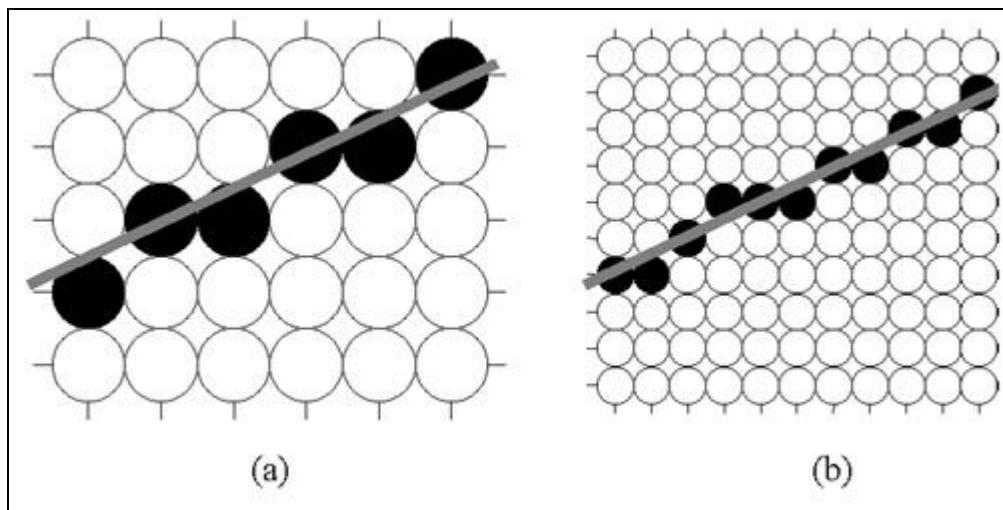


Figura 6.8

- (a) Segmento de reta renderizado com algoritmo do ponto médio em um display
 (b) Segmento de reta renderizado em um display com resolução maior

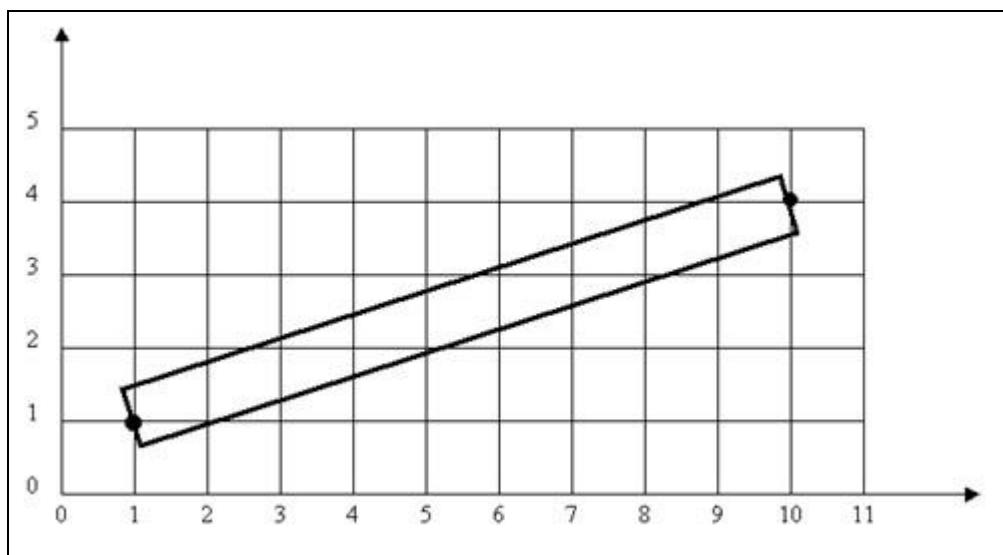


Figura 6.9 – Segmento de reta (1, 1) a (10, 4) definido com uma espessura diferente de zero

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 interseções da malha definem o posicionamento dos pixels, isto é, 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 6.10). O efeito é "suavizar" a definição 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*).

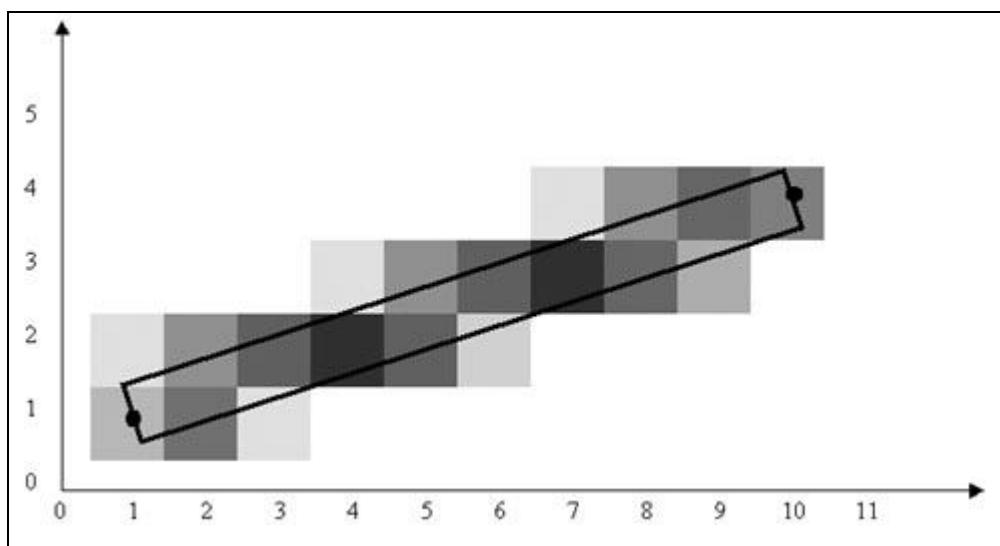


Figura 6.10 – A intensidade do pixel é proporcional à área coberta

6.8.2 AMOSTRAGEM DE ÁREAS PONDERADAS

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 a 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.

6.9 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. 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.

6.9.1 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 frequentemente 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.

Algoritmo para preenchimento de retângulos

```
void FillRect (int x1, int y1, int x2, int y2, int value){

    int x, int y;

    for (y = y1; y < y2; y++) do
        for (x = x1; x < x2; x++) do
            writepixel (x, y, value);

}
```

Os pixels que estão sobre os lados (arestas) 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 pertencem ao lado que está mais a direita. Sendo que na realidade, *blocos* dentro de um retângulo representam um intervalo que é fechado à esquerda e embaixo e aberto em cima e à 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. 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.

6.9.2 POLÍGONOS DE FORMA ARBITRÁRIA

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

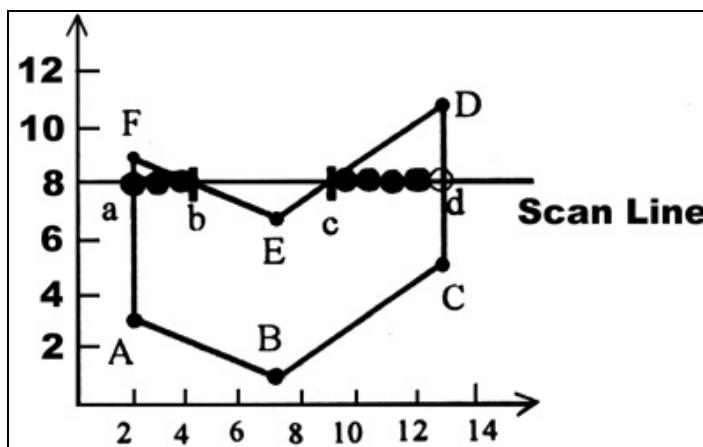


Figura 6.11 - Linhas de varredura

A Figura 6.11, ilustra o processo de linha de varredura para um polígono arbitrário. As interseções da linha de varredura 8 com os lados *FA* e *CD* possuem coordenadas inteiras, enquanto as interseções com os lados *EF* e *DE* possuem coordenadas reais.

É preciso determinar quais pixels da linha de varredura estão dentro do polígono, e estabelecer os pixels correspondentes (no exemplo da Figura 6.11, *bloco* 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.

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

1. Obter a interseção da linha de varredura com todos os lados do polígono.
2. Ordenar os pontos de interseção (em x crescente).
3. Preencher os pixels entre pares de pontos de interseção 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 interseção encontrada o bit de paridade é invertido. O pixel é pintado quando a paridade é ímpar e não é pintado quando é par.

Na Figura 6.11 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 interseção 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 interseção fracionária pela direita, arredondamos a coordenada x da interseção 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 interseção 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 y_{min} do lado adjacente. Por exemplo, na Figura 6.11, o vértice A é contado uma vez no cálculo de paridade porque é o vértice de y_{min} para o lado FA , mas é também o vértice de y_{max} 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 y_{min} ou y_{max} .

Ilustremos este processo, aplicando essas regras à linha de varredura 8, na Figura 6.11. 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 y_{min} do lado FA (e é também o vértice y_{max} do lado AB), isto causa paridade ímpar, assim o bloco é desenhado a partir de A até um pixel a esquerda da interseção com o lado CB , onde a paridade é estabelecida como par terminando o bloco. A linha de varredura 1 passa apenas pelo vértice B , e os lados AB e BC tem como vértice de y_{min} 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 interseção da linha de varredura 9 com o vértice F , compartilhado pelos lados FA e EF . Para ambos os lados, B é vértice y_{max} , e dessa forma não afeta a paridade, que continua par.

6.9.3 ARESTAS HORIZONTAIS

Examinemos os casos apresentados na Figura 6.12. Consideremos o lado AB (lado inferior). O vértice A é vértice y_{min} 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 y_{min} 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 y_{min} 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 y_{max} para BC , assim o bloco continua sobre o lado CD (até D), no entanto DE tem y_{min} em D e a paridade torna-se par e o bloco termina.

O lado IJ tem y_{max} em I e o lado HI não contribui para o cálculo da paridade, assim a paridade continua e o bloco sobre o lado HI não é desenhado.

Entretanto, o lado GH tem vértice y_{min} em H , e a paridade torna-se ímpar, e o bloco é desenhado a partir de H até o pixel a direita da interseção da linha de varredura com o lado EF .

Finalmente não há vértice y_{min} em G nem em F , e dessa forma o lado (que é o lado superior - topo) FG não é desenhado.

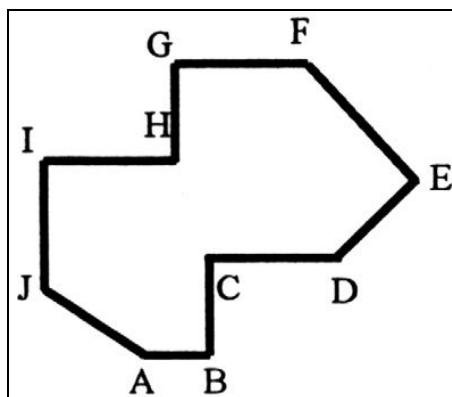


Figura 6.12 – Tratamento dos lados horizontais de um polígono

6.9.4 SLIVERS

Existe 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 6.13).

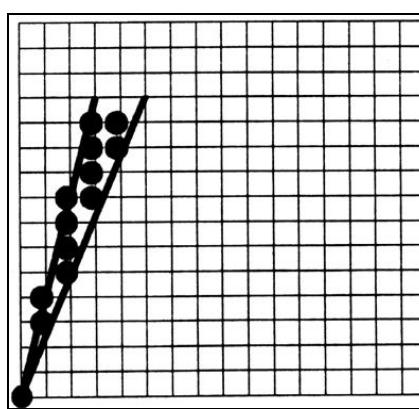


Figura 6.13 – Conversão matricial de um “sliver” de um polígono

Devido à regra de que são traçados apenas os pixels que estejam no

interior ou sobre arestas inferiores ou a esquerda, podem existir muitas linhas de varredura com um único pixel, ou sem nenhum. A ausência de pixels é 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.

6.10 PREENCHIMENTO COM PADRÓES (PATTERNS)

Preenchimento com padrões (patterns) dá-se em dois estágios:

- A determinação da matriz de pontos que comporá o padrão;
- A determinação, para um pixel qualquer, qual cor da matriz devemos utilizar para o pixel.

6.10.1 MATRIZ DE PONTOS

A matriz de pontos constitui-se numa matriz que mostra como o padrão é definido no mapa de pixels. O tamanho deste mapa é determinado pelo programador. No caso do MS-Windows, por exemplo, são utilizados padrões de 4x4 pixels em apenas duas cores.

Tomemos, por exemplo, essa matriz de 4x4 pixels que definirá um padrão semelhante a um tabuleiro de xadrez:

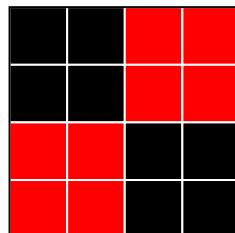


Figura 6.14 - Uma matriz 4x4 definindo um padrão de preenchimento

A implementação desta matriz, em Pascal, ficaria da seguinte maneira:

```

MATPIX[0, 0] := BLACK; {MATPIX[linha, coluna] = MATPIX[I, J]}
MATPIX[0, 1] := BLACK;
MATPIX[0, 2] := RED;
MATPIX[0, 3] := RED;
MATPIX[1, 0] := BLACK;
MATPIX[1, 1] := BLACK;
MATPIX[1, 2] := RED;
MATPIX[1, 3] := RED;
MATPIX[2, 0] := RED;
MATPIX[2, 1] := RED;
MATPIX[2, 2] := BLACK;
MATPIX[2, 3] := BLACK;
MATPIX[3, 0] := RED;

```

```
MATPIX[3, 1] := RED;
MATPIX[3, 2] := BLACK;
MATPIX[3, 3] := BLACK;
```

6.10.2 DETERMINAÇÃO DA COR PARA UM PIXEL QUALQUER NA TELA

A determinação da cor de um pixel na tela é feita da seguinte maneira:

- Divide-se o objeto a ser preenchido por um padrão;
- Para cada pixel (x, y) do objeto é calculada a cor do pixel correspondente à matriz que define o padrão.

Este cálculo é realizado através do operador "mod". Por exemplo, utilizando a matriz 4×4 definida anteriormente, um pixel de coordenadas $(30, 20)$ terá cor definida pelas seguintes coordenadas i, j da matriz do padrão:

$$\begin{aligned} i &= 20 \bmod 4 = 0; \\ j &= 30 \bmod 4 = 2; \end{aligned}$$

Portanto, a cor do pixel de coordenadas $(30, 20)$ terá a cor definida pelo elemento $\text{MATPIX}[0, 2] = \text{RED}$.

Então, em linguagem Pascal, a implementação desta definição ficaria assim:

```
function PEGA_COR (X, Y, HX, HY : integer) : byte;
{X e Y são as coordenadas do pixel pertencente ao objeto}
{HX, HY são a largura e altura da matriz do padrão}
var
  I, J : byte;
begin
  I := Y mod HY;
  J := X mod HX;
  PEGA_COR := MAT_PIX[I, J];
end;
```

6.11 PRIMITIVAS ESPESSAS

Podemos produzir primitivas espessas a partir de uma linha base obtida através de um algoritmo de conversão matricial para linhas. Para isto basta ajustarmos um pincel, com um determinado formato de seção transversal, em cada um dos pontos obtidos pelo algoritmo de conversão matricial. Uma linha com um pixel de largura pode ser entendida como se fosse desenhada com um pincel de tamanho igual a um pixel.

Abordaremos três métodos para traçar primitivas espessas: replicação de pixels, movimento da caneta e preenchimento de área entre dois limites.

6.11.1 REPLICAÇÃO DE PIXELS

A inserção de um laço de repetição, no algoritmo de conversão matricial de linhas, permitirá plotar múltiplos pixels para cada pixel definido pelo algoritmo;

aqui, pixels são duplicados em colunas para linhas com $-1 < \text{coeficiente angular} < 1$ e em filas para todas as outras linhas. O efeito, entretanto, é que as linhas terminam sempre em traços verticais ou horizontais, o que não é interessante para linhas espessas, como mostra a Figura 6.15.

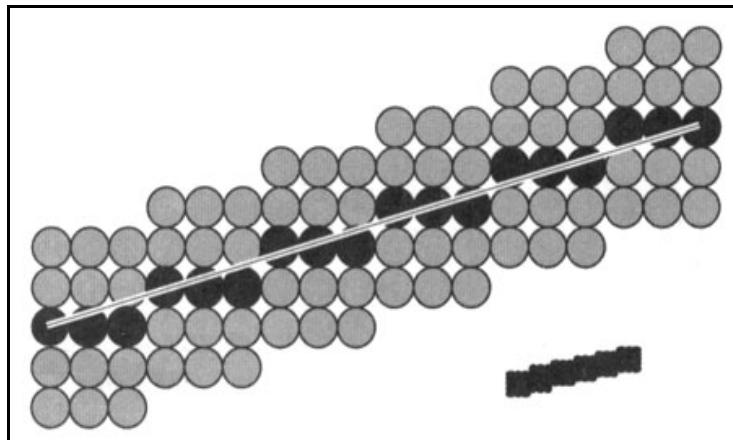


Figura 6.15 – Linha espessa gerada pela replicação em coluna

Podemos ajustar a forma no final das linhas, dando-lhes melhor aspecto, através da adição de capas (Figura 6.16). Um tipo de capa final é a “**butt cap**” obtida pelo ajuste dos extremos da linha de forma que a linha espessa seja exibida com extremos quadrados perpendiculares à mesma. Se a inclinação da linha é dada por m , o quadrado final da linha espessa tem inclinação $-1/m$.

Outra capa utilizada é a “**round cap**” obtida através da adição de um semicírculo preenchido para cada “butt cap”. Os arcos circulares são centrados nos pontos extremos da linha e têm diâmetro igual à espessura da mesma.

Um terceiro tipo de capa é a “**projecting square cap**” ou “capa de projeção quadrada”. Aqui, simplesmente estendemos a linha e adicionamos “butt caps” que são posicionadas metade da largura da linha além dos extremos.

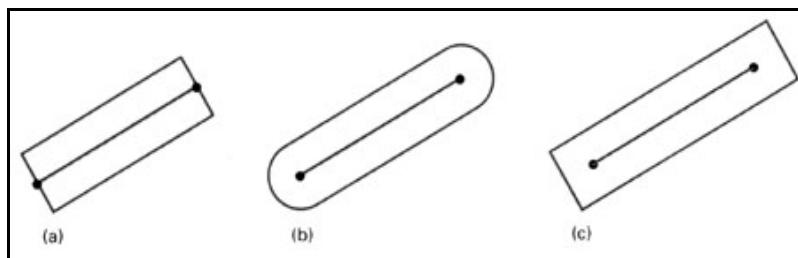


Figura 6.16 – Linhas espessas com (a) “butt caps”, (b) “round caps” e (c) “projecting square caps”

A geração de linhas múltiplas (*polylines*) requer algumas considerações adicionais. Geralmente, os métodos utilizados para traçar linhas espessas não produzem conexões suaves entre os segmentos da *polyline*. Podemos gerar *polyline* espessas com conexões suaves processando os extremos dos segmentos. A Figura 6.17 mostra três métodos para suavizar as junções entre dois segmentos de linhas. Uma **junção turbante** é efetuada estendendo-se as fronteiras externas de cada um dos segmentos até que eles se encontrem. Uma **junção redonda** é produzida adicionando-se um círculo com diâmetro igual à largura da linha sobre a conexão entre os segmentos. Uma **junção em bisel** é gerada criando segmentos de linhas com “butt caps” e preenchendo a falha triangular no encontro dos segmentos. Se o

ângulo entre dois segmentos de linhas conectados for muito pequeno, uma junção turbante pode gerar uma ponta alongada que distorce a aparência da *polyline*. Nestes casos alterna-se de uma junção turbante para uma junção em bisel.

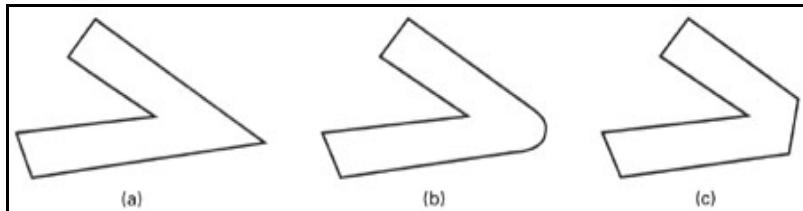


Figura 6.17 – Segmentos de linhas espessas conectadas com (a) junção turbante, (b) junção redonda e (c) junção em bisel

Os algoritmos de replicação de pixels também produzem falhas perceptíveis nos pontos onde os segmentos de linhas se encontram, desde que haja um ângulo entre os segmentos. Ainda, pixels podem desaparecer onde há mudanças na replicação de horizontal para vertical em função da inclinação dos segmentos. Esta última anomalia gera espessuras anormais num arco de elipse nas proximidades dos octantes, como mostra a Figura 6.18.

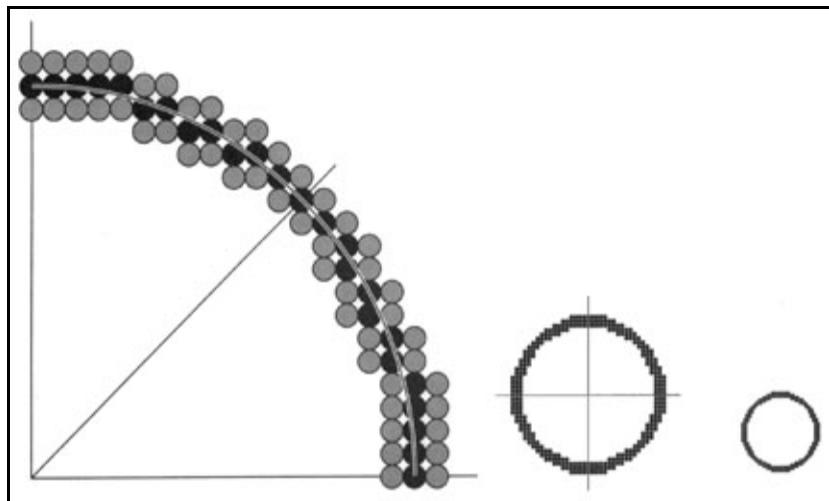


Figura 6.18 – Círculo espesso gerado através da replicação em linha e coluna em função da inclinação do segmento

Além disso, linhas horizontais e verticais apresentam espessuras diferentes das linhas inclinadas. Se a espessura for definida como t para linhas horizontais e verticais, a espessura para uma linha inclinada em 45° será em média $t/\sqrt{2}$. Outro problema com a replicação de pixels é o problema das larguras pares: não podemos centralizar as colunas ou linhas replicadas num determinado pixel; assim escolhemos um lado da primitiva que terá um pixel extra. No geral, a replicação de pixels é um método eficiente, mas aproximado, que funciona melhor para primitivas não muito espessas.

6.11.2 MOVIMENTO DA CANETA

Escolhendo uma caneta de seção transversal retangular cujo centro ou canto desloque-se ao longo de uma linha primitiva podemos produzir linhas

espessas razoáveis, como a mostrada na Figura 6.19. Perceba que esta linha é similar àquela produzida pela replicação de pixels mas estende-se um pouco mais nas extremidades.

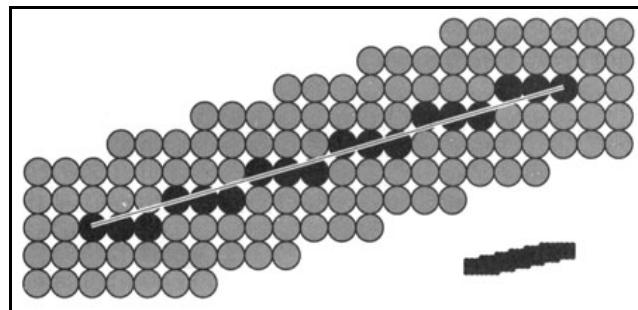


Figura 6.19 – Linha espessa gerada pelo movimento de uma caneta retangular

Como na replicação de pixels, devido à caneta permanecer alinhada verticalmente, a espessura da linha varia conforme o ângulo da primitiva, mas em situação oposta: a largura é menor para linhas horizontais e verticais, e maior para segmentos com inclinação entre ± 1 . Um arco de elipse, por exemplo, varia sua espessura ao longo de toda a trajetória, desde a espessura inicialmente desejada quando a tangente está próxima da horizontal e da vertical, e mais espessa num fator $\sqrt{2}$ quando a tangente estiver por volta de 45° (veja a Figura 6.20). Este problema pode ser resolvido se o quadrado que define a caneta girar ao longo do caminho, mas é melhor usar uma caneta com seção transversal circular, de forma que a espessura fique independente do ângulo de inclinação do segmento.

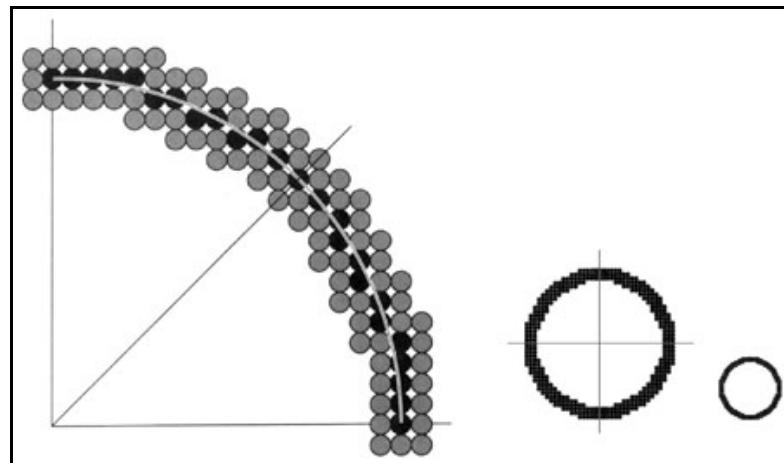


Figura 6.20 – Um círculo gerado pelo movimento de uma caneta retangular

6.11.3 PREENCHIMENTO DE ÁREA ENTRE DOIS LIMITES

O terceiro método para criar uma primitiva espessa é construir a primitiva interna à primitivas traças a $t/2$ de distância em ambos os lados do traçado da primitiva ideal.

Um método alternativo para definir a área da primitiva é deixar a primitiva original como fronteira externa e traçar uma fronteira a t de distância na parte interna. Esta alternativa tem a vantagem de funcionar para espessuras pares e ímpares e de não incrementar a extensão da primitiva. A desvantagem desta alternativa é que a área da primitiva aparenta contrair-se e que a linha central, a

linha base da primitiva, aparenta ter se deslocado. A Figura 6.21 mostra um círculo gerado através deste método.

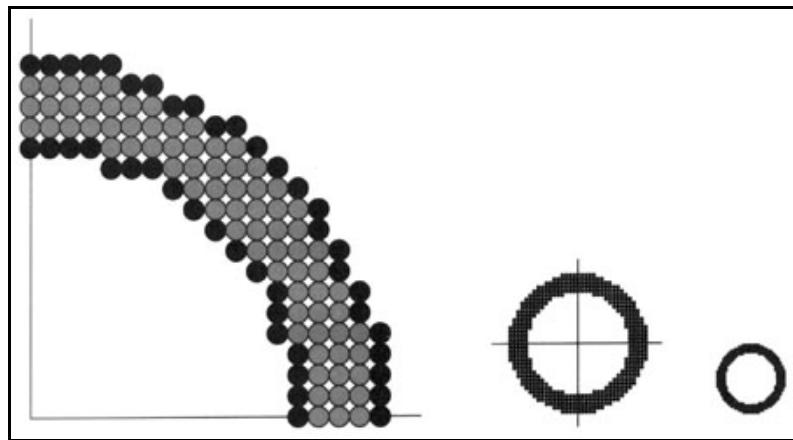


Figura 6.21 - Um círculo gerado pelo preenchimento de área entre dois limites

6.11.4 TIPOS DE LINHAS

As linhas podem ser sólidas, tracejadas e pontilhadas. Podemos modificar o algoritmo de conversão matricial de linhas para ajustar o comprimento e o espaçamento entre as seções sólidas da linha. Uma linha tracejada pode ser obtida gerando-se um espaço igual ao comprimento da seção sólida. Uma linha pontilhada pode ser exibida gerando-se segmentos muito pequenos cujo espaçamento é igual ou maior do que o próprio segmento. A Figura 6.22 mostra alguns tipos de linhas.

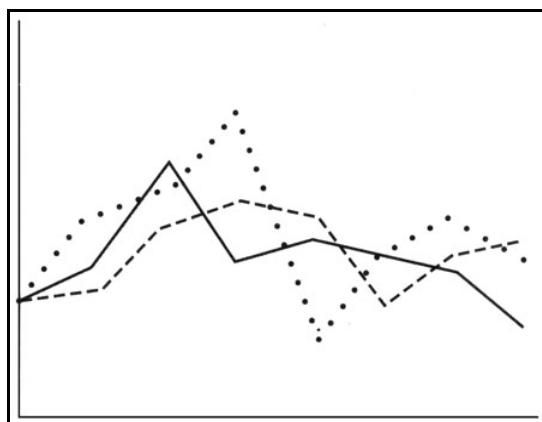


Figura 6.22 – Tipos de linhas. Pontilhadas, sólidas e tracejadas

Para os padrões tracejado, pontilhado e tracejado-pontilhado, os algoritmos de conversão matricial plotam blocos de pixels ignorando os pixels equivalentes aos espaços entre estes blocos. Os pixels plotados e os ignorados podem ser especificados numa **máscara de pixel**, que é uma string contendo os dígitos 1 e 0, indicando quais pontos devem ser plotados ou ignorados. A máscara 1111000, por exemplo, pode ser usada para exibir uma linha tracejada com segmentos de 4 pixels espaçados por 3 pixels.

6.12 CLIPPING

Qualquer procedimento que identifica partes de uma figura que são regiões de dentro ou fora de um espaço especificado, é chamado de **clipping algorithm** (algoritmo de recorte) ou simplesmente **clipping**. A região que recorta os objetos é chamada de **clip window**.

Aplicações do **clipping** incluem, por exemplo, extrair uma parte de uma cena para ser visualizada; identificar superfícies visíveis em visualizações tridimensionais, mostrar múltiplas janelas, etc.

Na visualização queremos mostrar somente as partes da figura que estão dentro da área da janela. Tudo o que está fora da janela é descartado.

Algoritmos de recorte podem ser aplicados em coordenadas do mundo, de maneira que somente o conteúdo do interior da janela é mapeado para coordenadas do dispositivo. Outra alternativa seria converter todas as coordenadas do mundo em coordenadas normalizadas ou do dispositivo e depois recortá-las através da *viewport*.

O recorte realizado em coordenadas do mundo remove as primitivas exteriores à janela de visualização, eliminando o processamento na conversão para o espaço do dispositivo. Por outro lado o recorte efetuado através da *viewport* pode reduzir o número de cálculos através da concatenação das matrizes de transformação geométrica e de visualização. Mas o recorte através da *viewport* requer que a transformação em coordenadas do dispositivo ocorra para todos os objetos, incluindo aqueles que estão fora da janela de visualização. Em sistemas raster, algoritmos de recorte estão frequentemente combinados com os algoritmos de conversão matricial.

6.12.1 POINT CLIPPING - RECORTE POR PONTOS

Assumimos que a janela de recorte é um retângulo na posição padrão. Armazenamos o ponto $P = (x, y)$ para exibição se as seguintes inequações são satisfeitas:

$$\begin{aligned} xW_{\min} &\leq x \leq xW_{\max} \\ yW_{\min} &\leq y \leq yW_{\max} \end{aligned}$$

onde os limites da janela de recorte ($xW_{\min}, xW_{\max}, yW_{\min}, yW_{\max}$) podem ser os limites da janela em coordenadas do mundo ou os limites da janela de visualização. Se qualquer uma destas inequações não for satisfeita o ponto é recortado.

Embora o recorte por pontos não seja tão eficiente, algumas aplicações requerem este procedimento. Por exemplo, em cenas que envolvem explosões ou ondas do mar modeladas com partículas (pontos) distribuídas em alguma região da cena.

6.12.2 LINE CLIPPING – RECORTE POR LINHAS OU RECORTE POR EQUAÇÕES SIMULTÂNEAS

A Figura 6.23 ilustra possíveis relações entre posições de linha e uma região de recorte retangular padrão.

O procedimento de recorte de linhas envolve várias etapas. Primeiro, nós

podemos testar um dado segmento de linha para determinar se os dois pontos extremos estão situados completamente dentro da janela de recorte. Se não estão, tentamos determinar quais segmentos estão completamente fora da janela. Finalmente, se não podemos identificar se a linha está completamente dentro ou fora, devemos fazer cálculos de interseção com uma ou mais fronteiras da janela de recorte.

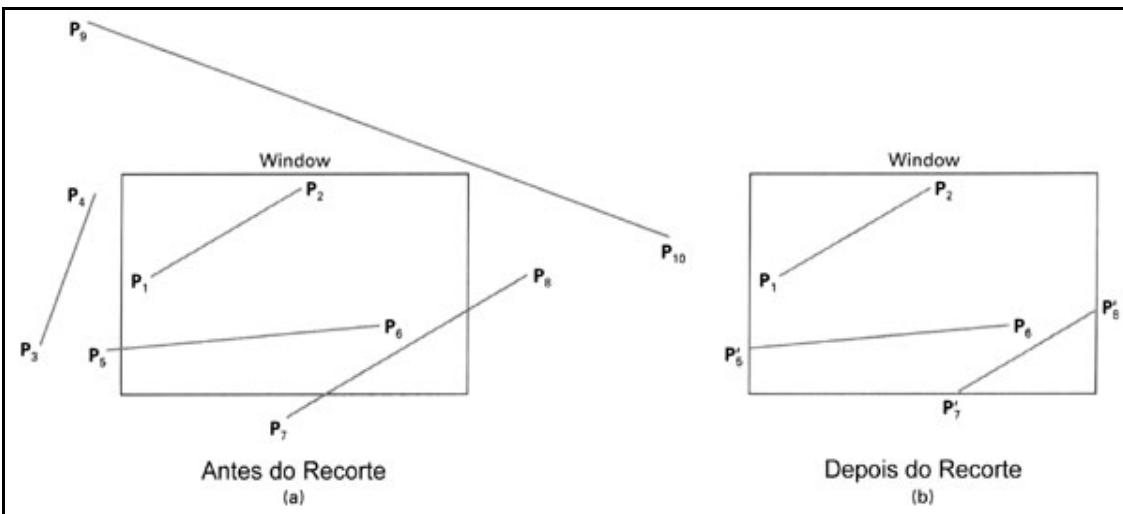


Figura 6.23 – Possíveis posições que segmentos de retas podem assumir em relação à janela de recorte retangular padrão

Processamos as linhas através do teste “dentro-fora”, checando os limites das linhas. Uma linha com ambos os pontos limites dentro das fronteiras de recorte tal como as linhas P1P2 é salva. A linha com ambos os pontos fora de uma das fronteiras de recorte, por exemplo P3P4, está fora da janela, ou seja, não será salva. Todas as outras linhas cruzam uma ou mais fronteiras da janela de recorte, e podem requerer cálculos de múltiplos pontos de interseção.

Para um segmento de linha com os pontos limites (x_1, y_1) e (x_2, y_2) e um ou ambos os pontos extremos fora do retângulo de recorte, a representação paramétrica

$$\begin{aligned} x &= x_1 + u(x_2 - x_1) \\ y &= y_1 + u(y_2 - y_1), \quad 0 \leq u \leq 1 \end{aligned}$$

pode ser utilizada para determinar o valor de u que proporciona a interseção do segmento com os limites da janela de recorte. Se o valor de u para uma interseção com as arestas do retângulo está fora do intervalo 0 até 1, a linha não entra no interior da janela por aquela fronteira. Se o valor de u está dentro da faixa de 0 a 1, o segmento de linha certamente adentra a área de recorte. Este método pode ser aplicado para cada fronteira da janela de recorte com o objetivo de determinar se alguma parte da linha deverá ser exibida. Segmentos de linhas que são paralelos a alguma das fronteiras são tratados como casos especiais.

• Implementação do Line Clipping

Primeiramente define-se a janela de recorte; se a janela de recorte for do mesmo tamanho da área de trabalho, não será feito nada, pois elas têm o mesmo tamanho e, certamente, todos os segmentos de reta estarão dentro da janela de recorte. Senão, com os limites da janela de recorte ($x_{\min}, x_{\max}, y_{\min}, y_{\max}$), faremos a verificação de todos os extremos dos segmentos, de forma a identificar quais

segmentos estão inteiramente dentro da janela de recorte.

Caso o segmento de reta for paralelo a janela de recorte, faremos o seguinte: verificamos qual dos pontos da reta não varia. Com este ponto verificaremos se ele está contido no intervalo de variação da janela de recorte. Após isso verificaremos se a variável que se altera também pertence ao intervalo da mesma janela de recorte e faremos os devidos ajustes.

Analisemos o exemplo: uma reta g com extremos em $P=(2, 2)$ e $Q=(2, 8)$ e uma janela de recorte com as coordenadas $(4, 2), (8, 2), (4, 8)$ e $(8, 8)$.

Neste caso a coordenada x não varia, mas não está dentro do limite de x da janela de recorte, pois $2 < 4$.

Caso $P=(6, 2)$ e $Q=(6, 10)$ verificamos que x não varia e está dentro da janela de recorte, então vamos verificar a variação de y . Constatamos que $P=(6, 2)$ pertence à janela, mas Q não, então criaremos um novo Q' , da janela de recorte, que seria $Q'=(6, 8)$.

Acontecendo casos em que alguns pontos do segmento de reta obedecem a essas restrições e outros não, será chamado o procedimento de recorte de linhas por equações simultâneas. Como será seu funcionamento: serão usadas as equações paramétricas ($x = x_1 + u(x_2 - x_1)$ e $y = y_1 + u(y_2 - y_1)$, onde $0 \leq u \leq 1$) e os limites dos segmentos de reta. Para cada ponto limite do segmento de reta, será feita uma verificação com cada aresta da janela de recorte. Se nessa verificação o valor de u estiver entre 0 e 1, este ponto está dentro da área de recorte. Quando o valor não estiver dentro deste intervalo, ele não pertence à janela, então se fará um corte no segmento de reta, através do cálculo da interseção do segmento com a aresta. Após isto o novo ponto será gravado na lista de pontos da janela de recorte.

6.12.3 ALGORITMO DE COHEN-SUTHERLAND

O algoritmo de recorte de Cohen-Sutherland rapidamente detecta dois casos comuns e triviais: o caso de linhas inteiramente dentro da área de recorte e o caso de linhas inteiramente fora da área de recorte.

Para recortar uma linha precisamos considerar somente seus pontos finais. Se ambos estão dentro da janela, a linha inteira está dentro da janela e não precisa de recorte. Por outro lado, se ambos os pontos finais de uma linha estiverem em um dos lados da janela, a linha deverá estar inteiramente fora da janela e não precisará ser recortada nem mostrada.

Para determinar quais pontos extremos estão dentro ou fora de uma janela, o algoritmo seta um código para cada ponto final. Cada limite da janela define uma linha infinita que divide todo o espaço em dois meio-espacos: o meio-espaco de dentro e o meio-espaco de fora da janela de recorte, conforme mostra a Figura 6.24.

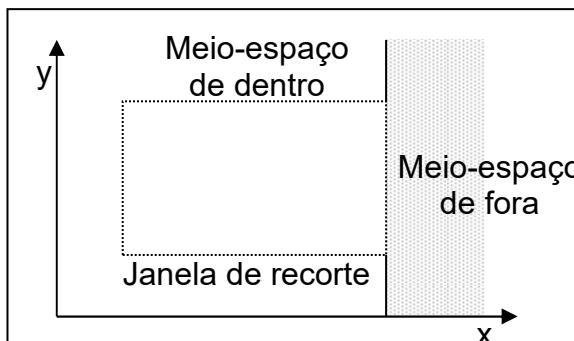


Figura 6.24 – Divisão do espaço em dois meio-espacos

Prosseguindo ao redor da janela, estendendo cada limite e definindo um meio de dentro e um meio de fora, nove regiões são criadas: oito regiões externas à janela de recorte um uma região interna.

Para cada uma das nove regiões associadas com a janela é associado um código de 4 bits para identificar a região. Cada bit do código é setado para 1 (verdadeiro) ou 0 (falso). Se a região está à esquerda da janela de recorte, o primeiro bit é setado para 1. Se está à direita o segundo bit é setado para 1. Se está abaixo o terceiro bit é setado para 1 e se está acima o quarto bit é setado para 1. O primeiro bit é o mais à direita e o quarto bit é o mais à esquerda. A Figura 6.25 mostra os códigos associados à cada uma das regiões.

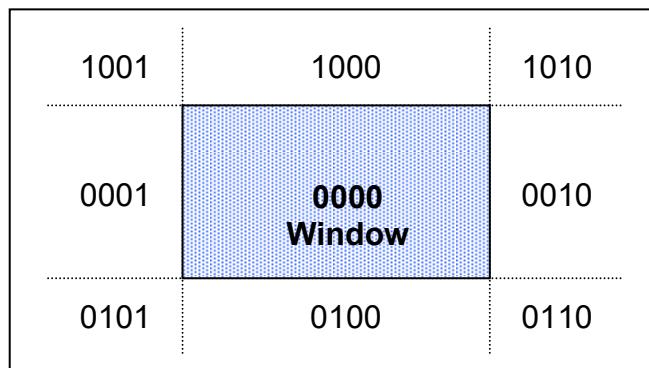


Figura 6.25 – Codificação binária das 9 regiões associadas com a janela de recorte

Para qualquer ponto extremo (x, y) de uma linha, pode ser determinado o código que identifica em que região o ponto extremo está. Os bits são setados conforme as seguintes condições:

- Primeiro bit setado para 1: o ponto está à esquerda da janela $\Rightarrow x < x_{\min}$;
- Segundo bit setado para 1: o ponto está à direita da janela $\Rightarrow x > x_{\max}$;
- Terceiro bit setado para 1: o ponto está abaixo da janela $\Rightarrow y > y_{\max}$;
- Quarto bit setado para 1: o ponto está acima da janela $\Rightarrow y < y_{\min}$.

A sequência para leitura dos códigos é TBRL (Top, Bottom, Right, Left).

Uma vez determinados os códigos para cada extremo da linha, a operação lógica AND dos códigos determina se a linha está completamente fora da janela. Se o AND lógico dos códigos dos extremos não for zero, a linha pode ser trivialmente rejeitada. Por exemplo, se um extremo tem um código de 1001 enquanto o outro tem um código de 1010, o AND lógico será 1000 que significa que o segmento de linha está fora da janela. Por outro lado, se os extremos tivessem códigos 1001 e 0110, o AND lógico seria 0000, e a linha não poderia ser trivialmente rejeitada.

O OU lógico dos códigos dos extremos determina se a linha está completamente dentro da janela. Se o OU lógico é zero, a linha pode ser trivialmente aceita. Por exemplo, se os códigos dos extremos são 0000 e 0000, o OU lógico é 0000 e a linha pode ser trivialmente aceita. Se os códigos são 0000 e 0110, o OU lógico é 0110 e a linha não pode ser trivialmente aceita.

• Implementação do Algoritmo de Cohen-Sutherland

O algoritmo de Cohen-Sutherland usa uma estratégia de divisão-e-conquista. Os pontos extremos dos segmentos de linha são testados para ver se a

linha pode ser trivialmente aceita ou rejeitada. Se a linha não pode ser trivialmente aceita ou rejeitada, uma interseção da linha com o limite da janela é determinado e o teste de rejeição/aceitação é repetido. Este processo continua até que a linha seja aceita.

Para realizar o teste de aceitação trivial ou rejeição, estendemos os limites da janela para dividir o plano da janela em nove regiões. Cada extremo dos segmentos de linha é associado o código da região em que ele se situa.

- 1 – Dado um segmento de linha com ponto final $P=(x_1, y_1)$ e $Q=(x_2, y_2)$.
- 2 – Calculamos os códigos de 4 bits para cada ponto.

Se ambos os códigos são 0000, (o OU lógico dos códigos retorna 0000) a linha está completamente dentro da janela: passa para a rotina de pontos extremos para a rotina de desenho.

Se ambos os códigos tem 1 na mesma posição do bit (AND lógico dos códigos não é zero), a linha está fora da janela. E pode ser trivialmente rejeitada.

3 – Se a linha não pode ser trivialmente rejeitada ou aceitada, pelo menos um dos extremos deve estar fora da janela e o segmento de linha cruza um limite da janela. Esta linha deve ser recortada aos limites da janela depois passada para a rotina de desenho.

4 – Examine um dos extremos, digamos $P=(x_1, y_1)$. Ler os 4 bits de código de P na ordem: Esquerda, Direita, Abaixo e Acima.

5 – Quando um bit setado (1) é encontrado, calculamos a interseção I do limite correspondente da janela com a linha de P para PQ . Substituímos P por I e repetimos o algoritmo.

Para ilustrar como funciona o algoritmo de Cohen-Sutherland mostraremos como as linhas da Figura 6.26 podem ser processadas.

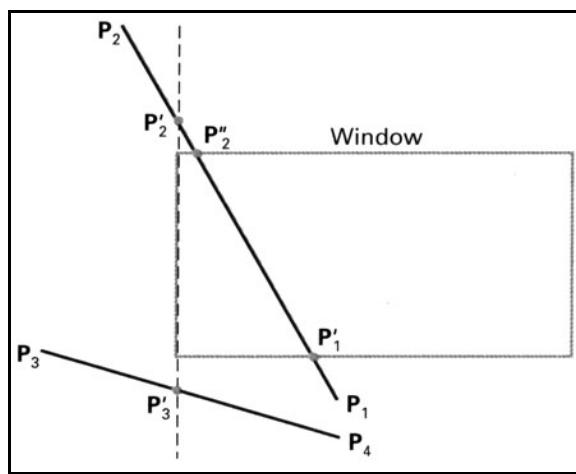


Figura 6.26 – Utilização do algoritmo de Cohen-Sutherland

Iniciando com o extremo inferior da linha de P_1 até P_2 , checamos P_1 contra as fronteiras à esquerda, direita, e abaixo até detectar que o ponto fica abaixo do retângulo de recorte. Então encontramos a o ponto interseção P'_1 com a fronteira inferior e descartamos a seção da linha de P_1 até P'_1 . A linha foi reduzida para a seção entre P'_1 e P_2 . Uma vez que P_2 está fora do retângulo de recorte, checaremos este extremo contra as fronteiras até encontrar àquela à esquerda do retângulo. O ponto de interseção P'_2 é calculado, mas este ponto está acima do retângulo. Então

a interseção final P''_2 é calculada e a linha de P'_1 até P''_2 é salva. Isto completa o processamento para esta linha e, então, salvamos o segmento e prosseguimos para a próxima linha. O ponto P_3 na próxima linha está à esquerda do retângulo de corte, então determinamos a interseção P'_3 e eliminamos a seção da linha de P_3 até P'_3 . Checando o código da região para a linha de P'_3 até P_4 , encontramos que o restante desta linha está abaixo da janela de recorte e então podemos descartá-la.

6.12.4 RECORTE DE POLÍGONOS

O recorte de polígonos utiliza, com algumas alterações, os algoritmos vistos anteriormente. Um polígono processado por um algoritmo de recorte de linhas, dependendo da orientação do polígono em relação à janela de recorte, resulta numa série de segmentos de linhas não conectadas (Figura 6.27).

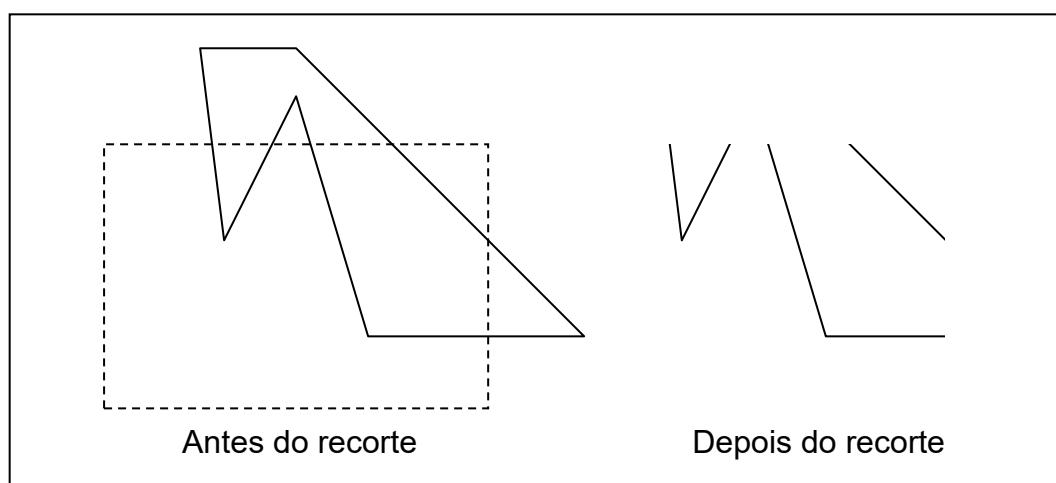


Figura 6.27 – Polígono processado por um algoritmo de recorte de linhas

Estes algoritmos deveriam fornecer como saída regiões fechadas, como as mostradas na Figura 6.28. Posteriormente estas regiões poderão ser preenchidas através de um algoritmo adequado.

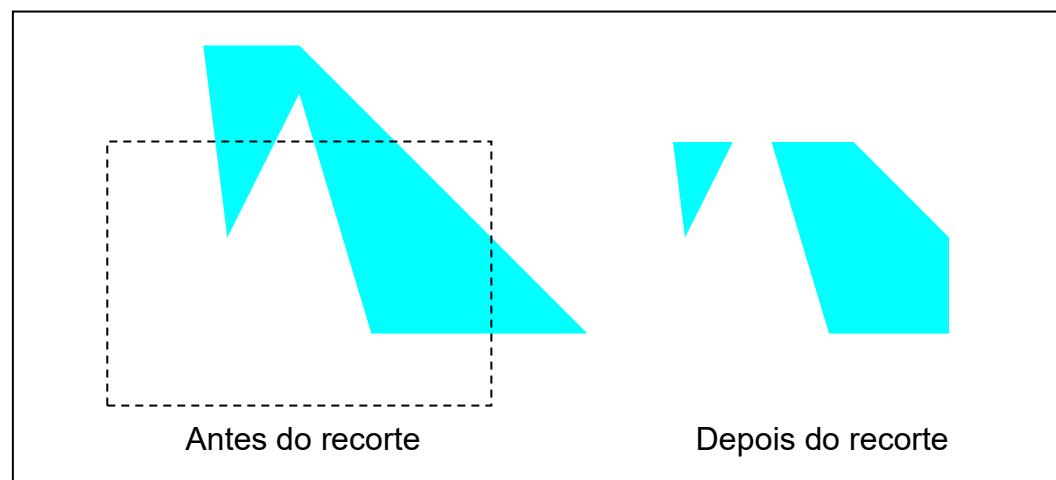


Figura 6.28 – Exibição correta de um polígono recortado

6.12.5 RECORTE DE POLÍGONOS – SUTHERLAND-HODGEMAN

O recorte de polígonos pode ser corretamente realizado recortando todas as arestas do polígono contra cada borda da janela de recorte. Para tanto, percorre-se circularmente os vértices do polígono recortando suas arestas contra cada aresta da janela.

Inicia-se o processo com o conjunto de vértices do polígono. Estes vértices são então recortados contra a aresta da esquerda da janela, produzindo um novo conjunto de vértices. Posteriormente este novo conjunto de vértices é recortado contra as arestas da direita, inferior e superior da janela, como mostra a Figura 6.29. A cada passo, um novo conjunto de vértices é gerado e repassado para o recorte contra a próxima aresta da janela.

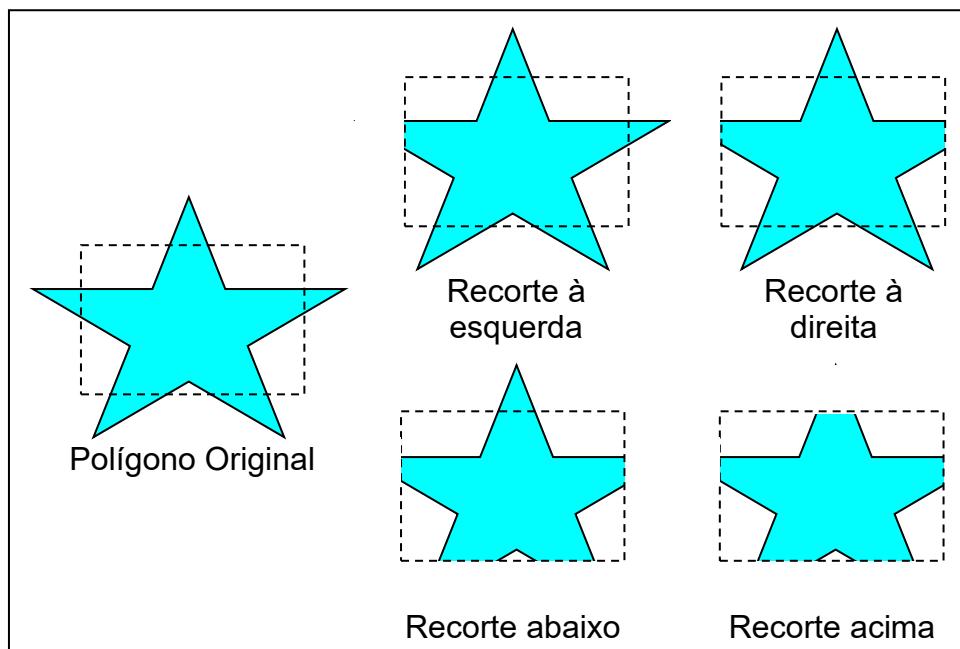


Figura 6.29 – Recortando o polígono contra as sucessivas arestas da janela

Há 4 possíveis casos quando processamos os vértices ao redor do perímetro de um polígono. Para cada par de vértices adjacentes do polígono realizam-se os seguintes testes:

- 1) Se o primeiro vértice está fora da janela de recorte e o segundo está dentro, então o vértice da interseção e o vértice que está dentro do polígono são inseridos na lista de vértices de saída;
- 2) Se ambos os vértices estão dentro da janela de recorte então apenas o segundo vértice é adicionado à lista de saída;
- 3) Se o primeiro vértice está dentro e o segundo está fora da janela de recorte, então apenas o vértice da interseção é adicionado à lista de saída;
- 4) Se ambos os vértices estão fora da janela de recorte então nenhum vértice é adicionado à lista de saída.

Estes 4 casos estão ilustrados na Figura 6.30. Uma vez que todos os vértices foram processados contra uma das arestas da janela de recorte, a lista de vértices de saída é recortada contra a próxima aresta da janela.

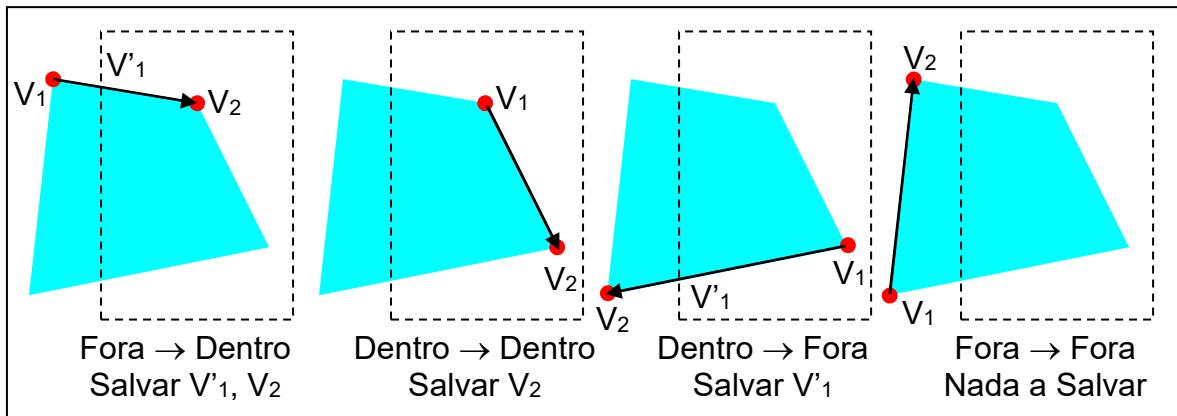


Figura 6.30 – Processamento sucessivo de pares de vértices do polígono contra a aresta esquerda da janela de recorte

Podemos visualizar melhor este método processando a área da Figura 6.31 contra a aresta esquerda da janela. Os vértices 1 e 2 são desconsiderados, pois estão fora da janela. Movendo-se até o vértice 3, que está dentro da janela, calculamos o vértice na interseção salvando ambos os vértices na lista de saída. Os vértices 4 e 5 estão dentro da janela e também são salvos na lista. O sexto e último vértice está fora da janela, então calculamos o vértice da interseção e o salvamos na lista de saída. Usando os cinco pontos salvos podemos repetir o processo para a próxima aresta da janela.

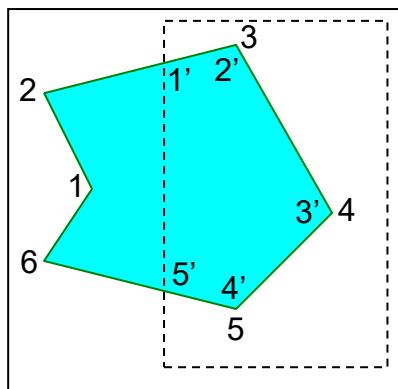


Figura 6.31 – Recortando um polígono contra a aresta da esquerda da janela de recorte

Polígonos convexos são recortados corretamente pelo algoritmo de Sutherland-Hodgman, mas em polígonos côncavos podem surgir linhas estranhas, como demonstra a Figura 6.32. Isto ocorre quando o polígono recortado tem duas ou mais seções separadas. Como há apenas uma lista de vértices de saída, o último vértice da lista é sempre conectado ao primeiro vértice. Existem várias ações que poderíamos tomar para exibir corretamente polígonos côncavos. Uma delas seria dividir o polígono côncavo em dois ou mais polígonos convexos e processá-los separadamente. Outra possibilidade é modificar o algoritmo de Sutherland-Hodgeman para verificar a lista final de vértices e ligar os pares de vértices corretamente. A solução final é utilizar um algoritmo de recorte de polígonos mais geral, como o algoritmo de Weiler-Atherton.

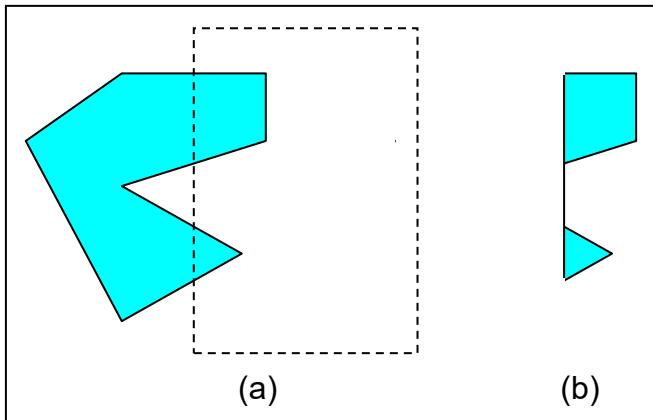


Figura 6.32 – Recortando um polígono côncavo (a) com o algoritmo de Sutherland-Hodgeman. Saída produzida pelo algoritmo (b)

6.12.6 RECORTE DE POLÍGONOS – WEILER-ATHERTON

Este algoritmo foi desenvolvido como um método para identificar superfícies visíveis, mas também pode ser aplicado para recortar polígonos de formas arbitrárias.

A ideia básica deste algoritmo é que nem sempre percorreremos o perímetro do polígono para realizar o recorte; às vezes teremos que percorrer as fronteiras da região de recorte. O caminho a seguir depende da direção de processamento do polígono (horária ou anti-horária) e se o par de vértices do polígono que está sendo processado no momento representa um par fora-dentro ou um para dentro-fora. Para o sentido horário de processamento dos vértices do polígono, usamos as seguintes regras:

- para um par de vértices fora-dentro, siga o perímetro do polígono;
- para um par de vértices dentro-fora, siga o perímetro da região de recorte no sentido horário.

Na Figura 6.33, a direção de processamento no algoritmo de Weiler-Atherton e o polígono recortado são exibidos de acordo com uma região de recorte retangular.

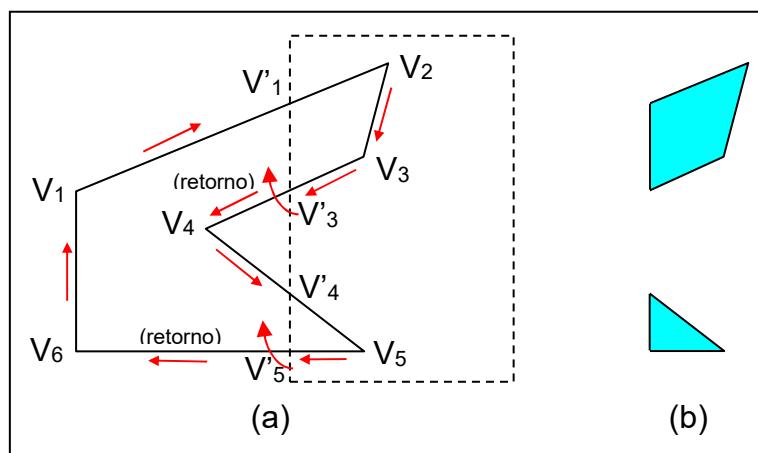


Figura 6.33 – Recorte de um polígono côncavo (a) como o algoritmo de Weiler-Atherton gera as duas áreas poligonais apresentadas em (b)

Uma melhoria do algoritmo de Weiler-Atherton é o algoritmo de Weiler, que permite recortar um polígono arbitrário contra qualquer região de recorte poligonal. A Figura 6.34 mostra uma ideia geral deste algoritmo. Para os dois polígonos desta figura, o polígono de saída é a interseção entre o polígono original e o polígono que define a região de recorte.

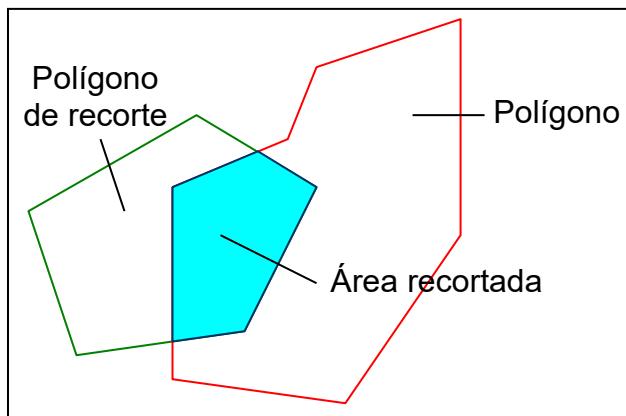


Figura 6.34 – Recorte de um polígono pela determinação da interseção das duas áreas poligonais

6.13 TESTES DENTRO-FORÁ

Algoritmos de preenchimento de polígonos e outros processos gráficos frequentemente necessitam identificar a região interna dos objetos. Polígonos convencionais, como o triângulo, o hexágono e outros, não interceptam a si mesmo; as arestas destes objetos unem-se apenas nos vértices. Identificar as regiões interiores deste tipo de polígono é, geralmente, um processo direto. Mas em muitas aplicações gráficas, nós podemos determinar qualquer sequência de vértices para um polígono, incluindo sequências que produzem arestas que se interceptam, como mostrado na Figura 6.35. Para estas formas, não é sempre fácil determinar quais regiões do plano xy podemos chamar de “interiores” e quais regiões podemos chamar de “exteriores” ao polígono. Pacotes gráficos usam a regra da **paridade ímpar** ou a regra do **número não nulo de voltas**.

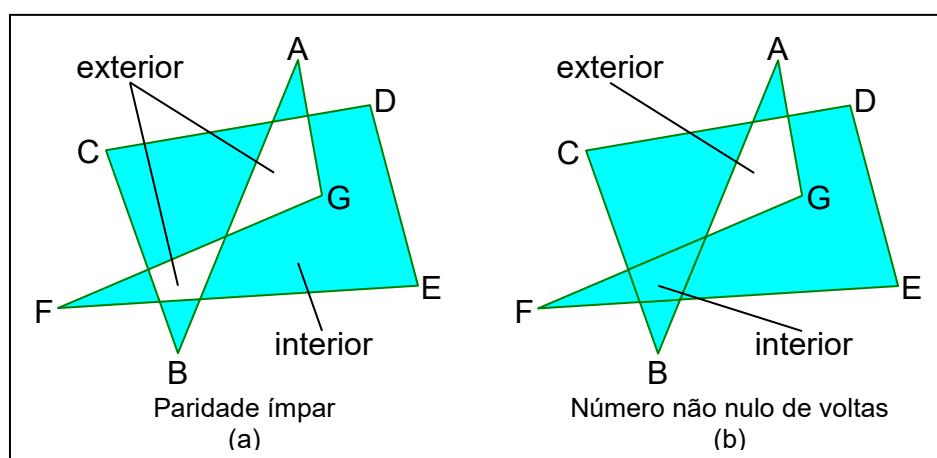


Figura 6.35 – Identificando regiões internas e externas de um polígono complexo

A regra de paridade ímpar consiste em desenhar uma linha a partir de um ponto **P** qualquer até um ponto distante do polígono e contar quantas arestas do

polígono são interceptadas por esta linha. Se o número de arestas interceptadas pela linha for ímpar, então o ponto P está no interior do polígono; se este número for par o ponto P está no exterior do polígono. Para obtermos um resultado correto a linha traçada não pode passar por nenhum dos vértices do polígono analisado. A Figura 6.35(a) mostra as regiões interiores e exteriores, obtidas através da regra da paridade ímpar, para um polígono que se autointercepta. O método de preenchimento de polígonos discutido anteriormente é um exemplo de algoritmo que utiliza a regra da paridade ímpar.

Outro método para definir as regiões interiores é a regra do número não nulo de voltas, que conta o número de vezes que as arestas do polígono giram ao redor de um ponto em particular na direção anti-horária. Este contador é chamado de número de voltas, e um ponto interior de um objeto bidimensional possui um valor de número de voltas não nulo. Para aplicar este algoritmo inicializamos o contador com 0 e, novamente, imaginamos uma linha desenhada de um ponto P qualquer até um ponto distante do polígono. Esta linha não pode passar por nenhum vértice do polígono. Quando nos movemos ao longo desta linha a partir do ponto P até o ponto distante, contamos o número de arestas que cruzam a linha em cada direção. Adicionamos 1 ao contador sempre que interceptamos uma aresta do polígono que cruza a linha da direita para a esquerda, e subtraímos 1 sempre que interceptamos uma aresta que cruza a linha da esquerda para a direita. O valor final do contador, depois de contar todas as arestas cruzadas, determina a posição relativa de P . Se o número de voltas é não nulo, P é definido como um ponto interior. Por outro lado, P é definido como exterior. A Figura 6.35(b) mostra as regiões interiores e exteriores definidas pela regra do número não nulo de voltas.

Para polígonos simples ambas as regras apresentam os mesmos resultados. Mas para figuras complexas, os dois métodos podem produzir resultados distintos, como na Figura 6.35.

Uma forma de determinar a direção do cruzamento das arestas é calcular o produto vetorial entre o vetor u , definido pela linha entre P e o ponto afastado, e o vetor E para cada uma das arestas cruzadas pela linha. Se a componente z do produto vetorial $u \times E$ para uma aresta é positivo, esta aresta cruza a linha da direita para a esquerda e adicionamos 1 ao contador de voltas. Se a componente for negativa, a aresta cruza a linha da esquerda para a direita e subtraímos 1 do contador de voltas. O vetor E é calculado subtraindo o vértice inicial do vértice final da aresta. Por exemplo, o vetor E sobre a primeira aresta da Figura 6.35 é obtido por

$$E_{AB} = V_B - V_A$$

onde V_A e V_B representam os vértices A e B do polígono (no formato (x, y, z)).

Uma forma mais simples de computar a direção de cruzamento é utilizar o produto escalar entre dois vetores ao invés do produto vetorial. Para fazer isto, determinamos um vetor perpendicular a u . Se o vetor u é (u_x, u_y) , então o vetor perpendicular a u possui componentes $(-u_y, u_x)$. Assim, se o produto escalar entre o vetor perpendicular e um vetor sobre a aresta considerada é positivo, então esta aresta cruza a linha da direita para a esquerda e adicionamos 1 ao contador de voltas. Caso contrário, a aresta cruza a linha da esquerda para a direita e subtraímos 1 do contador de voltas.

UNIDADE 7 – CORES E SISTEMAS DE CORES

7.1 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 frequências do espectro eletromagnético, que inclui as ondas de rádio, micro-ondas, os raios infravermelhos e os raios X, como mostrado na Figura 7.1.

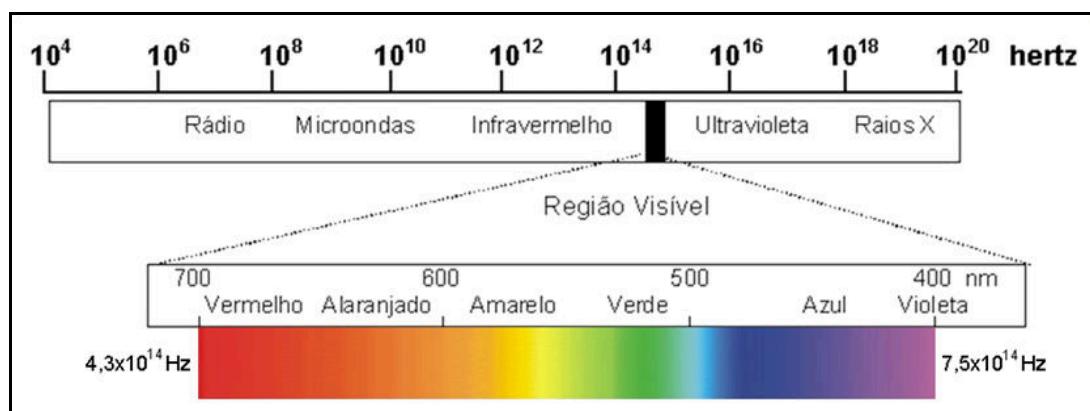


Figura 7.1 – Espectro eletromagnético

A frequência mais baixa do espectro visível corresponde à cor vermelha (4.3×10^{14} hertz) e a mais alta à cor violeta (7.5×10^{14} hertz). Os valores de frequência intermediários correspondem a cores que passam pelo alaranjado e amarelo e por todas as outras cores, até chegar aos verdes e azuis.

As cores são ondas eletromagnéticas descritas pelo seu comprimento de onda (λ) 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 faz com que a lente altere o seu formato constantemente, causando cansaço no olho.

Uma fonte de luz (como, por exemplo, o sol ou uma lâmpada) emite em todas as frequências 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 frequências baixas, diz-se que o objeto é vermelho, ou que a luz percebida possui uma frequência dominante (ou um comprimento de onda dominante) na frequência baixa do espectro. A frequência 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: vermelho, amarelo, laranja, azul ou púrpura. 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.

Um aspecto que 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, por exemplo, menos sensíveis à cor amarela do que à cor azul.

7.2 SISTEMAS DE CORES PRIMÁRIAS

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 em 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 (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 7.2 ilustra a demonstração do funcionamento desses modelos através da sobreposição de círculos coloridos.

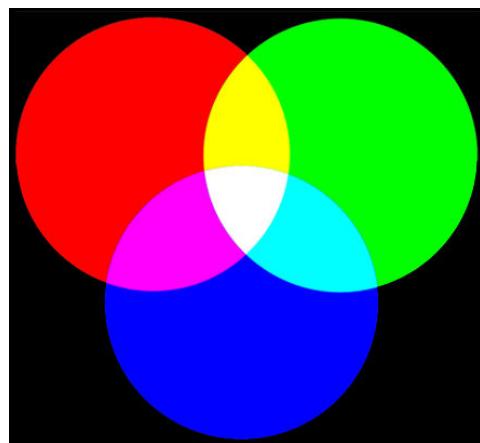


Figura 7.2 – Ilustração da mistura de cores aditivas

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 ou ciano 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 7.3 ilustra a demonstração do funcionamento desses modelos através da sobreposição de círculos coloridos.

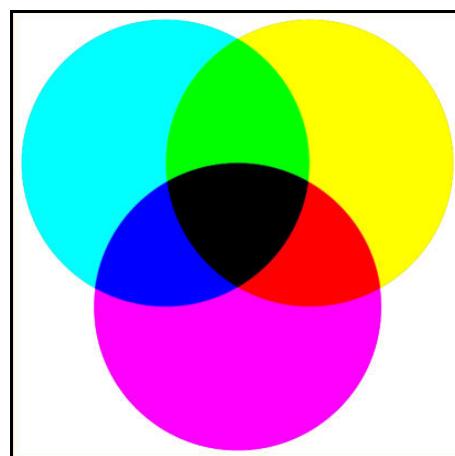


Figura 7.3 – Ilustração da mistura de cores subtrativas

Pode-se pensar que o preto é a combinação das 3 cores subtrativas (turquesa ou ciano, 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.

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 7.3 ilustra a obtenção de *tints*, *shades* e *tones* obtidos a partir da cor vermelha.

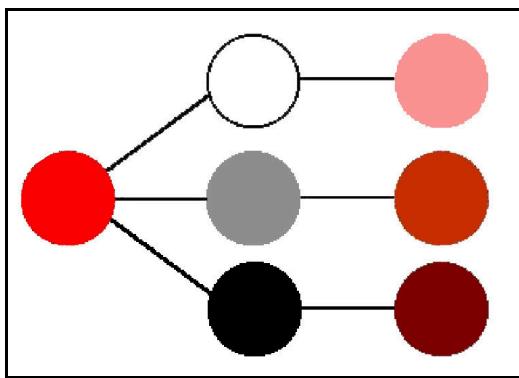


Figura 7.4 – Ilustração da obtenção de *tints*, *shades* e *tones*

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 – marrom). 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.

7.3 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 $C\lambda$ podem ser expressas pela seguinte equação:

$$C\lambda = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$$

em que X, Y e Z especificam as quantidades das primárias padrão 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 pelas quantidades normalizadas abaixo:

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

com $x + y + z = 1$.

Assim, qualquer cor pode ser definida através 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 (x e y) 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 através das equações a seguir:

$$X = \frac{x}{y} Y \quad Z = \frac{z}{y} Y, \text{ onde } z = 1 - x - y$$

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.

Através das coordenadas de cromaticidade x e y é possível representar todas as cores num gráfico bidimensional. O traçado das quantidades de x e y normalizadas para as cores no espectro visível resulta na curva ilustrada na Figura 7.5.

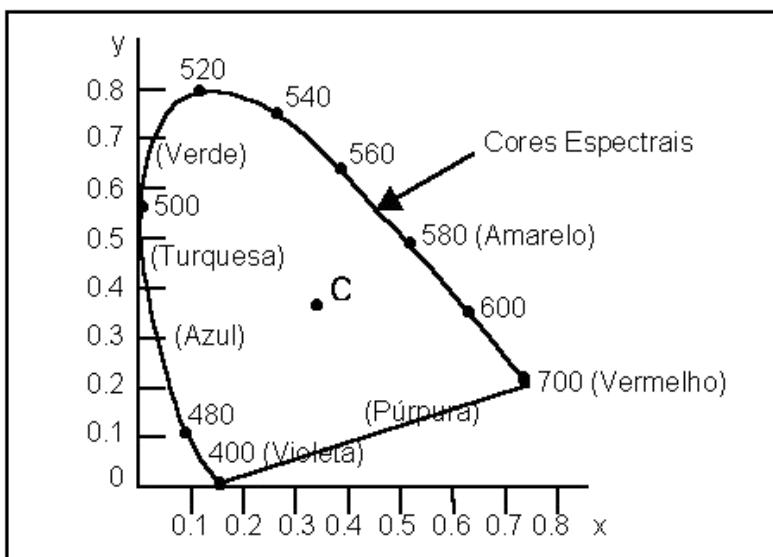


Figura 7.5 – Diagrama de cromaticidade do CIE

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 correspondentes ao vermelho e ao 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 7.6a, através de linhas retas ou de polígonos. Todas as cores ao longo da linha que une os pontos C1 e C2 na Figura 7.6a podem ser obtidas através da mistura de quantidades apropriadas das cores correspondentes a esses pontos.

A escala de cores para 3 pontos (por exemplo, C3, C4 e C5 na Figura 7.6a) é representada por um triângulo cujos vértices são definidos pelas cores correspondentes às 3 posições e inclui cores contidas no interior e nas margens fronteiricas desse triângulo.

Através do 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.

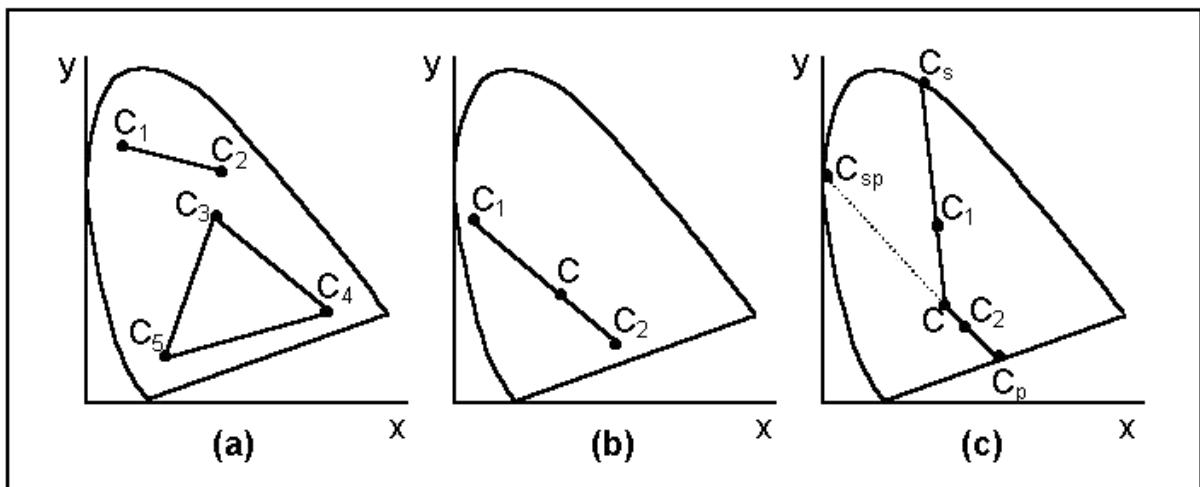


Figura 7.6 - Representação de escalas de cor no diagrama de cromaticidade do CIE

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 7.6b), obtém-se a luz branca.

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 7.6c, 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 7.6c) 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 espetrais 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 7.6c) 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 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 aproximadamente 25% pura porque está situada a aproximadamente $\frac{1}{4}$ da distância total entre C e C_s .

7.4 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.

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 7.7.

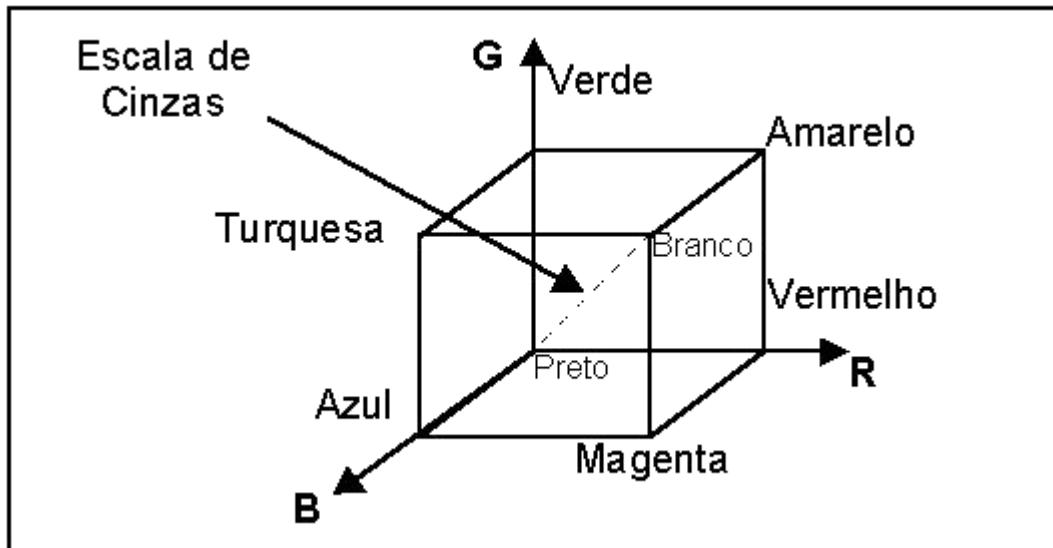


Figura 7.7 – Cubo do RGB

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 à 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 $C\lambda$ desse sistema podem ser expressas na forma:

$$C\lambda = rR + gG + bB$$

A resposta do olho aos estímulos espectrais 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.

7.5 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 7.8.

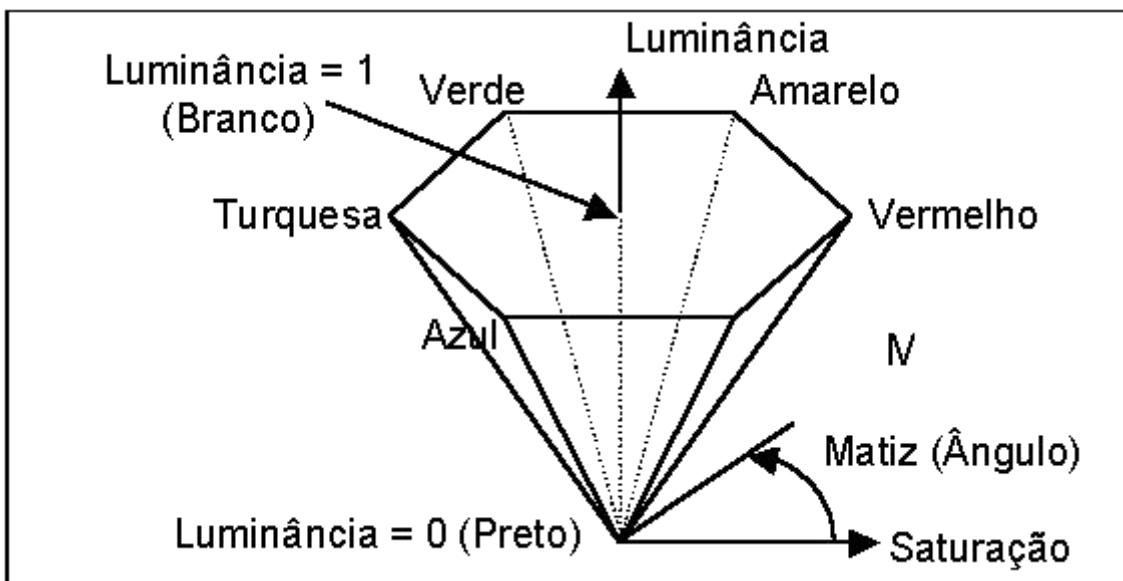


Figura 7.8 - Cone hexagonal do HSV

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° (vermelho) a 360° , e o ângulo entre os vértices é de 60° .

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.

7.6 MODELO HLS (HUE, LIGHTNESS, SATURATION)

O sistema HLS 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 7.9. Os 3 parâmetros de cor utilizados são o matiz (*hue*), a luminosidade (*lightness*) e a saturação (*saturation*).

O ângulo em relação ao eixo vertical varia de 0° (matiz azul) a 360° em intervalos de 60° 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 "escuras".

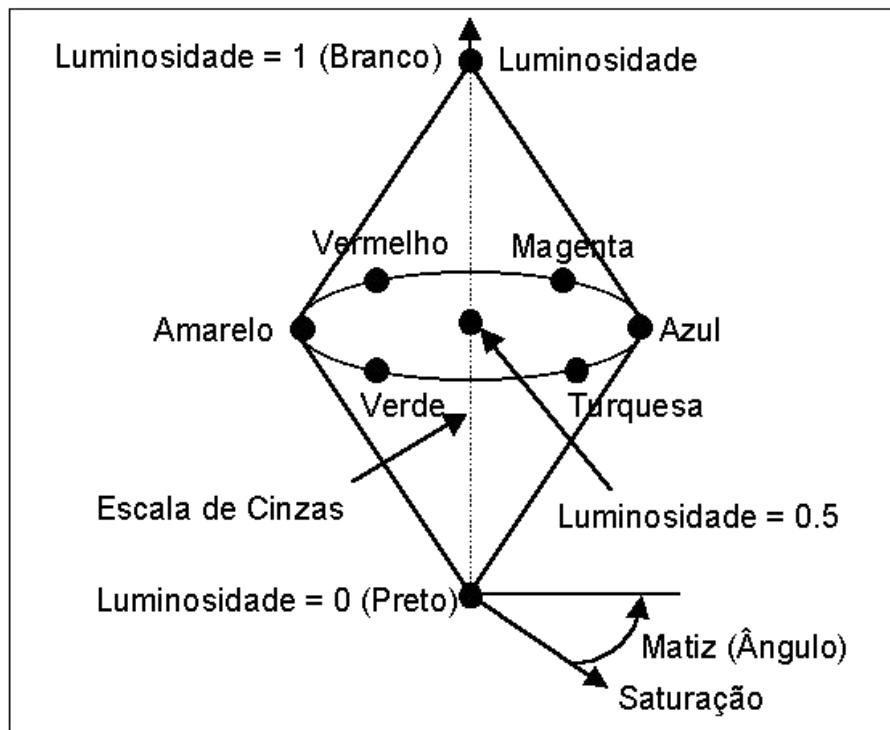


Figura 7.9 - Cone duplo do HLS

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 escuras pela diminuição dos mesmos. As cores intermediárias são obtidas através da diminuição da saturação.

7.7 USO DE CORES NA COMPUTAÇÃO GRÁFICA

As cores podem ser usadas em inúmeros fins como: estéticos, estabelecimento de um aspecto ou humor, realismo, seleção de texto ou imagem, para associar áreas e codificação. Com cuidado as cores podem ser utilizadas para todos estes fins.

Os usuários tendem a gostar das cores, mesmo sem evidências quantitativas de que elas aumentam seu rendimento.

O uso cuidadoso das cores pode tornar um monitor mais ou menos atrativo que um monitor monocromático. Experimentos realizados por KREBS & WOLF (1979) mostraram que o uso de cores insignificantes reduz o rendimento do usuário em até 1/3. Qualquer uso decorativo de cores deve ser subserviente à funcionalidade. As cores não podem ser erroneamente interpretadas ou terem significados dúbios. Portanto, o uso das cores deve ser testado por usuários reais para identificar e corrigir os problemas.

O olho humano é mais sensível à variação espacial na intensidade do que na cromaticidade, ou seja, textos, linhas e outros detalhes da imagem devem diferenciar da cor de fundo em brilho e não somente em matiz.

O olho humano não é capaz de distinguir as cores em objetos muito pequenos, portanto o uso da codificação em cores não deve ser utilizada em pequenos objetos. A percepção das cores de uma região é afetada pelas cores em

sua vizinhança e isto pode trazer sérios problemas quando as cores são utilizadas em codificação de informações.

Se colorirmos uma área com uma cor altamente saturada e a observarmos por alguns segundos e então olharmos para outro local, uma imagem posterior desta grande área aparecerá. Portanto, o uso de grandes áreas com cores saturadas não é inteligente.

Da mesma forma, grandes áreas em cores diferentes podem parecer estar a distâncias diferentes do usuário devido ao índice de refração da luz depender do comprimento de onda. O azul e o vermelho, por estarem em lados opostos no espectro tem um elevado efeito de disparidade na profundidade, com o vermelho aparentando estar mais próximo do usuário e o azul mais distante. Assim, o uso simultâneo do azul para a cor dos objetos e do vermelho para cor de fundo não é adequado.

UNIDADE 8 – ILUMINAÇÃO E SOMBREAMENTO

8.1 ILUMINAÇÃO

Em computação gráfica a manipulação da luz assume um papel fundamental no aspecto realístico da apresentação. Os efeitos da luz sobre as superfícies e seus materiais, o obscurecimento de superfícies em função de sua posição, orientação e características da luz são, portanto, peças chave. Existem vários modelos de iluminação diferentes que expressam e controlam os fatores que determinam a cor de uma superfície, em função de um determinado conjunto de luzes. Alguns modelos de iluminação controlam o efeito da luz para cada pixel na imagem enquanto outros consideram um modelo de iluminação para alguns poucos pixels e aplicam interpoladores para o cálculo dos demais.

A computação gráfica refere-se frequentemente às regras da ótica e à física das radiações para explicar a interação da luz nos objetos. No entanto a complexidade ou a inexistência de modelos completos e abrangentes levam os programadores a simplificações do processo computacional. Por consequência, muitos modelos de iluminação, em uso na computação gráfica, contém simplificações sem nenhum fundamento teórico, mas que realizam um bom trabalho prático.

8.1.1 LUZ AMBIENTE

Talvez o modelo de iluminação mais simples possível é aquele em que cada objeto usa um valor intrínseco de cor quando é exibido. Este valor intrínseco é fruto de reflexões múltiplas da luz nas muitas superfícies presentes no ambiente. Este tipo de fenômeno é definido como luz ambiente. Ao assumir-se que a luz ambiente afeta igualmente todas as superfícies em todas as direções, pode-se definir luz ambiente como:

$$I_a = I_{la} \cdot K_a$$

Onde I_{la} é a intensidade da luz ambiente, assumida como constante para todos os objetos. A quantidade de luz ambiente refletida pela superfície de um objeto é determinada através de K_a , ou coeficiente de reflexão ambiente, que varia de 0 a 1. O coeficiente de reflexão ambiente é uma propriedade do material que compõe a superfície do objeto.

Pode-se imaginar este modelo como não contendo nenhuma fonte de luz externa e descrevendo um mundo irreal onde cada objeto possui luz própria. O coeficiente de reflexão ambiente é uma conveniência empírica e não corresponde diretamente a qualquer propriedade física real dos materiais. Além disso, luz ambiente não é por si só de muito interesse como veremos a seguir.

8.1.2 REFLEXÃO DIFUSA

A maioria dos objetos que vemos ao nosso redor não emite luz própria, eles absorvem parte da luz de uma fonte e refletem outra parte. A reflexão se deve à

interação molecular entre a luz incidente e o material que compõe a superfície. Uma superfície é definida como um difusor perfeito se ela é capaz de refletir a luz igualmente em todas as direções. Isto significa que a quantidade de luz refletida percebida não depende da posição do observador.

Embora os objetos iluminados pela luz ambiente sejam mais ou menos brilhantes em proporção direta à intensidade desta luz, eles ainda são iluminados uniformemente ao longo de toda a sua superfície.

Considere-se um objeto iluminado por uma fonte pontual de luz cujos raios emanam uniformemente em todas as direções. O brilho de um objeto iluminado varia ao longo de sua superfície em função do seu ângulo e distância em relação à fonte de luz.

Superfícies foscas apresentam uma reflexão difusa, também conhecida como reflexão Lambertiana. Superfícies Lambertianas apresentam brilho uniforme em todos os ângulos de observação, pois a reflexão da luz ocorre uniformemente em todas as direções, sendo sua intensidade uma função inversamente proporcional ao cosseno do ângulo de incidência do feixe de luz com a reta normal à superfície. Assim define-se a componente de reflexão difusa como:

$$I_d = I_l \cdot K_d \cdot \cos \theta$$

onde alfa é o ângulo de incidência do feixe de luz na superfície.

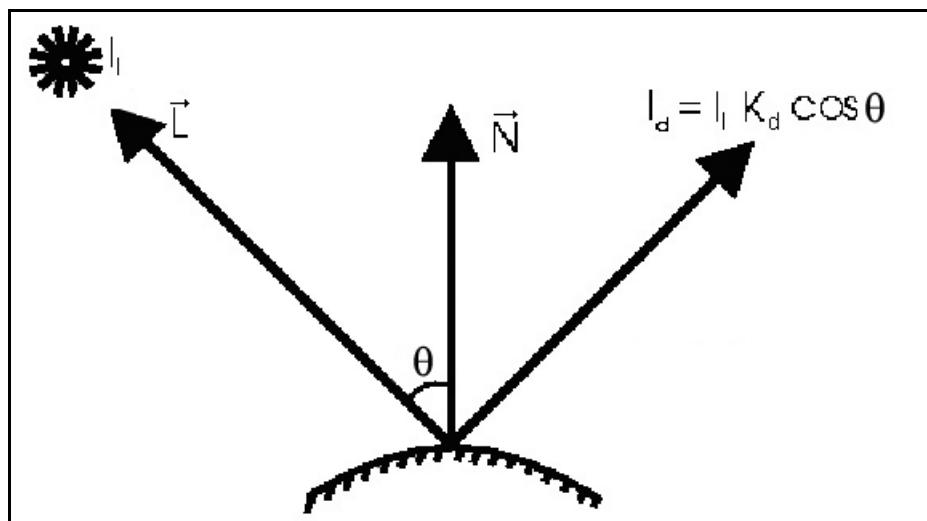


Figura 8.1 – Ângulo de incidência e a Lei de Lambert

Podemos descrever a orientação da superfície num ponto, através do vetor normal unitário \hat{N} , e a direção da luz pelo vetor \hat{L} , também unitário.

Desta forma o ângulo entre a luz e a normal é calculado facilmente como:

$$\cos \theta = \hat{N} \cdot \hat{L}$$

Simplificando, podemos considerar $K_d = 1$ e $I_l = 1$, e a fórmula original fica:

$$I_d = \hat{N} \cdot \hat{L}$$

Ou ainda, combinando a luz ambiente com a luz pontual numa única

fórmula:

$$I_t = I_{la} \cdot K_a + \hat{N} \cdot \hat{L}$$

Para os casos em que não é válida a simplificação $K_d = 1$ e $I_l = 1$ o modelo de reflexão lambertiana, somado à reflexão da luz ambiente pode ser obtida através de:

$$I_t = I_{la} \cdot K_a + I_l \cdot K_d (\hat{N} \cdot \hat{L})$$

8.1.3 ATENUAÇÃO DAS FONTES DE LUZ

A projeção de duas superfícies paralelas, constituídas de um mesmo material, afastadas da fonte de luz por diferentes distâncias resulta, segundo a equação anterior, apresentar intensidades luminosas iguais. Entretanto, quando observamos objetos idênticos que estão em distâncias diferentes somos afetados por fatores externos que atenuam as luzes.

Considerando este fato foi inserido ao modelo anterior um fator de atenuação, que deve considerar a distância entre o objeto e fonte de luz. O novo modelo fica assim constituído:

$$I_t = I_{la} \cdot K_a + f_{att} \cdot I_l \cdot K_d (\hat{N} \cdot \hat{L})$$

Uma escolha um tanto óbvia para o fator f_{att} deve considerar que a energia de uma fonte luminosa pontual decresce com o inverso do quadrado da distância entre o objeto e a fonte luminosa. Neste caso,

$$f_{att} = \frac{1}{d_L^2}$$

Na prática esta aproximação não traz resultados significativos. Portanto um novo modelo foi proposto:

$$f_{att} = \min\left(\frac{1}{c_1 + c_2 d_L + c_3 d_L^2}, 1\right)$$

O valor de f_{att} a ser escolhido deve o menor valor entre os dois termos da expressão. Os valores de c_1 , c_2 e c_3 são constantes definidas pelo usuário que estão diretamente associadas com a fonte de luz.

8.1.4 REFLEXÃO ESPECULAR

Quando observamos uma superfície brilhante iluminada como um metal polido, uma maçã ou a testa de uma pessoa vemos, de acordo com a direção de observação, um brilho ou ponto de luz concentrada.

Este fenômeno é chamado de reflexão especular e é resultado da

reflexão total ou quase total da luz incidente em uma região concentrada ao redor do ângulo de reflexão espelhado. A Figura 8.2 mostra a direção da reflexão espelhada num ponto sobre uma superfície iluminada.

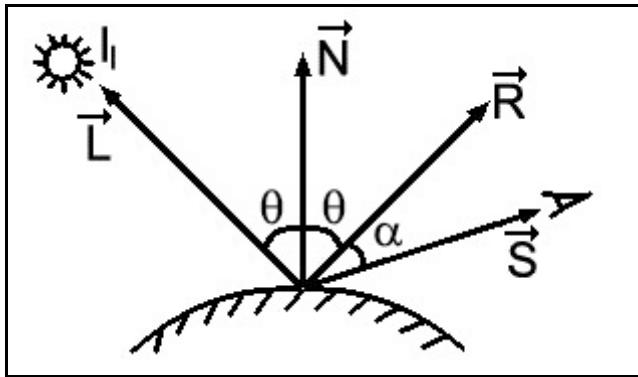


Figura 8.2 – Reflexão espelhada

O ângulo formado entre o vetor L (luz incidente) e o vetor N, normal à superfície, é de θ graus. Este ângulo é igual ao ângulo formado entre o vetor N e o vetor R (luz refletida); a direção do observador é definida através do vetor S. Para uma superfície reflectora ideal (espelho perfeito), a luz incidente é refletida somente na direção da do raio refletido (vetor R). Neste caso, somente veremos luz refletida se os vetores R e S são coincidentes.

Outros objetos que não refletem ideais apresentam reflexão espelhada para uma faixa finita de variação no ângulo α . Superfícies brilhantes tem uma faixa menor de reflexão espelhada enquanto que superfícies foscas tem maior faixa de reflexão espelhada.

O modelo empírico de Bui Tuong Phong é frequentemente usado em modelos simples de iluminação. Especificamente,

$$I_s = I_l \cdot W(i, \lambda) \cdot \cos^n \alpha$$

onde $W(i, \lambda)$ é a curva de reflexão que dá a razão entre a luz refletida espelhadamente e a luz incidente em função do ângulo de incidência i e do comprimento de onda, e n é a aproximação da distribuição espacial da luz refletida espelhadamente.

Valores grandes de n caracterizam distribuições espaciais de metais e outras superfícies especulares. Valores pequenos de n caracterizam superfícies não metálicas e opacas.

A reflexão espelhada é direcional, isto é, depende do ângulo com que a luz incidente atinge a superfície. Para alguns materiais não metálicos a reflexão pode ser pequena, em torno de 4%, enquanto que para materiais metálicos pode exceder 80%.

Combinando os resultados obtidos anteriormente, tem-se um modelo de iluminação dado pela equação:

$$I_t = I_a \cdot K_a + I_l (K_d \cdot \cos \theta + K_s \cdot \cos^n \alpha), \quad 0 \leq \theta \leq \frac{\pi}{2}$$

onde K_s é uma constante frequentemente utilizada para representar $W(i, \lambda)$.

Se múltiplas fontes de luz estiverem presentes, os efeitos são linearmente

adicionados e o modelo de iluminação torna-se

$$I_t = I_{la} \cdot K_a + \sum_{j=1}^m I_{lj} \left(K_d \cdot \cos \theta_j + K_s \cdot \cos^n \alpha_j \right), \quad 0 \leq \theta \leq \frac{\pi}{2}$$

onde m é o número de fontes de luz. Utilizando o produto escalar de dois vetores, pode-se escrever:

$$\cos \theta = \frac{\bar{N} \cdot \bar{L}}{|\bar{N}| \cdot |\bar{L}|} = \hat{N} \cdot \hat{L}$$

onde \hat{N} e \hat{L} são as unidades vetoriais da normal à superfície e da direção da fonte de luz, respectivamente.

Similarmente,

$$\cos \alpha = \frac{\bar{R} \cdot \bar{S}}{|\bar{R}| \cdot |\bar{S}|} = \hat{R} \cdot \hat{S}, \text{ onde } \hat{R} = (2\hat{L} \cdot \hat{N}) \cdot \hat{N} - \hat{L}$$

onde \hat{R} e \hat{S} são unidades vetoriais para a luz refletida e a direção da linha de visão.

Então, o modelo de iluminação para uma única fonte de luz é dado por:

$$I_t = I_{la} K_a + I_l \left(K_d (\hat{N} \cdot \hat{L}) + K_s (\hat{R} \cdot \hat{S})^n \right)$$

Em computação gráfica este modelo é frequentemente chamado de função de iluminação. A função de iluminação é aplicada individualmente a cada uma das três cores primárias em uma imagem colorida. Isso é uma simplificação (em geral, a reflexão varia para cada componente espectral), mas é simples de implementar e produz resultados aceitáveis.

8.1.5 CONSIDERAÇÕES SOBRE CORES

Os modelos de iluminação até aqui estudados consideram somente os efeitos de iluminação monocromática. Para incorporar as cores precisamos escrever a equação da intensidade de iluminação como uma função das cores das fontes de luz e das superfícies dos objetos.

No modelo RGB cada cor na cena é expressa como uma combinação das componentes vermelho, verde e azul. Então, especificando as componentes RGB da fonte de luz e das cores das superfícies, podemos usar um modelo de iluminação para calcular as componentes RGB da luz refletida. Uma maneira de ajustar as cores das superfícies é representar os coeficientes de reflexão como vetores tridimensionais. O vetor coeficiente de reflexão difusa, por exemplo, terá as componentes RGB (K_{dR} , K_{dG} , K_{dB}). Se desejarmos um objeto com superfície azul, selecionamos um valor diferente de zero, numa faixa de 0 a 1, para a componente de reflexão azul, K_{dB} , enquanto as componentes de reflexão vermelha e verde assumem valor 0 ($K_{dR} = K_{dG} = 0$). Qualquer componente vermelha ou verde da luz

incidente será absorvida e somente a componente azul será refletida. O cálculo da intensidade de iluminação para este exemplo pode ser simplificado para a expressão:

$$I_{tB} = K_{aB} I_{laB} + \sum_{i=1}^n f_{att}(d_i) \cdot I_{lBi} \left[K_{dB} (\hat{N} \cdot \hat{L}_i) + K_{sB} (\hat{R}_i \cdot \hat{S})^n \right]$$

No modelo de reflexão especular original Phong ajustou o parâmetro K_s para um valor constante independente da cor da superfície. Isto produz uma reflexão especular com a mesma cor da luz incidente (geralmente branco), que dá à superfície uma aparência plástica. Para um material não plástico, a cor da reflexão especular é função das propriedades da superfície e pode ser diferente da cor da luz incidente e diferente da cor da reflexão difusa.

8.1.6 TRANSPARÊNCIA

Uma superfície transparente geralmente reflete e refrata a luz. A quantidade de luz refratada depende do grau de transparência da superfície e se há fontes de luz ou superfícies iluminadas atrás dela. A Figura 8.3 ilustra o efeito de uma superfície iluminada na transparência de um objeto.

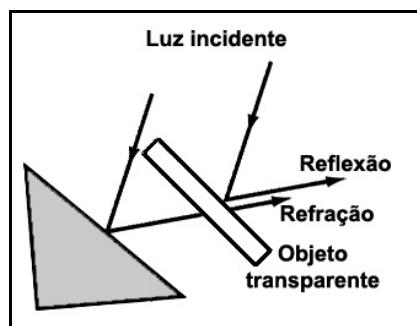


Figura 8.3 – Reflexão e refração de luz em uma superfície transparente.

Quando uma superfície transparente for modelada, a função de iluminação deverá ser modificada para incluir a quantidade da luz que passa através da superfície. Na maioria dos casos, a luz transmitida é gerada pelos objetos refletores que estão atrás da superfície, como mostra a Figura 8.4. A luz refletida por estes objetos passa através da superfície transparente e contribui na intensidade luminosa total desta.



Figura 8.4 – A luz refletida pelos objetos mais afastados atravessa a superfície

Tanto a reflexão difusa como a reflexão especular afetam as superfícies dos objetos transparentes. Os efeitos difusos assumem importância quando a superfície a ser modelada é parcialmente transparente, como um copo coberto por gelo. A luz que passa através destes objetos é dispersa de forma que uma imagem embaçada do objeto de fundo é obtida. A refração difusa pode ser gerada decrementando a intensidade da luz refratada e aumentando a intensidade luminosa para cada ponto na superfície refratadora numa área finita. Esta manipulação consome tempo e muitos modelos de iluminação empregam somente os efeitos da reflexão especular.

Os efeitos de transparência realista são modelados considerando-se a refração da luz. Quando a luz incide sobre uma superfície transparente, parte desta luz é refletida a parte é refratada (Figura 8.5).

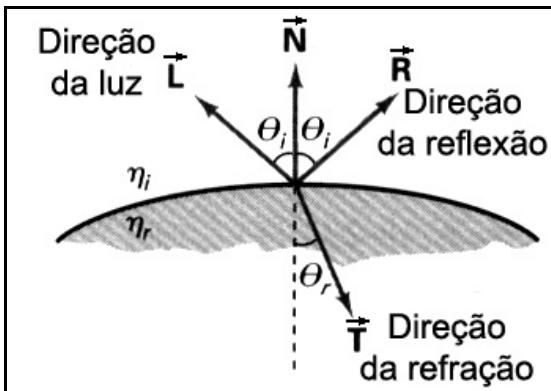


Figura 8.5 – Direção de reflexão e a direção de refração em uma superfície transparente

Devido à velocidade da luz ser diferente em materiais diferentes, o caminho da luz refratada é diferente daquele da luz incidente. A direção da luz refratada, especificada pelo ângulo de refração, é uma função do índice de refração de cada material e da direção da luz incidente. O índice de refração para um material é definido como o quociente entre a velocidade da luz no vácuo e velocidade da luz no material. O ângulo de refração θ_r é calculado a partir de um ângulo de incidência θ_i , do índice de refração η_i do material incidente (geralmente o ar), a o índice de refração η_r do material refratário de acordo com a lei de Snell:

$$\sin \theta_r = \frac{\eta_i}{\eta_r} \sin \theta_i$$

Atualmente, o índice de refração de um material é uma função do comprimento de onda da luz incidente, de maneira que as diferentes componentes de cor de um raio luminoso serão refratadas em diferentes ângulos. Para muitas aplicações podemos usar um índice médio de refração para os diferentes materiais que estão modelados na cena. O índice de refração do ar é aproximadamente 1 enquanto que o de uma coroa de vidro é aproximadamente 1,5. Usando estes valores na lei de Snell com um ângulo de incidência de 30° obtemos um ângulo de refração de aproximadamente 19°. A Figura 8.6 ilustra a mudança no caminho do raio de luz refratado pelo objeto de vidro. O efeito global da refração é mudar a direção da luz incidente para um caminho paralelo. Como o cálculo trigonométrico utilizado na lei de Snell é computacionalmente caro, os efeitos da refração podem ser modelados por um pequeno deslocamento do raio de luz incidente.

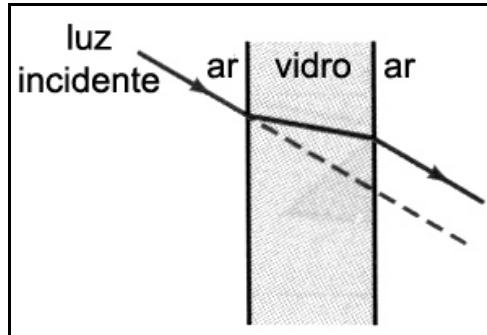


Figura 8.6 – A refração da luz através de um objeto de vidro

Partindo da lei de Snell e do diagrama da Figura 8.5, podemos obter o vetor unitário de transmissão \hat{T} na direção de refração θ , como:

$$\hat{T} = \left(\frac{\eta_i}{\eta_r} \cos \theta_i - \cos \theta_r \right) \hat{N} - \frac{\eta_i}{\eta_r} \hat{L}$$

onde \hat{N} é o vetor normal unitário, e \hat{L} é o vetor unitário na direção do fonte de luz. O vetor de transmissão \hat{T} pode ser usado para localizar interseções do caminho da luz refratada com os objetos atrás da superfície transparente. Incluindo os efeitos da refração na cena podemos produzir cenas altamente realistas, mas a determinação dos caminhos da refração e a interseção com os objetos requer um elevado esforço computacional. A maioria dos modelos de transmissão de luz tem aproximações que reduzem o tempo de processamento.

Um procedimento simples para modelar objetos transparentes é ignorar completamente a mudança no caminho do raio de refração. De fato, esta aproximação assume que não há mudança no índice de refração de um material para outro, de maneira que o ângulo de refração é sempre o mesmo ângulo de incidência. Este método acelera o cálculo das intensidades e pode produzir efeitos de transparência razoáveis para polígonos de superfícies finas.

Podemos combinar a intensidade transmitida I_{trans} através da superfície por um objeto de fundo com a intensidade de reflexão I_{refl} da superfície transparente (Figura 8.3) usando um coeficiente de transparência K_t . Assumimos o parâmetro K_t variando de 0 a 1 para especificar quanto da luz de fundo é transmitida. A intensidade total da superfície pode ser calculada como:

$$I = (1 - K_t) I_{refl} + K_t I_{trans}$$

o termo $(1 - K_t)$ é o fator de opacidade.

Para objetos altamente transparentes assumimos K_t próximos a 1. Objetos quase opacos transmitem muito pouca luz dos objetos de fundo, então podemos ajustar K_t para um valor próximo de 0 para estes materiais (opacidade próxima de 1).

Os efeitos de transparência são frequentemente implementados com algoritmos de profundidade modificados (*z-buffer*). Uma maneira simples para fazer isto é processar primeiro os objetos opacos, determinando a profundidade para as superfícies opacas visíveis. Então a profundidade dos objetos transparentes é comparada com os valores previamente armazenados no *buffer* de profundidade. Se

qualquer superfície transparente é visível, sua intensidade de reflexão é calculada e combinada com a intensidade da superfície opaca previamente armazenada no *buffer* de imagem.

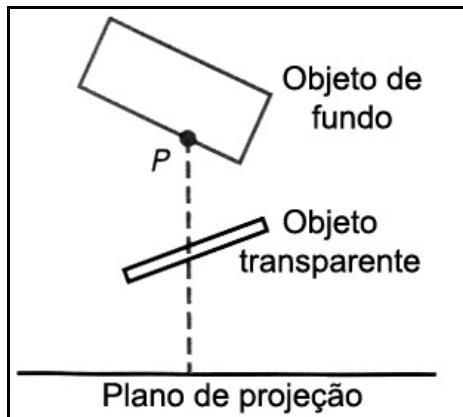


Figura 8.7 – A intensidade de um objeto de fundo num ponto P

Este método pode ser modificado para produzir imagens mais apuradas através da armazenagem adicional da profundidade e outros parâmetros das superfícies transparentes. Isto permite comparar os valores de profundidade das superfícies transparentes com quaisquer outras, bem como os valores de profundidade das superfícies opacas. Superfícies transparentes visíveis são então renderizadas combinando sua intensidade com a intensidade de superfícies visíveis e superfícies opacas que estão atrás dela.

8.1.7 SOMBRAIS

Os métodos de ocultação de superfícies podem ser utilizados para localizar as áreas onde as fontes de luz produzem sombras. Aplicando um método de ocultação de superfícies, com o observador posicionado na fonte de luz, podemos determinar quais partes das superfícies não podem ser vistas a partir da posição da fonte de luz. Estas são as áreas de sombra. Uma vez que determinamos as áreas de sombra para todas as fontes de luz, estas podem ser tratadas como superfícies e armazenadas na lista de superfícies. A Figura 8.8 mostra a determinação das sombras para dois objetos sobre uma mesa iluminados por uma fonte de luz distante. Todas as áreas de sombra nesta figura são superfícies que não são visíveis a partir da posição da fonte de luz.

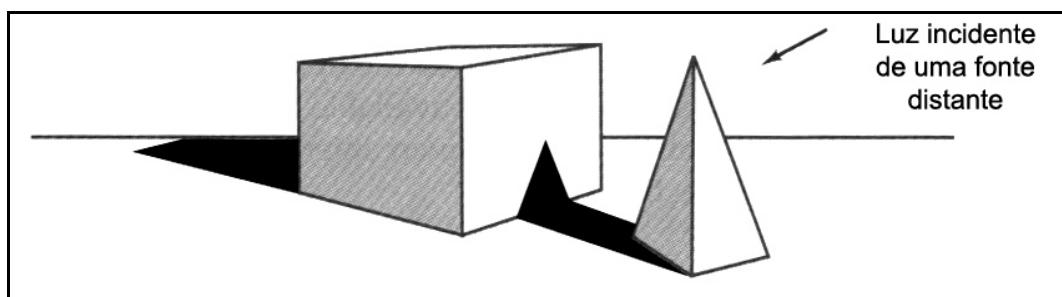


Figura 8.8 – Modelagem de objetos com regiões de sombra.

As sombras geradas por métodos de ocultação de superfícies são válidas para qualquer posição do observador, desde que a fonte de luz não tenha sua

posição alterada. As superfícies que são visíveis a partir da posição do observador são sombreadas de acordo com o modelo de iluminação empregado, o qual pode ser combinado com texturas. Somente podemos exibir áreas de sombra com a intensidade de iluminação ambiente, ou então podemos combinar a luz ambiente com texturas especificadas.

8.2 SOMBREAMENTO PARA POLÍGONOS

8.2.1 SOMBREAMENTO CONSTANTE

O modelo de sombreamento mais simples que podemos aplicar a um polígono é o sombreamento constante. Nesta solução aplica-se um modelo de iluminação apenas uma vez por polígono plano, determinando-se um único valor de intensidade utilizado para o preenchimento de todo o polígono. Esta aproximação é ainda assim válida se várias suposições forem verdadeiras:

1. A fonte de luz localiza-se no infinito, fazendo com que o ângulo de incidência dos feixes de luz possua o mesmo ângulo ao longo de toda a superfície do polígono;
2. O observador localiza-se no infinito, os feixes de luz refletida na superfície do polígono atingem o observador sob o mesmo ângulo de visada;
3. O polígono (plano) representa uma superfície plana e não é uma aproximação de uma superfície curva.

Se qualquer uma das suposições apresentadas não for satisfeita, então o modelo de sombreamento constante não pode ser aplicado com resultados satisfatórios.

8.2.2 SOMBREAMENTO GOURAUD

Uma alternativa à técnica de sombreamento foi proposta por H. Gouraud em 1971. Esta técnica consiste no cálculo do modelo de iluminação a amostras da superfície de um polígono seguida da aplicação de um interpolador linear. Este método é particularmente adequado a algoritmos de apresentação ou cálculo de visibilidade que se utilizem de *scan line* para a ocultação de faces, como o *z-buffer*.

Para obter-se maior eficiência no cálculo da luz refletida pode-se usar uma equação incremental como a apresentada para projeções paralelas com *z-buffer*. Quando se utiliza a projeção em perspectiva ou quando a superfície representada pelo polígono plano não é verdadeiramente plana, este modelo não pode ser aplicado com resultados realmente satisfatórios. Outra grande desvantagem da solução de Gouraud está na sua incapacidade de apresentar pontos de reflexão acentuada situados no interior da superfície de polígonos.

Podemos dividir o sombreamento Gouraud nos seguintes passos:

- Determinar vetor normal unitário médio para cada vértice do objeto;
- Aplicar a função de iluminação calculando a intensidade de iluminação total em cada um dos vértices do objeto;
- Interpolar linearmente as intensidades dos vértices para o restante de cada face do objeto.

A Figura 8.9 apresenta uma superfície para a qual será calculado o vetor normal unitário médio.

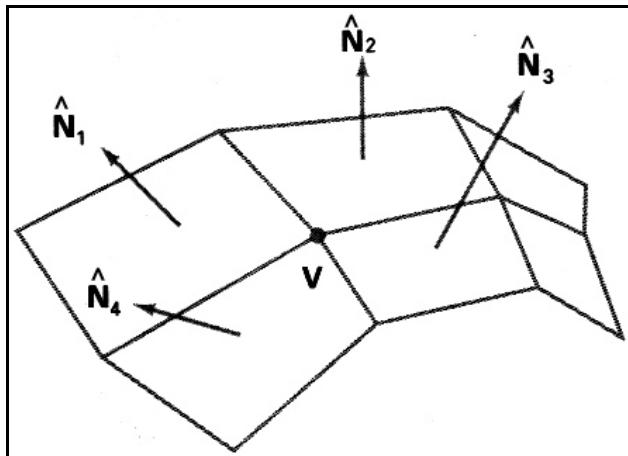


Figura 8.9 – O processo de obtenção do vetor normal unitário médio num vértice V

Para cada vértice do polígono obtemos um vetor normal através da média dos vetores unitários normais às superfícies dos polígonos que compartilham o vértice V. O vetor normal unitário médio pode ser obtido através de:

$$\hat{N}_V = \frac{\sum_{k=1}^n \hat{N}_k}{\left| \sum_{k=1}^n \hat{N}_k \right|}$$

Uma vez calculada as normais aos vértices podemos determinar a intensidade dos vértices através de um modelo de iluminação.

A Figura 8.10 mostra o próximo passo: interpolar a intensidade ao longo dos vértices do polígono. Para cada *scan line*, a intensidade na interseção da *scan line* com uma aresta do polígono é linearmente interpolada entre as intensidades dos pontos extremos da aresta.

No exemplo da Figura 8.10, a aresta do polígono com vértices extremos nos pontos 1 e 2 é interceptada pela *scan line* no ponto 4 e sua intensidade é interpolada entre as intensidades I_1 e I_2 usando somente o deslocamento vertical da *scan line*:

$$I_4 = \frac{y_4 - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y_4}{y_1 - y_2} I_2$$

Similarmente, a intensidade na interseção da direita da *scan line* é interpolada entre os valores das intensidades nos vértices 2 e 3. Uma vez calculada as intensidades para os extremos da *scan line* podemos, para um ponto interior p , interpolar o valor da intensidade de iluminação entre as intensidades nos pontos 4 e 5:

$$I_p = \frac{x_5 - x_p}{x_5 - x_4} I_4 + \frac{x_p - x_4}{x_5 - x_4} I_5$$

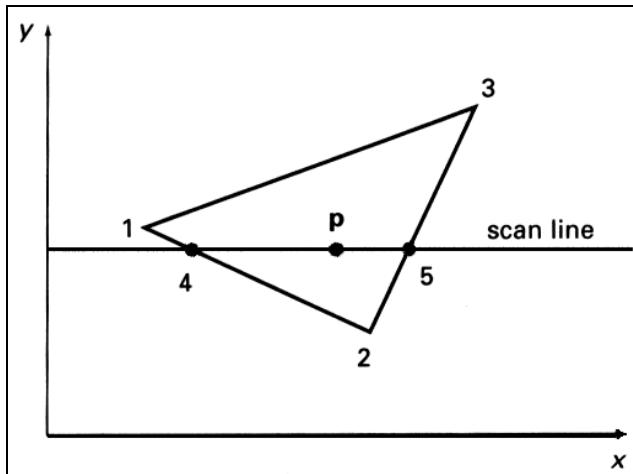


Figura 8.10 – A interpolação de pontos entre os vértices do polígono

Cálculos incrementais são usados para obter os valores de intensidade entre as *scan lines* e as intensidades dentro de cada *scan line*. Como mostra a Figura 8.11, se a intensidade na posição (x, y) da aresta é interpolada como:

$$I = \frac{y - y_2}{y_1 - y_2} I_1 + \frac{y_1 - y}{y_1 - y_2} I_2$$

então podemos obter a intensidade ao longo desta aresta para a próxima *scan line*, $y - 1$, através de:

$$I' = I + \frac{I_2 - I_1}{y_1 - y_2}$$

Cálculos similares são utilizados para obter a intensidade dos pixels sucessivos na linha horizontal ao longo da *scan line*.

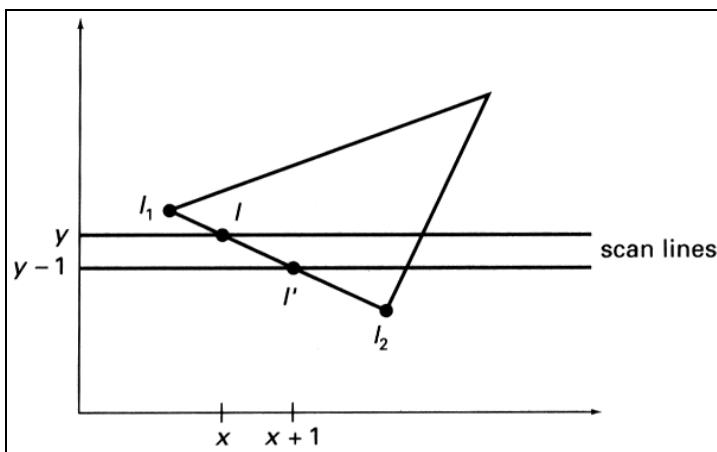


Figura 8.11 – Interpolação incremental dos valores de intensidade ao longo da aresta do polígono para *scan lines* sucessivas

8.2.3 SOMBREAMENTO PHONG

Outra técnica bastante difundida foi descrita por Bui Tuong Phong em 1973. Phong propôs a interpolação linear dos vetores normais para o cálculo do

sombreamento, com o posterior cálculo do modelo de iluminação. O modelo de Phong captura adequadamente a variação do ângulo de incidência do feixe de luz na superfície, possibilitando a apresentação de pontos de alta reflexividade afastados das extremidades dos polígonos.

Para obter-se maior eficiência no cálculo do ângulo de incidência do feixe de luz pode-se usar uma equação incremental como a apresentada para projeções paralelas com *z-buffer*. Quando se utiliza a projeção em perspectiva este modelo pode ser aplicado com resultados satisfatórios. Outra grande vantagem da solução de Phong está na sua capacidade de apresentar pontos de reflexão acentuada situados no interior da superfície de polígonos.

Podemos dividir o sombreamento Phong nos seguintes passos:

- Determinar o vetor normal unitário médio para os vértices de todas as faces do objeto;
- Interpolar linearmente o vetor normal unitário médio dos vértices ao longo da superfície do polígono;
- Aplicar a função de iluminação ao longo de cada *scan line* para calcular a intensidade de cada pixel projetado.

A interpolação de normais às superfícies ao longo de uma aresta entre dois vértices é ilustrada na Figura 8.12. O vetor normal \hat{N} na interseção da *scan line* ao longo da aresta entre os vértices 1 e 2 pode ser obtido pela interpolação vertical entre a normal dos pontos extremos da aresta:

$$\hat{N} = \frac{y - y_2}{y_1 - y_2} \hat{N}_1 + \frac{y_1 - y}{y_1 - y_2} \hat{N}_2$$

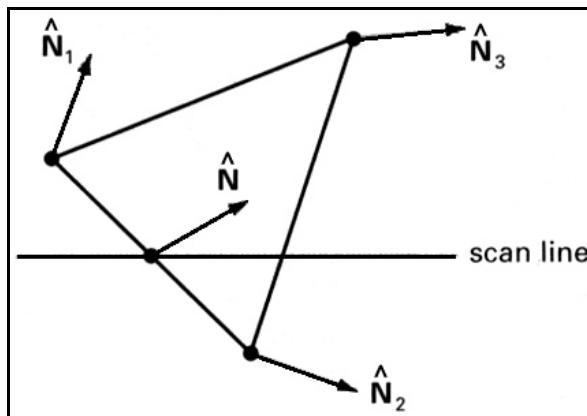


Figura 8.12 – Interpolação dos vetores normais ao longo da aresta de um polígono

Métodos incrementais são utilizados para calcular as normais entre *scan lines* e ao longo de cada *scan line* individual. Em cada pixel ao longo da *scan line*, o modelo de iluminação é aplicado para determinar a intensidade da superfície naquele ponto.

O cálculo da intensidade utilizando um vetor normal aproximado para cada ponto ao longo da *scan line* produz resultados mais precisos do que a interpolação direta das intensidades, como no sombreamento Gouraud. A desvantagem é que o modelo Phong requer um número maior de cálculos.

8.2.4 SOMBREAMENTO “FAST PHONG”

A renderização de superfícies com o sombreamento Phong pode ser acelerado com o uso de aproximações nos cálculos dos vetores normais no modelo de iluminação. O sombreamento “Fast Phong” aproxima os cálculos de intensidade usando uma série de Taylor expandida e superfícies triangulares.

O sombreamento Phong interpola vetores normais a partir das normais nos vértices. Portanto podemos expressar um vetor \mathbf{N} normal à superfície num ponto (x, y) através de um triângulo como:

$$\vec{N} = \vec{A}x + \vec{B}y + \vec{C}$$

onde os vetores A, B e C são determinados através de uma equação de três vértices:

$$\vec{N}_k = \vec{A}x_k + \vec{B}y_k + \vec{C}, \quad k = 1, 2, 3$$

com (x_k, y_k) denotando a posição de um vértice.

Omitindo a reflexividade e os parâmetros de atenuação, podemos escrever o cálculo para reflexão difusa num ponto (x, y) de uma superfície como:

$$\begin{aligned} I_{\text{diff}}(x, y) &= \frac{\vec{L} \cdot \vec{N}}{|\vec{L}| \cdot |\vec{N}|} \\ &= \frac{\vec{L} \cdot (\vec{A}x + \vec{B}y + \vec{C})}{|\vec{L}| \cdot |\vec{A}x + \vec{B}y + \vec{C}|} = \frac{(\vec{L} \cdot \vec{A})x + (\vec{L} \cdot \vec{B})y + \vec{L} \cdot \vec{C}}{|\vec{L}| |\vec{A}x + \vec{B}y + \vec{C}|} \end{aligned}$$

Nós podemos reescrever esta expressão no forma:

$$I_{\text{diff}}(x, y) = \frac{ax + by + c}{\sqrt{(dx^2 + exy + fy^2 + gx + hy + i)}}$$

onde os parâmetros como a, b, c e d são usados para representar os vários produtos escalares. Por exemplo:

$$a = \frac{\vec{L} \cdot \vec{A}}{|\vec{L}|}$$

Finalmente podemos expressar o denominador da equação anterior como um série de Taylor expandida com os termos x e y reduzidos ao segundo grau. Desta forma:

$$I_{\text{diff}}(x, y) = T_5x^2 + T_4xy + T_3y^2 + T_2x + T_1y + T_0$$

onde cada T_k é uma função dos parâmetros a, b, c e assim por diante.

O sombreamento Phong é 6 a 7 vezes mais lento que o sombreamento Gouraud. O sombreamento “Fast Phong” é mais rápido que o sombreamento Phong

mas ainda é 2 vezes mais lento que o sombreamento Gouraud.

8.3 MÉTODOS DE RAY TRACING

O *Ray tracing* é um método utilizado na descoberta da interseção entre objetos, na ocultação de superfícies e também na iluminação das superfícies.

Em vez de simplesmente olharmos as superfícies visíveis para cada pixel, continuamos a trajetória do raio na cena, como ilustra a Figura 8.13, somando contribuições na intensidade luminosa das superfícies. Este processo oferece uma técnica de renderização simples e poderosa para obtenção da reflexão global e da transmissão de efeitos.

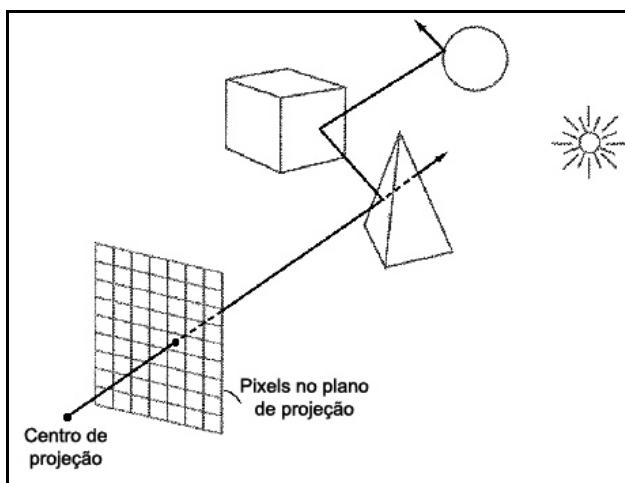


Figura 8.13 – Traçando um raio do ponto de observação através de um pixel com múltiplas reflexões e transmissões

O algoritmo de *ray tracing* básico permite, também, identificar as superfícies ocultas, efeitos de sombras, transparência, e iluminação a partir de luzes múltiplas. Muitos aperfeiçoamentos para o algoritmo foram elaborados para produzir cenas fotorealistas. Cenas que utilizam o *ray tracing* podem ser altamente realistas, particularmente no brilho dos objetos, mas este algoritmo requer elevado esforço computacional. Um exemplo de reflexão global e transmissão de efeitos usando *ray tracing* é mostrado na Figura 8.14.

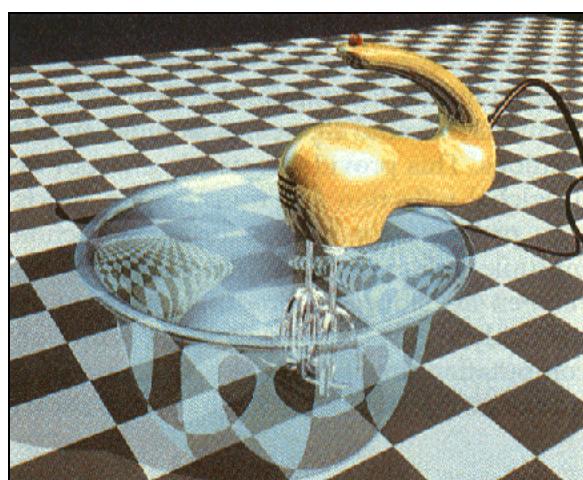


Figura 8.14 – Uma cena renderizada com *ray tracing* mostrando efeitos de transmissão e reflexão globais

8.3.1 ALGORITMO DE RAY TRACING BÁSICO

Primeiramente ajustamos um sistema de coordenadas com os pixels da imagem representados no plano xy . A descrição da cena é fornecida num quadro de referência (Figura 8.15). A partir de centro de projeção determinamos o caminho de um raio que passa através do centro de cada pixel da tela. Os efeitos de iluminação que se acumulam ao longo do caminho do raio são, então, atribuídos ao pixel.

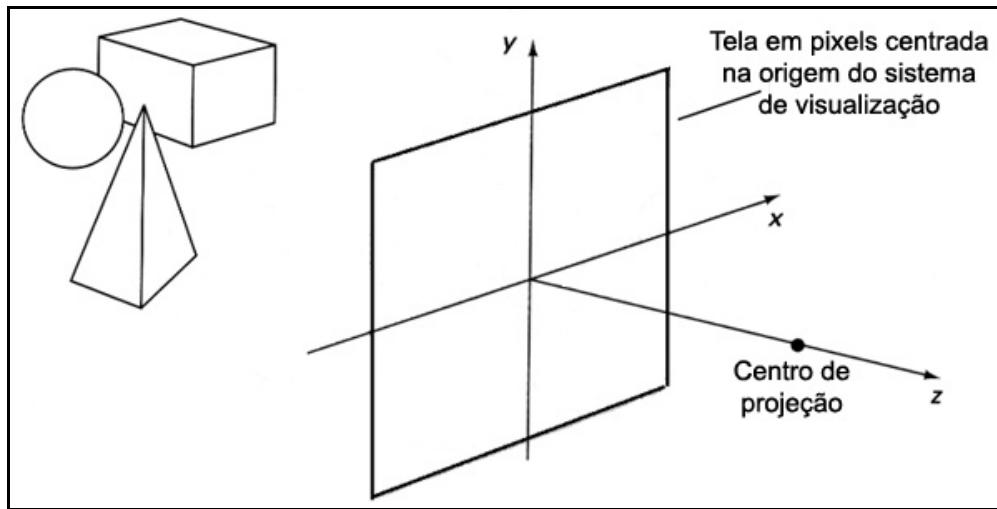


Figura 8.15 – Quadro de coordenadas de referência do *ray tracing*

Esta renderização aproximada está baseada nos princípios da geometria óptica. Raios de luz emanam da superfície dos objetos em todas as direções e alguns deles passarão através dos pixels no plano de projeção. Como na cena há um número infinito de raios, determinamos a contribuição de um pixel em particular pelo traçado de um raio de luz do pixel até a cena. Primeiro consideraremos o algoritmo de *ray tracing* básico com um raio por pixel, que é equivalente a visualizar a cena através do diafragma de uma câmara.

Testamos cada superfície na cena para determinar se ela é interceptada pelo raio que sai do centro de projeção e passa por um pixel na tela. Se a superfície é interceptada calculamos a distância do pixel até o ponto de interseção. A menor distância de interseção calculada identifica a superfície visível através daquele pixel. Então refletimos o raio, partindo da superfície visível, através de um caminho espelhado (ângulo de reflexão igual ao ângulo de incidência). Se a superfície é transparente, também emitimos um raio através da superfície na direção da refração. Os raios de reflexão e refração são referenciados como raios secundários.

Este procedimento é repetido para cada raio secundário: objetos são testados para verificar interseções e a superfície mais próxima no caminho do raio secundário é utilizada para definir, recursivamente, os próximos caminhos de reflexão e refração. Como os raios partindo de um pixel ricocheteiam através da cena, cada interseção com as superfícies é sucessivamente adicionada numa árvore binária, como mostra a Figura 8.16. Usamos os ramos da esquerda para representar caminhos de reflexão, e os ramos da direita para representar os caminhos de transmissão (refração). A profundidade máxima da árvore pode ser definida pelo usuário, ou determinada pela quantidade de memória disponível. O trajeto de um raio tem fim quando atinge a profundidade máxima da árvore, quando o raio atinge uma fonte de luz ou quando não intercepta nenhum outro objeto.

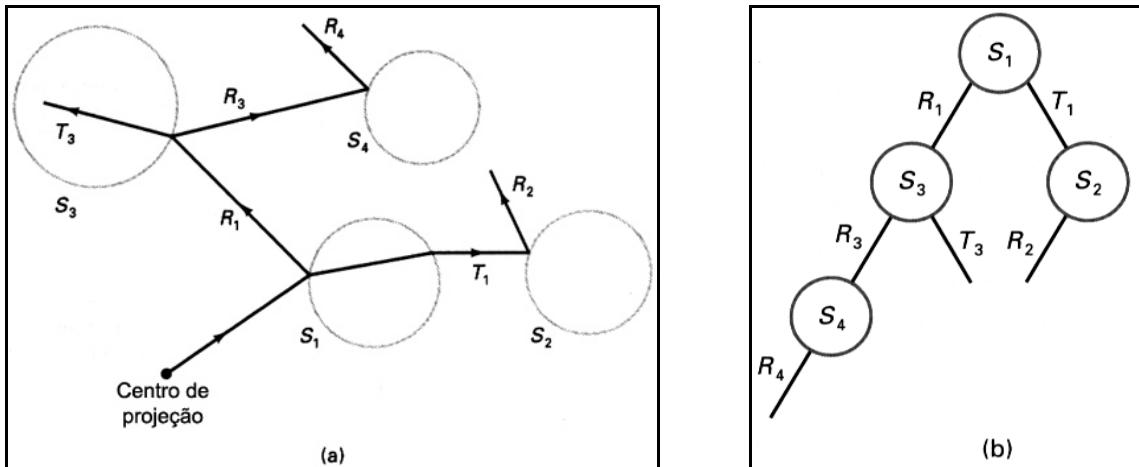


Figura 8.16 – (a) O caminho dos raios de reflexão e refração através da cena para um pixel da tela. (b) Árvore binária mostrando os caminhos do raio na cena (a)

A intensidade luminosa atribuída ao pixel é determinada pelo acúmulo das contribuições, iniciando na base da árvore (nós terminais). A intensidade da superfície em cada nó da árvore é atenuada pela distância da superfície “pai” (nó em nível superior) e adicionada à intensidade desta. A intensidade do pixel é então a soma das intensidades atenuadas até o nó raiz da árvore do raio. Se nenhuma superfície é interceptada pelo raio saindo do pixel, a árvore do raio é vazia e ao pixel é atribuído o valor da intensidade do fundo. Se um raio que sai do pixel intercepta uma fonte de luz opaca, ao pixel é atribuída a intensidade da fonte, embora as fontes de luz sejam usualmente colocadas além do caminho dos raios iniciais.

A Figura 8.17 mostra uma superfície interceptada por um raio e os demais elementos necessários calcular a intensidade da luz refletida. O vetor unitário \hat{u} indica a direção do caminho do raio, \hat{N} é o vetor unitário normal à superfície, \hat{R} é o vetor unitário na direção da reflexão, \hat{L} é o vetor unitário apontando para a fonte de luz e \hat{H} é o vetor unitário na bissetriz entre os vetores $-\hat{u}$ e \hat{L} . O caminho através de \hat{L} é referenciado como o **raio da sombra**. Se qualquer objeto é interceptado pelo raio da sombra entre a superfície e a fonte de luz, a superfície está na sombra com respeito àquela fonte.

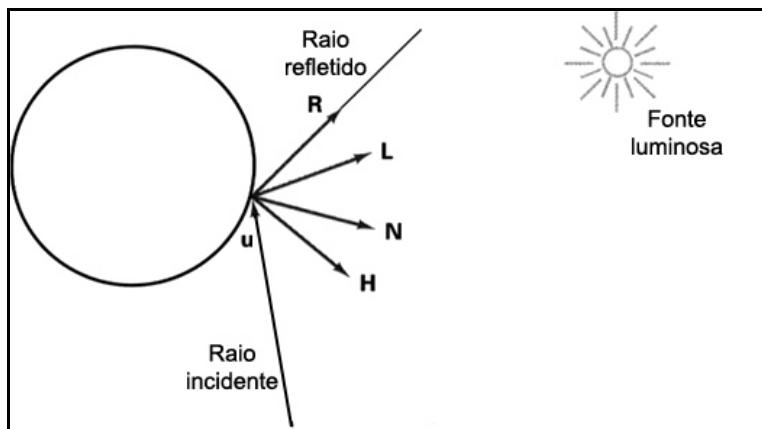


Figura 8.17 – Vetores unitários na superfície de um objeto interceptado por um raio chegando na direção de \hat{u}

A luz ambiente na superfície é calculada por $I_{la} \cdot K_a$; a reflexão difusa devido à fonte é proporcional a $K_d (\hat{N} \cdot \hat{L})$; e a componente da reflexão especular é

proporcional à $K_s(\hat{H} \cdot \hat{N})$. A reflexão especular para o raio secundário \hat{R} depende da normal da superfície e da direção de chegada do raio:

$$\hat{R} = \hat{u} - (2\hat{u} \cdot \hat{N})\hat{N}$$

Para uma superfície transparente, necessitamos obter, também, as contribuições da luz transmitida através do material. Podemos localizar a fonte desta contribuição traçando um raio secundário através da direção de transmissão \hat{T} , como mostra a Figura 8.18. O vetor unitário de transmissão pode ser obtido a partir dos vetores \hat{u} e \hat{N} , assim:

$$\hat{T} = \frac{\eta_i}{\eta_r}\hat{u} - \left(\cos\theta_r - \frac{\eta_i}{\eta_r} \cos\theta_i \right)\hat{N}$$

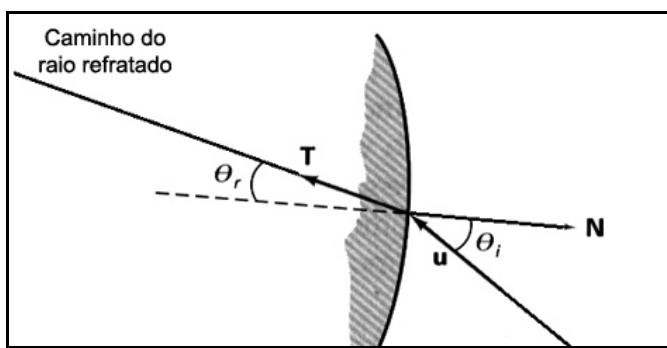


Figura 8.18 – O caminho de um raio refratado por um material transparente

Os parâmetros η_i e η_r são os índices de refração do meio e do material refratário, respectivamente. O ângulo de refração θ , pode ser calculado pela lei de Snell:

$$\cos\theta_r = \sqrt{1 - \left(\frac{\eta_i}{\eta_r} \right)^2 (1 - \cos^2\theta_i)}$$

8.3.2 CALCULANDO A INTERSEÇÃO DO RAIO COM AS SUPERFÍCIES

Um raio pode ser descrito através de uma posição inicial \mathbf{P}_0 e um vetor direção unitário \hat{u} , como ilustra a Figura 8.19.

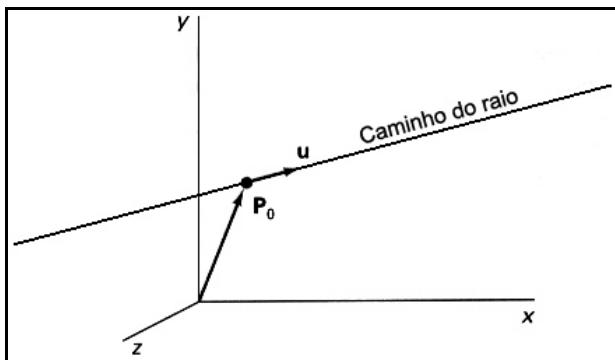


Figura 8.19 – Descrição de um raio com posição inicial em \mathbf{P}_0 e direção definida pelo vetor \hat{u}

As coordenadas de um ponto \mathbf{P} , ao longo do raio, numa distância s a partir de \mathbf{P}_0 é calculada através de **equação do raio**:

$$\mathbf{P} = \mathbf{P}_0 + s\hat{\mathbf{u}}$$

Inicialmente, \mathbf{P}_0 pode ser localizado na posição de um pixel no plano de projeção, ou pode assumir a posição do centro de projeção. O vetor unitário $\hat{\mathbf{u}}$ é inicialmente obtido a partir da posição do pixel, através do qual o raio passa, e do centro de projeção:

$$\hat{\mathbf{u}} = \frac{\mathbf{P}_{pix} - \mathbf{P}_{cp}}{|\mathbf{P}_{pix} - \mathbf{P}_{cp}|}$$

Em cada superfície interceptada, os vetores \mathbf{P}_0 e $\hat{\mathbf{u}}$ são atualizados para os raios secundários no ponto de interseção do raio com a superfície. Para os raios secundários, a direção da reflexão para $\hat{\mathbf{u}}$ é $\hat{\mathbf{R}}$ e a direção da transmissão é $\hat{\mathbf{T}}$. A localização das interseções é feita resolvendo-se a igualdade, **equação do raio = equação da superfície**, para cada objeto na cena.

O objeto mais simples para o *ray tracing* é a esfera. Se temos uma esfera de raio r e centro em \mathbf{P}_c (Figura 8.20), então qualquer ponto \mathbf{P} sobre a superfície da esfera satisfaz a equação da esfera:

$$|\mathbf{P} - \mathbf{P}_c|^2 - r^2 = 0$$

Substituindo a **equação do raio** na equação anterior temos:

$$|\mathbf{P}_0 + s\hat{\mathbf{u}} - \mathbf{P}_c|^2 - r^2 = 0$$

Fazendo com que $\Delta\mathbf{P} = \mathbf{P}_c - \mathbf{P}_0$ e expandindo o produto escalar, obtemos a equação quadrática:

$$s^2 - 2(\hat{\mathbf{u}} \cdot \Delta\mathbf{P})s + (|\Delta\mathbf{P}|^2 - r^2) = 0$$

cuja solução é:

$$s = \hat{\mathbf{u}} \cdot \Delta\mathbf{P} \pm \sqrt{(\hat{\mathbf{u}} \cdot \Delta\mathbf{P})^2 - |\Delta\mathbf{P}|^2 + r^2}$$

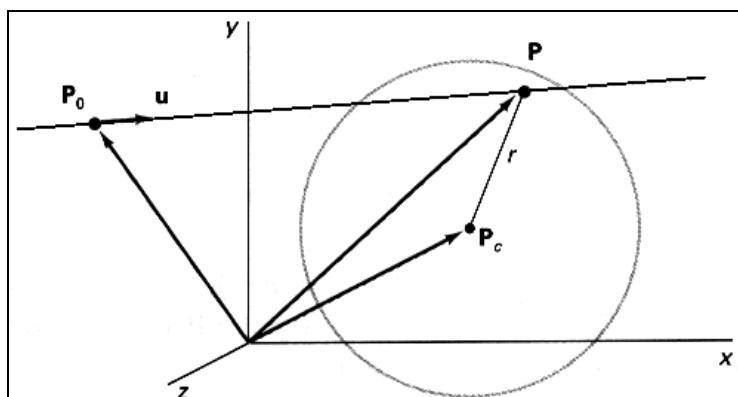


Figura 8.20 – Um raio interceptando uma esfera de raio r e centro em \mathbf{P}_c

Se o radicando for negativo, o raio não intercepta a esfera. Por outro lado, as coordenadas da interseção são obtidas a partir da **equação do raio** usando o menor dos dois valores obtidos na equação anterior.

Em pequenas esferas distantes da posição inicial do raio, a equação anterior é suscetível a erros de arredondamento. Isto é, se $r^2 \ll |\Delta P|^2$, podemos perder o termo r^2 no arredondamento de $|\Delta P|^2$. Evitamos este erro na maioria dos casos rescrevendo o cálculo para a distância s :

$$s = \hat{u} \cdot \Delta P \pm \sqrt{r^2 - |\Delta P - (\hat{u} \cdot \Delta P)\hat{u}|^2}$$

A localização das interseções entre um raio e a superfície de um poliedro requer mais processamento que a esfera. Por este motivo, é sempre melhor realizar um teste de interseção inicial num volume limitado. Por exemplo, a Figura 8.21 mostra um poliedro limitado por uma esfera.

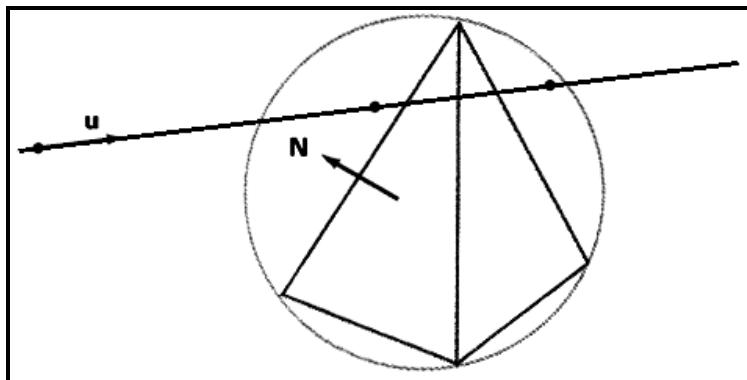


Figura 8.21 – Um poliedro circunscrito numa esfera

Se o raio não intercepta a esfera, não precisamos realizar testes para o poliedro. Mas se o raio intercepta a esfera, primeiro localizamos as faces frontais como o teste:

$$\hat{u} \cdot \hat{N} < 0$$

onde \hat{N} é a normal unitária à superfície. Para cada face do poliedro que satisfaça a inequação anterior, resolvemos a equação do plano

$$\vec{N} \cdot P = -D$$

para o ponto P da superfície que também satisfaz a **equação do raio**. Aqui, $\vec{N} = (A, B, C)$ e D é o quarto parâmetro da equação do plano. A posição P é a mesma para o plano e caminho do raio se

$$\vec{N} \cdot (P_0 + s\hat{u}) = -D$$

e a distância a partir da posição inicial do raio até o plano é

$$s = -\frac{D + \vec{N} \cdot P_0}{\vec{N} \cdot \hat{u}}$$

Isto nos dá uma posição no plano infinito que contém a face do poliedro, mas esta posição pode não estar contida na fronteira do polígonos que delimita a face do poliedro (Figura 8.22). Então precisamos realizar um teste “dentro-fora” para determinar se o raio intercepta a região interna à face testada. Realizamos este teste para cada face que satisfaz a inequação $\hat{u} \cdot \hat{N} < 0$. Várias faces frontais de um objeto podem ser interceptadas pelo mesmo raio; neste caso a mais próxima será aquela que apresentar a menor distância s calculada. Se não há pontos de interseção internos, o raio não intercepta o objeto.

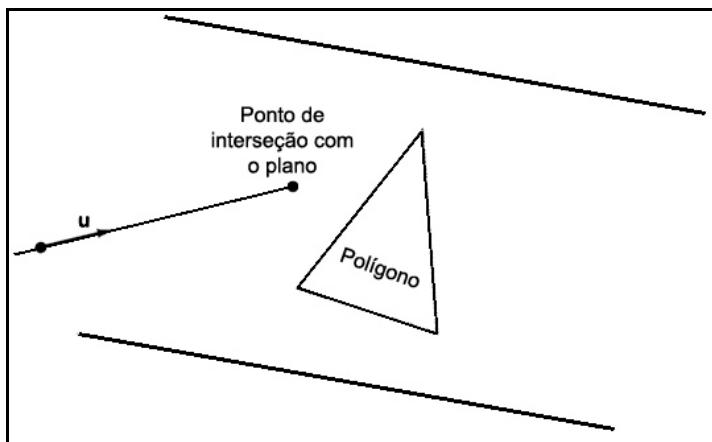


Figura 8.22 – Interseção de um raio com o plano que contém o polígonos