Apostila do Curso de Computação Gráfica

Prof. Dr. André Luiz Battaiola
(Adaptado por: Prof. Dr. José Hiroki Saito)

Departamento de Computação - DC
Universidade Federal de São Carlos - UFSCar
Via Washington Luiz, Km. 235
CP. 676 CEP: 13565-905 São Carlos - SP
fone: 016-260-8232 // fax: 016-260-8233

Índice

AGRADECIMENTOS: Aos alunos **Elson Ambrósio Barbosa**, **Renato Alexandre Bolzan de Paula, Rafael Togami, Luciano Pereira Soares** e **Ana Claudia Stecko Russo** dos cursos de Engenharia em Computação e bacharelado em Ciência da Computação do DC/UFSCar pela ajuda na confecção deste texto.

Objetivo do Curso:

Proporcionar o aprendizado de técnicas e conceitos básicos de computação gráfica, que podem ser utilizados para o desenvolvimento e/ou avaliação e/ou uso de aplicativos gráficos.

Programa Resumido:

1- Conceituação de computação gráfica e áreas de aplicação
2- Dispositivos gráficos e arquiteturas de terminais gráficos
3- Conceituação de aplicativos gráficos.
4- Portatilidade, evolução das bibliotecas gráficas e o padrão GKS.
5- Algoritmos de transformação de primitivas gráficas 3D
   - coordenadas homogêneas
   - transformações: rotação, translação e escalamento
   - projeções planares: paralelas e perspectivas
6- Visualização científica e seus principais conceitos:
   - análise científica
   - evolução das ferramentas gráfica
   - ferramentas avançadas de visualização
   - conceito de eficiência e expressividade gráfica
   - tipos de dados científicos e formas de manipulação pelas ferramentas gráficas
   - processamento distribuido e paralelo em visualização científica
7- Curvas: Bezier e Spline
8- Transformações e recorte em pacotes gráficos
   - algortimos de clipping
   - *pipeline* de transformação

Bibliografia

J. Foley, A. van Dam,. S. Feiner, J. Hughes - Computer Graphics - Principles and Pratices. Addison-Wesley 1990.

D. Hearn, M. Pauline Baker - Computer Graphics. Prentice Hall - Second Edition - 1994

W. M. Newman, R. F. Sproull -Principles of Interactive Computer Graphics. McGraw Hill - 1979

D. F. Rogers et all - Mathematical Elements for Computer Graphics. McGraw Hill - 1976

Romero Tori et all.- Fundamentos de Computação Gráfica.
Livros Técnicos e Científicos Editora – 1987

Jon Q.Jacobs- Delphi Developer’s Guide to OpenGL. Wordware Publishing, Inc. – 1998

Mason Woo, Jackie Neider, Tom Davis & Dave Shreiner (OpenGL Architecture Review Board) - OpenGL Programming Guide, Third Edition The Official Guide to Learning OpenGL, Version 1.2. Addison Wesley,1999

Gomes, J. & Velho, L. - Computação Gráfica – Volume 1. IMPA, RJ, 1998

Esquema de Avaliação

Nfinal = NPV*0,5 + NPJ*0,5

NPV = Valor da Prova_1 ou, caso feita, o valor da Prova Substitutíva

NPJ = Nota do Projeto

1) Conceitos de Computação Gráfica

A computação gráfica pode ser entendida como o conjunto de algoritmos, técnicas e metodologias para o tratamento e a representação gráfica de informações através da criação, armazenamento e manipulação de desenhos, utilizando-se computadores e periféricos gráficos. Em termos de aplicação, ela pode ser dividida atualmente nas seguintes áreas:

CAD
Apresentações Gráficas
Arte . por Computador
Entretenimento
Educação e Treinamento
Visualização Científica

Atualmente a potencialidade da computação gráfica está bastante relacionada com a evolução dos computadores, em termos de hardware e software. Por exemplo, hoje microcomputadores tem performance compatível com alguns tipos de estações de trabalho e no mercado se encontram uma grande variedade de dispositivos gráficos de alta perfomance, tais como, traçadores gráficos, mesas digitalizadoras, *scanners*, *mouses*, *trackballs* e impressoras de alta qualidade. Estes fatores tem permitido que os aplicativos gráficos aumentem acentuadamente a sua versatilidade, capacidade e performance.

Outro fator a se destacar é a grande interação da computação gráfica com outras áreas da ciência (fig. 1.1), caracterizada por uma via de dois sentidos, onde tanto a computação gráfica recebe subsídios como fornece.

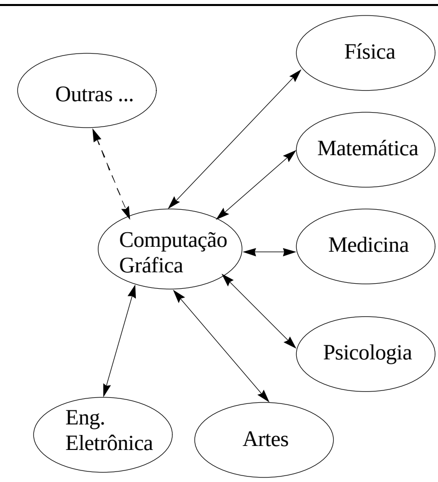


Figura 1.1: Interação da computação gráfica com outras áreas.

A física, por exemplo, forneceu à computação gráfica todos os conceitos da ótica, o que permitiu a elaboração de modelos de cores e de iluminação e preenchimento de areas. A computação gráfica por sua vez forneceu à física as ferramentas de visualização científica que permitem codificar em informação gráfica os dados de seus experimentos.

A matemática está presente de forma acentuada em praticamente todas as áreas do conhecimento, no entanto, somente para dar um exemplo mais específico, no caso da computação gráfica pode se citar os conceitos de algebra linear e de trigonometria utilizados extensivamente. A retribuição da computação gráfica está nas ferramentas de visualização do comportamento de funções matemáticas, por exemplo.

Conforme mais detalhado no item 1.5, a medicina deu a computação gráfica vários conceitos importantes relacionados a forma como o olho capta uma imagem. A computação gráfica fornece à medicina ferramentas de visualização de partes ou de todo o corpo humano.

A psicologia forneceu à computação gráfica importantes conceitos relacionados a forma de interpretação de uma informação gráfica, os quais serão estudados no capítulo sobre visualização científica. A computação gráfica deu a psicologia mecanismos para ela explorar mais eficientemente a comunicação visual objetivando, por exemplo, a implementação de ferramentas educacionais.

A engenharia eletrônica deu a computação gráfica a tecnologia para a produção dos dispositivos gráficos. Através da computação gráfica, a engenharia eletrônica ganhou ferramentas de auxílio para o desenvolvimento de placas de circuito impresso e de pastilhas de circuito integrado.

1.1 Conceitos de Aplicativo Gráfico

Uma aplicativo gráfico (fig. 1.2) é um programa ou um sistema composto de vários programas que permitem a geração de uma determinada apresentação gráfica que pode ser composta de recursos 2D e/ou 3D e/ou imagem. Atualmente, os aplicativos gráficos em geral dispõem de recursos de interação homem-máquina que permitem a composição ou a geração interativa de uma apresentação gráfica.

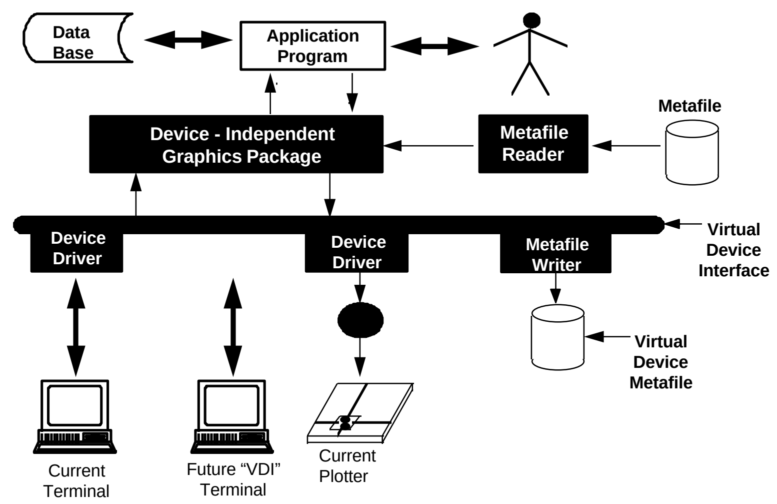


Figura 1.2: Esquema de um aplicativo gráfico

Na implementação de um aplicativo gráfico três parâmetros são importantes de consideração:

1) Sistema Operacional
   Isto implica em priorizar o desenvolvimento para estações de trabalho com sistema operacional UNIX ou microcomputadores com ambiente Windows, OS2, Mac, etc.

2) Biblioteca Gráfica
   A biblioteca gráfica contem as funções que o aplicativo aciona para a geração da primitivas gráficas.
   Um importante fator relacionado as bibliotecas gráficas é o da portabilidade do aplicativo em termos de ambiente computacional e de dispositivos, ou seja, a biblioteca gráfica a ser utilizada deve ser passível de acesso na maior gama possível de computadores e garantir a sua independência em relação aos dispositivos gráficos, permitindo que através de pequenos programas denominados *device drivers* se realize a comunicação entre a biblioteca e os dispositivos, permitindo, assim, que a substituição de um dispositivo gráfico seja feita de forma simples.
   O implementador de qualquer sistema gráfico tem a sua disposição um enorme conjunto de bibliotecas gráficas:Core, Dore, XFDI, Starbase, GKS, PHIGS, GL, OpenGL, etc, no entanto, como atualmente a biblioteca gráfica OpenGL tem se tornado um padrão de fato, dado que os grandes produtores de software e hardware tem priorizado o seu uso, a implementação de qualquer aplicativo gráfico deve considerar prioritariamente a sua utilização.
   Outra biblioteca que está se tornando bastante importante é a DirectX da Microsoft, dado que ela é suportada por uma empresa líder no mercado mundial de software.

3) Sistema de enjanelamento

Em se tratando de ambiente UNIX e com sistema XWindow de enjanelamento, o implementador se depara com dois estilos de interface: Motif e Open View. O Motif, adotado como padrão pela OSF (*Open Software Foundation*), tem sido atualmente bastante considerado pelos implementadores de aplicativos gráficos.

No caso de microcomputadores, os sistemas de enjanelamento já tem interfaces únicas associadas ao ambientes operacionais Windows, Mac, OS2, etc.

Estes conceitos serão analisados novamente no capítulo sobre o GKS.

1.2 Conceitos de CAD

A palavra CAD (*Computer Aided Design*) se refere ao processo de se utilizar um computador para auxiliar no projeto dos mais variados produtos, tais como, um edifício, um automóvel, um návio, uma espaçonave, um eletrodoméstico, um tapete, uma roupa, etc.

O objetivo principal de uma ferramenta CAD é permitir a elaboração do projeto de um produto com maior qualidade, rapidez e precisão. Em termos industriais, isto significa aumentar a qualidade e a eficiência de um produto e diminuir o seu custo de produção.

Sistemas CAD tem por objetivo auxiliar o projetista em todo ou em parte do processo de criação, manipulação e representação de desenhos e projetos. São implementados através de conceitos de computação gráfica, tais como, interfaces gráficas interativas, técnicas de geração de gráficos 2D ou 3D, *rendering*, animação, etc.

Em geral, sistemas CAD funcionam de forma integrada a sistemas CAM (*Computer Aided Manufacturing*) que disponibilizam recursos para a transformação do produto projetado no sistema CAD em códigos de controle das máquinas responsáveis pela manufatura do mesmo.

Atualmente, sistemas CAD/CAM sofisticados fazem parte de um conjunto maior de programas denominado de sistema de gerenciamento de produção industrial (Management Information System, MIS), cuja função é gerenciar todas as etapas de produção, o que envolve controle de estoque, de máquinas na linha de montagem, de inspeção de qualidade, etc.

1.3 Arquiteturas dos Terminais Gráficos

No início do desenvolvimento dos terminais gráficos, o modo de operação destes dispositivos era semelhante ao de um osciloscópio, dado que para produzir uma linha, o feixe de elétrons sensibilizava de forma contínua a tela. Devido a persistência luminosa do material que reveste a tela, a imagem se mantinha ativa até o próximo ciclo de redesenho, e devido a forma de geração de uma linha na tela, estes terminais foram denominados de terminais vetoriais (fig. 1.3).

Dado a deficiência tecnológica da época, traduzida pelo alto preço das memórias e sua baixa capacidade de armazenamento, bem como a baixa qualidade dos terminais, esta arquitetura era a mais conveniente devido ao uso pequeno de memória e a simplicidade dos terminais vetoriais. A memória do terminal, denominada de *refresh buffer*, armazenava comandos simples que indicavam ao controlador de exibição onde traçar os segmentos de reta que compunham a imagem total.

Os dois maiores problemas desta arquitetura eram: a qualidade ruim das imagens geradas e o fato de que imagens complexas começavam a piscar devido a baixa taxa de redesenho. A qualidade ruim da imagem advinha do fato de que gerar áreas coloridas não era uma operação natural para estes terminais e, usualmente, uma imagem complexa é composta de áreas coloridas. Uma imagem complexa também é usualmente composta de várias primitivas gráficas, o que significa uma maior quantidade de elementos para serem armazenados no *refresh buffer*. Assim, se a taxa de redesenho não for suficientemente alta, a imagem gerada começa a piscar.

Paralelamente à popularização da televisão, o que diminuiu o preço dos televisores, e a evolução da microeletrônica, o que permitiu aumentar a capacidade de armazenamento das memórias e diminuir o seu custo, surgiram os terminais gráficos com arquitetura por varredura (*raster terminals*). Neste caso, o terminal pode ser encarado como uma matriz de pontos que podem estar acesos ou não, para tanto o feixe de elétrons varre a tela horizontalmente em um movimento conjugado com um deslocamento vertical, atingindo os pontos que devem ser acesos (fig. 1.4). O controlador de vídeo controla a ativação de cada ponto na tela, através de uma matriz de pontos, onde está armazenado o estado de cada um deles. Assim, o tamanho da memória de pontos é proporcional a resolução da tela. A modificação da imagem gerada é realizada através da alteração dos valores da matriz de pontos, o que é realizado pelo controlador de exibição, sob o controle da CPU. Este tipo de arquitetura é a mais utilizada atualmente e os modernos terminais por varredura permitem a geração de imagens com altíssima qualidade e grande variedade de elementos.

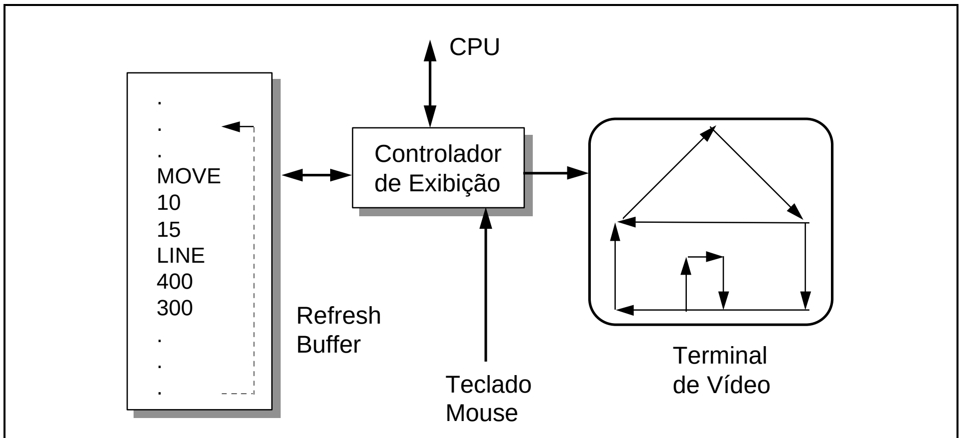


Figura 1.3 - Terminal vetorial

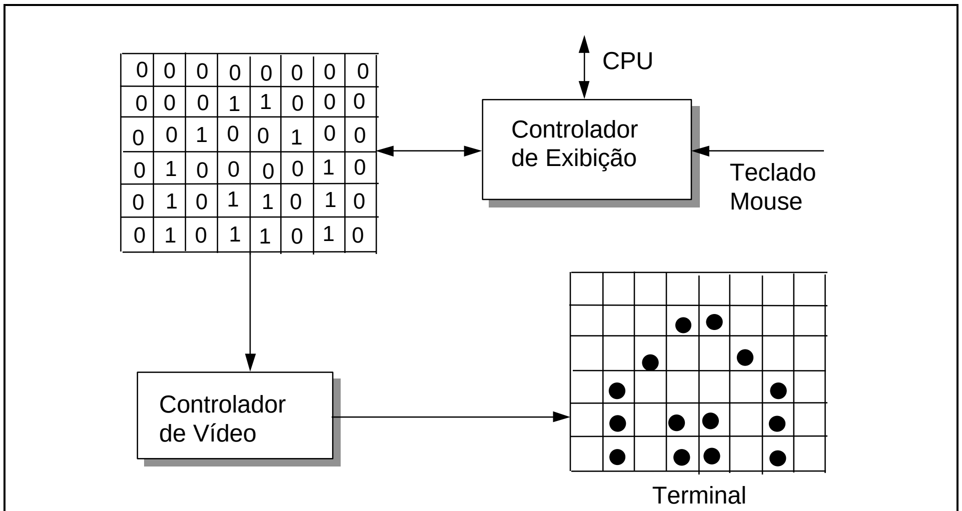


Figura 1.4 - Terminal por varredura (*Raster*).

A arquitetura mais comum para um terminal gráfico é exibida na figura 1.5, onde o *frame buffer* é armazenado na memória local do sistema. Como neste caso a transmissão do conteúdo do frame buffer para o controlador de vídeo é feita através do barramento central, tanto a CPU como o barramento ficam ocupados com uma operação menos nobre em termos computacionais.

A arquitetura mais comum dos terminais por varredura é exibida na figura 1.6. Neste caso, uma porção da memória do sistema serve como *frame buffer* e o controlador de vídeo exibe a imagem armazenada acessando esta porção da memória através de uma porta separada. A razão de acesso é definida pela taxa de varredura. Em muitos sistemas, uma porção fixa da memória é alocada permanentemente para o *frame buffer*, no entanto, outros sistemas tem várias áreas de memória intercambiáveis. Em microcomputadores estas porções de memórias são chamadas de páginas e, em geral, duas delas são usadas como *frame buffer*, o que permite se implementar um esquema de *double buffer*, que consiste em manter uma página ativa, ou seja, em exibição, e a outra desativada. A escrita de uma nova imagem é feita na página desativada e, ao final da escrita, ela se torna a página ativa. Isto permite tornar mais suave o processo de animação de um conjunto de imagens, dado que o observador não visualiza a escrita de cada imagem.

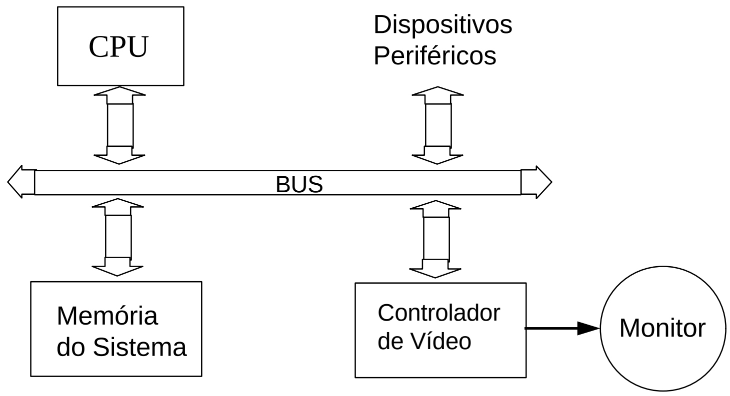


Figura 1.5 - Arquitetura simples.

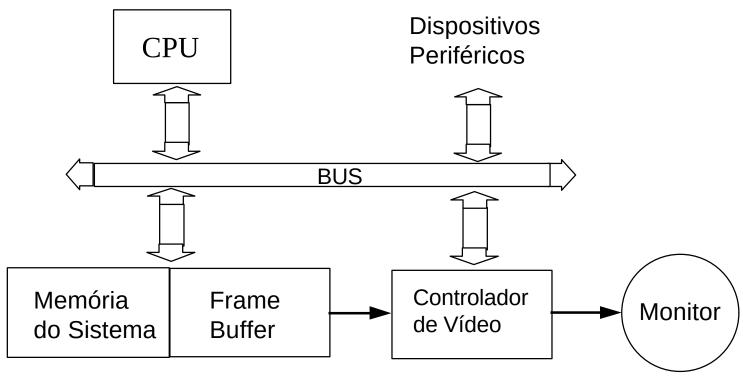


Figura 1.6 - Arquitetura comum.

Algumas operações realizadas pela CPU e relacionadas com a manipulação dos pixels são custosas em termos de tempo para serem realizadas somente pela CPU. Por exemplo, a operação de *scan conversion* (determinação de endereço em memória de um determinado pixel) ou as operações *raster* (cópia, movimentação ou combinação de áreas retangulares de pixels) consomem tempo, apesar da simplicidade do procedimento. Desta forma, é possível se adicionar um processador gráfico à arquitetura, de maneira que estas tarefas possam ser realizadas por este processador, aliviando a carga sobre a CPU. Estes processadores também podem realizar outras tarefas como, por exemplo, gerar primitivas gráficas (linhas, arcos, etc) ou operações mais sofisticadas (linhas com *anti-aliasing*, *rendering*, etc). A figura 1.7 mostra como fica a arquitetura da figura 1.6 acrescida de um processador gráfico.

Certamente o valor <u>0</u> ou <u>1</u> para um pixel não é suficiente para se produzir uma imagem de qualidade. Assim, a pergunta é: quantos bits por pixeis são necessários para se gerar uma imagem com qualidade satisfatória, bem como garantir uma transição suave de cores ou tons de cinza? A resposta é que 5 ou 6 bits pode ser suficiente, mas muitas vezes se necessita 8 ou mais bits. No caso de terminais coloridos o usual é 8 bits por cor primária: vermelho, verde e azul, cuja sigla em inglês é RGB.

Mesmo atualmente com a queda do preço das memórias RAM, é custoso se montar uma memória de vídeo com 24 bits por pixel, além disto, é incomum se utilizar 2^{24} diferentes cores em uma imagem. Por outro lado, muitas vezes é interessante se observar uma imagem onde a associação de cores com os pixels possa ser mudada. Desta forma, muitos terminais utilizam o esquema da tabela de consulta de cores, também conhecida como LUT (*look-up table*). Por este processo, cada pixel tem um valor de

8 bits associado, o que constitui um índice para uma tabela de cores de 256 posições (figura 1.8). Nesta posição está armazenado o valor da cor que decomposta em frações de suas componentes básicas, ou seja, as cores primárias RGB, caracterizarão o pixel na tela. Note-se que no caso da figura 1.8 é possível se trabalhar com 2^{12} cores diferentes, no entanto, somente 256 diferentes cores poderão ser exibidas simultaneamente.

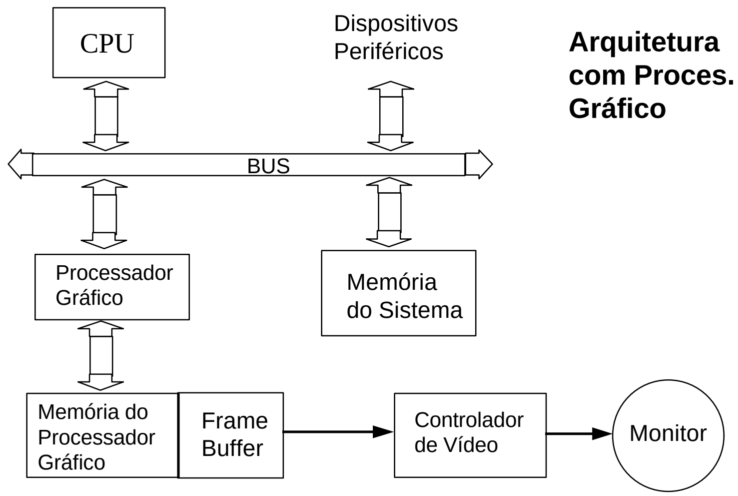


Figura 1.7 - Arquitetura com processador gráfico.

A definição de características dos terminais gráficos deve ser baseada nas propriedades do olho humano, dado que ele é o dispositivo que irá transmitir uma imagem para o cérebro, onde será interpretada.

Um esquema simplificado do olho humano é exibido na figura 1.9. Note-se neste esquema a retina, que é uma membrana que reveste a parte interna do olho é o local de formação da imagem. Ela é formada basicamente por 2 classes de receptores de imagens:

1) cones - 6 a 7 milhões - muito sensíveis a níveis altos de luminosidade e a cores
2) bastonetes - 75 a 150 milhões - sensíveis a baixos níveis de luminosidade

A fóvea é a parte central da retina. Ela é formada principalmente de cones e é responsável pela distinção de detalhes finos de uma imagem. Note-se que o olho humano distingue melhor cores em ambientes bem iluminados que são adequados a ação dos cones. Ao contrário, em ambientes de pouca luz, onde os bastonetes atuam, o nível de distinção de cores é menor.

Em termos da resolução de uma imagem, os olhos também ditam os parâmetros adequados. Por exemplo, a resolução de uma imagem de TV é da ordem de 512 x 384 pixels, em largura x altura, que é um valor relativamente baixo e origina imagens de qualidade média. A resolução dos terminais gráficos modernos é de 1280 x 1024, o que permite a exibição de imagens com boa qualidade e, por isto, será adotada como a resolução das TVs digitais. Resoluções maiores do que esta esbarram no limite da capacidade do olho humano em distinguir detalhes, ou seja, o efeito visual de resoluções muito maiores do que 1280 x 1024 podem ser imperceptíveis para o olho humano.

As características dos olhos também influenciam em termos de níveis de cores. Por exemplo, na exibição de uma faixa de dégradé de tons de cinza variando do branco até o preto, quantos tons são necessários para que se observe a faixa com uma transição suave de tons, ou seja, sem a distinção das linhas de mudança de tons? A resposta é um valor de no mínimo 100 tons. Em função de dados deste tipo, bem como do número mínimo de cores necessárias para se compor uma imagem de boa qualidade, os terminais modernos permitem a exibição simultânea de 128 ou 256 cores.

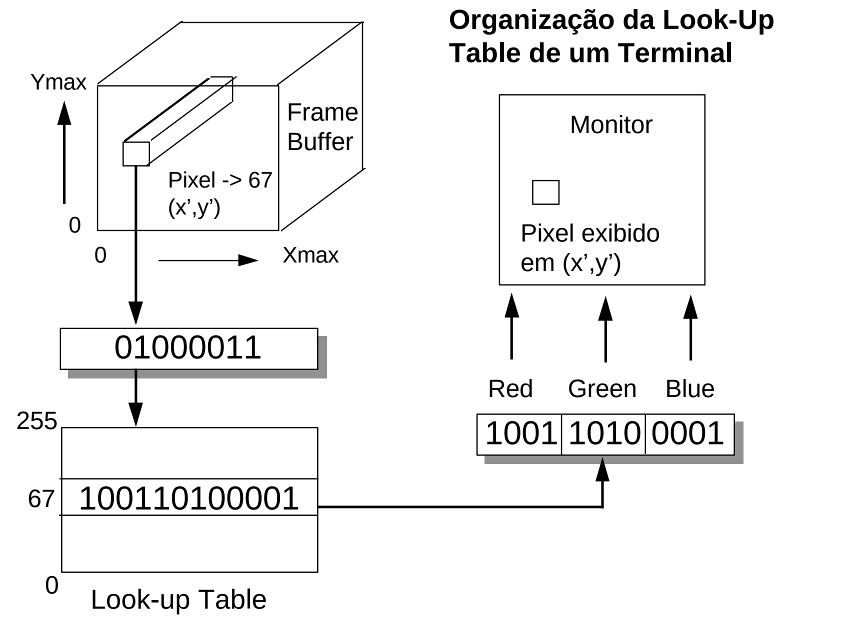


Figura 1.8: Operação da Tabela de Cores (LUT).

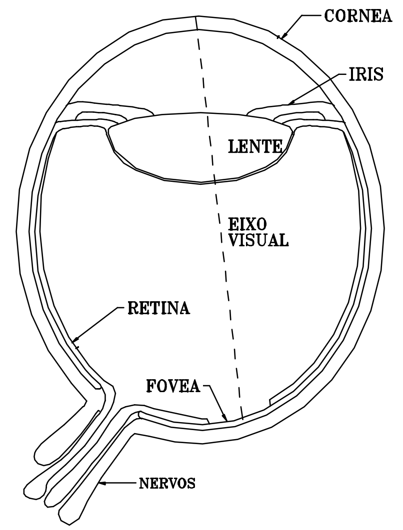


Figura 1.9: Esquema simplificado do olho humano.

2. Transformações Afins

Uma transformação de coordenadas da forma:

$$\vec{v}' = A\ \vec{v} + \vec{b}$$

ou

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \qquad \text{(eq. 2.1)}$$

é denominada uma transformação “afim”. Neste caso, as coordenadas (x’,y’,z’) do vetor $\vec{v}'$, que definem um ponto no espaço, são uma função linear de (x,y,z) e a_{ij} e b_i são constantes determinadas pelo tipo de transformação. As transformações afim têm a função de modificar a posição dos pontos no espaço, ou dos objetos no espaço. Essas transformações tem a característica geral de transformar linhas paralelas em linhas paralelas e mapear pontos finitos em pontos finitos. O grupo de transformações afins do espaço define a geometria afim, que estuda as razões e proporções entre objetos geométricos. Note-se que em geometria afim, paralelismo é um conceito importante, sendo relações entre linhas paralelas uma parte substancial da geometria e os teoremas da geometria afim são idênticos aos da geometria euclidiana.

Rotação, translação, escalamento, espelhamento e cisalhamento são exemplos de transformações afins detalhados a seguir.

2.1 Translação

A translação, alteração da posição de um ponto através da soma de constantes de deslocamento as suas coordenadas, é normalmente aplicada sobre todos os pontos de uma figura, de maneira a possibilitar a sua movimentação no espaço (fig. 2.1). O exemplo clássico em computação gráfica de aplicação desta transformação é a função *pan*, disponível em vários sistemas gráficos.

Em termos de transformação afim, a translação corresponde à soma de um vetor de deslocamento ao vetor que define o ponto que se deseja deslocar. Assim, na equação 2.1, o vetor com as componentes $\boldsymbol{b_i}$ corresponde ao vetor de deslocamento.

2.2 Escalamento (Mudança de Escala - *Scaling*)

A Mudança de Escala corresponde à multiplicação das coordenadas de um ponto por valores iguais ou diferentes. É normalmente aplicada sobre todos os pontos de uma figura com o objetivo de ampliar ou reduzir a sua dimensão ou então distorcer a sua forma geométrica (fig. 2.2). O uso clássico desta operação em computação gráfica é a função *zoom in* (ampliação) ou *zoom out* (redução).

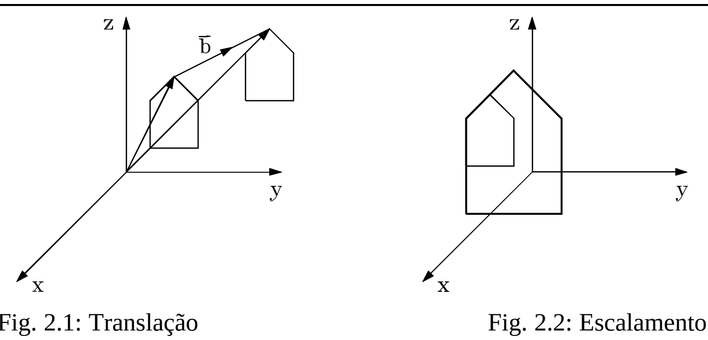


Fig. 2.1: Translação

Fig. 2.2: Escalamento

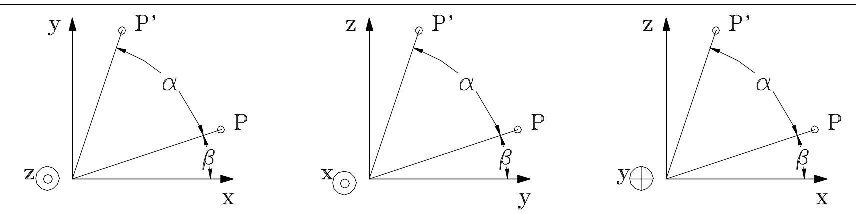
Quando somente a operação de escalamento é realizada, a matriz *A* na equação 2.1 fica a matriz E, onde $\boldsymbol{e_x}$, $\boldsymbol{e_y}$, $\boldsymbol{e_z}$, são os fatores de escala das coordenadas $\boldsymbol{x}$, $\boldsymbol{y}$ e $\boldsymbol{z}$, respectivamente. Observa-se facilmente que a aplicação desta matriz sobre o vetor de coordenadas gera o vetor escalado $\vec{v}$, conforme descrito pelas equações 2.2 e 2.3, a seguir.

$$E = \begin{bmatrix} e_x & 0 & 0 \\ 0 & e_y & 0 \\ 0 & 0 & e_z \end{bmatrix} \quad \text{(eq.2.2)} \qquad\qquad \vec{v} = \begin{bmatrix} e_x x \\ e_y y \\ e_z z \end{bmatrix} \quad \text{(eq.2.3)}$$

2.3 Rotação

A rotação é o giro de um determinado ângulo de um ponto em torno de um ponto de referência, sem alteração da distância entre eles. Esta operação é aplicada normalmente sobre todos os pontos de uma figura, o que possibilita que ela seja rotacionada. Vários programas gráficos dispõem desta operação, sendo que alguns restringem o ângulo de rotação a valores fixos, tais como, 90° e 180°.

Para o cálculo da matriz de rotação, será considerado inicialmente apenas duas coordenadas, por exemplo, **x** e **y**. Assim, na figura 2.3a, o ponto **P**, de coordenadas (x,y), será rotacionado de um ângulo α em torno do eixo **z**, até a posição do ponto **P’** (x’,y’). A linha que une o ponto **P** a origem do sistema de coordenadas está rotacionada de um ângulo β em relação ao eixo **x**.



Figuras 2.3a, 2.3b, 2.3c - Rotações em torno dos eixo z, x e y, respectivamente.

Supondo-se que a distância do ponto **P** à origem seja “D”, tem-se:

$$x = D\cos(b) \quad \text{(eq. 2.2)} \qquad x' = D\cos(a+b) \quad \text{(eq. 2.4)}$$

$$y = D\,\mathrm{sen}(b) \quad \text{(eq. 2.3)} \qquad y' = D\,\mathrm{sen}(a+b) \quad \text{(eq. 2.5)}$$

Da trigonometria tem-se:

$$\cos(a+b) = \cos(b)\times\cos(a) - \mathrm{sen}(b)\times\mathrm{sen}(a) \quad \text{(eq. 2.6)}$$

$$sin(a+b) = \cos(b)\times\mathrm{sen}(a) + \mathrm{sen}(b)\times\cos(a) \quad \text{(eq. 2.7)}$$

Usando-se as equações 2.2, 2.3, 2.6 e 2.7 nas equações 2.4 e 2.5 tem-se:

$$x' = x\cos(a) - y\,\mathrm{sen}(a) \quad \text{(eq. 2.8)}$$

$$y' = x\,\mathrm{sen}(a) + y\cos(a) \quad \text{(eq. 2.9)}$$

Em forma matricial, as equações 2.8 e 2.9 ficam:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(a) & -\mathrm{sen}(a) \\ \mathrm{sen}(a) & \cos(a) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{(eq. 2.10)}$$

Similarmente, a fórmula 2.10 se aplica as rotações das figuras 2.3b e 2.3c. No entanto, para se estender esta fórmula para rotações tridimensionais, deve-se considerar primeiramente o problema da orientação dos eixos. Note-se que:

$$v\vec{z} = v\vec{x} \times v\vec{y} \quad \text{(eq. 2.11)} \qquad v\vec{x} = v\vec{y} \times v\vec{z} \quad \text{(eq.2.12)}$$

$$-v\vec{y} = v\vec{x} \times v\vec{z} \quad \text{(eq. 2.13)}$$

Neste caso, $v\vec{x}$, $v\vec{y}$ e $v\vec{z}$ são versores nas direções **x**, **y** e **z**, respectivamente. Nas figuras 2.3a, 2.3b e 2.3c os resultados das equações 2.11, 2.12 e 2.13 são indicados pelo conteúdo das circunferências posicionadas do lado esquerdo dos eixos. Sendo o conteúdo uma circunferência menor, o eixo resultante do produto vetorial dos eixos referenciados no plano tem sinal positivo, o que implica que o seu sentido é do plano para o leitor. Se o conteúdo é uma cruz, o sinal é negativo, sentido inverso. Assim, para manter a consistência, a aplicação da fórmula matricial 2.10 deve considerar que os eixos referenciados no plano da figura 2.3c estão invertidos, sendo o correto:

$$\begin{bmatrix} z' \\ x' \end{bmatrix} = \begin{bmatrix} \cos(a) & -\operatorname{sen}(a) \\ \operatorname{sen}(a) & \cos(a) \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \qquad \text{(eq. 2.14)}$$

Considerando-se as figuras 2.3a, 2.3b e 2.3c e as fórmulas 2.10 e 2.14, pode-se facilmente deduzir as fórmulas para o cálculo no espaço tridimensional da rotação de um ponto **A** para o ponto **A’**, sendo o plano de rotação perpendicular ao eixo **z** (fig. 2.4), **y** (fig. 2.5) e **x** (fig. 2.7).

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(a) & -\operatorname{sen}(a) & 0 \\ \operatorname{sen}(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \text{(eq. 2.15)} \qquad \text{Rotação em torno do eixo } \mathbf{z}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(a) & 0 & \operatorname{sen}(a) \\ 0 & 1 & 0 \\ -\operatorname{sen}(a) & 0 & \cos(a) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \text{(eq. 2.16)} \qquad \text{Rotação em torno do eixo } \mathbf{y}$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\operatorname{sen}(a) \\ 0 & \operatorname{sen}(a) & \cos(a) \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \text{(eq. 2.17)} \qquad \text{Rotação em torno do eixo } \mathbf{x}$$

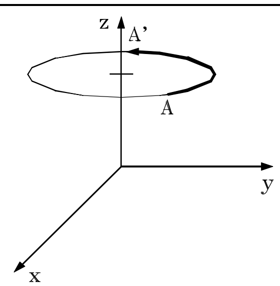


Fig. 2.4: Rotação em torno do eixo “z”

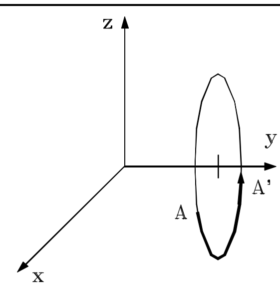


Fig. 2.5: Rotação em torno do eixo “y”

2.4 Espelhamento

Uma operação bastante conhecida em computação gráfica é o espelhamento, a qual consiste em rotacionar um objeto em torno de um eixo de tal maneira que os pontos do objeto na posição original e na rotacionada mantenham a mesma distância em relação a um linha de referência, caso bidimensional, ou a um plano de referência, caso tridimensional. Na figura 2.7, há o espelhamento de um objeto em torno do eixo **y** em relação ao plano **xy** e em torno do eixo **z** em relação ao plano **xz**. Em ambos os casos, o ângulo de rotação é 180°.

2.5 Rotação em torno de um eixo genérico

A matriz de rotação em torno de um eixo genérico (fig. 2.8) não é complexa, porém trabalhosa em termos de dedução, assim, será apresentado apenas a matriz **MGR** que permite esta operação. Sendo **N** um vetor <u>unitário</u> de coordenadas (x,y,z) e **θ** o ângulo de rotação, tem-se:

$$MGR = \begin{bmatrix} tx^2 + c & txy - sz & txz + sy \\ txy + sz & ty^2 + c & tyz - sx \\ txz - sy & tyz + sx & tz^2 + c \end{bmatrix} \qquad \text{(eq. 2.18)}$$

Onde:

x,y,z = coordenadas de N t = 1- cos(θ)

s = sin(θ) c = cos(θ)

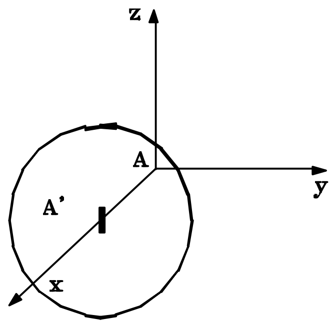


Fig. 2.6: Rotação em Torno do Eixo “x”

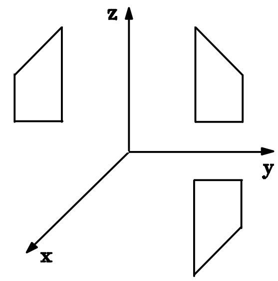


Fig. 2.7: Espelhamento

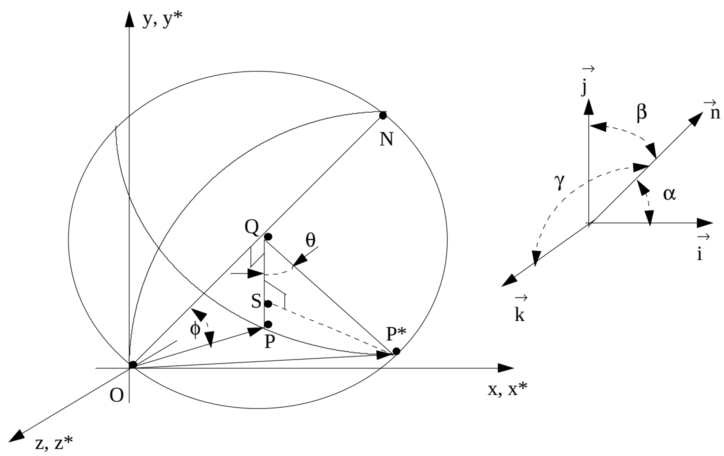


Figura 2.8: Determinação da matriz de rotação em torno de um eixo genérico.

2.6 Cisalhamento

Outra transformação afim importante de ser estudada é o cisalhamento (*shear)*, cujo exemplo clássico para o sistema de coordenadas bidimensional que explica a sua função é o da italização de um caracter (fig. 2.9). Neste caso, há uma variação no valor da coordenada **x** em função do valor da **y** (fig. 2.9 i_1 e i_2), sendo *MTS_1* a matriz de transformação correspondente (eq.2.19). Pode-se associar uma outra transformação a de cisalhamento, como, por exemplo, o escalamento da coordenada **y** (fig. 2.9 i_3), conforme exemplificado em *MTS_2*, (eq.2.20). A matriz *MTS_3* (eq.2.21) ilustra o uso desta transformação para o caso tridimensional.

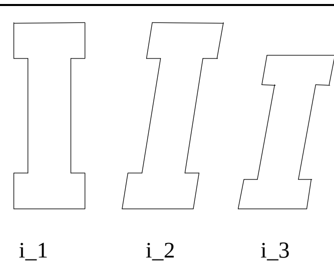


Figura 2.9: Exemplo de cisalhamento

$$\text{MTS\_1} = \begin{bmatrix} 1 & sh_x \\ 0 & 1 \end{bmatrix} \quad \text{(eq.2.19)} \qquad \text{MTS\_2} = \begin{bmatrix} 1 & sh_x \\ 0 & e_y \end{bmatrix} \quad \text{(eq.2.20)} \qquad \text{MTS\_3} = \begin{bmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{bmatrix}$$

(eq.2.21)

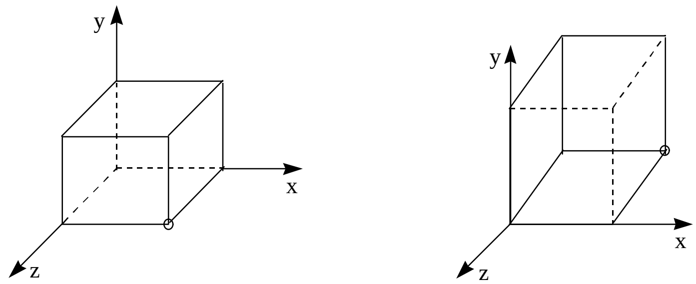


Figura 2.10: Exemplo de cisalhamento no caso 3D.

2.5 Composição de Transformações

Uma importante questão que sempre deve ser considerada com relação as transformações afins se refere a sua composição. Neste caso, a ordem em que elas são executadas pode alterar o resultado final esperado. Considere-se então duas transformações afins, uma somente de rotação de **45°** em torno do eixo **z** e outra somente de translação de valor **Δx** ao longo do eixo **x**. Como a rotação é realizada em relação a origem do sistema de coordenadas, considerando-se a figura 2.11.a, a aplicação primeiro da rotação e depois da translação resulta na figura 2.11.b e o inverso na figura 2.11.c.

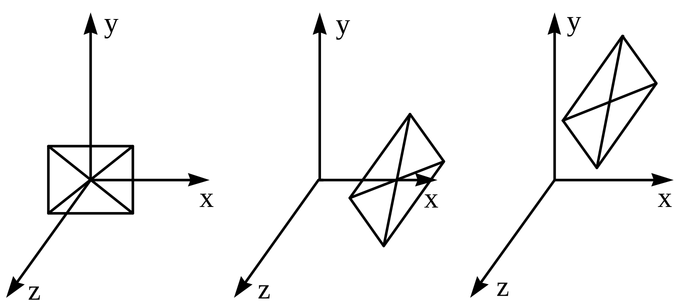


Figura 2.11.a Figura 2.11.b Figura 2.11.c

Figura 2.11: Composição de transformações.

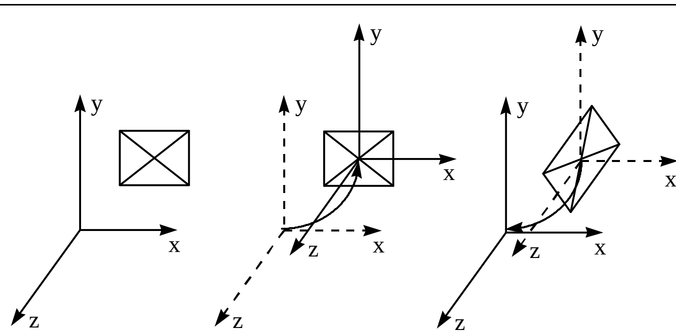


Figura 2.12: Mudança de centro de rotação.

Neste sentido, caso se deseje rotacionar um objeto no espaço em torno de um ponto interno a ele, deve se primeiramente deslocar o centro de rotação (origem dos eixos) para este ponto, proceder a rotação e posteriormente voltar o centro de rotação a sua posição inicial (fig. 2.12). Note-se que isto equivale a deslocar o objeto para o centro de coordenadas.

3. Coordenadas Homogêneas

Como detalhado no capítulo 2, uma transformação afim segue a forma:

$$\vec{v}' = M \vec{v} + \vec{b} \qquad \text{(eq. 3.1)}$$

Esta formulação é, em termos de cálculo, bastante inconveniente para se determinar as coordenadas do vetor final, após uma série de transformações de um vetor inicial. A formulação da equação 3.4 é muito mais conveniente porque permite que o cálculo de múltiplas transformações seja realizado calculando-se a matriz de transformação resultante e aplicando-se esta matriz sobre o vetor (eq. 3.5).

$$\vec{v}_1 = M_1 \vec{v} + \vec{b}_1 \qquad \text{(eq. 3.2)}$$

$$\vec{v}_2 = M_2 M_1 \vec{v} + M_2 \vec{b}_1 + \vec{b}_2 \qquad \text{(eq. 3.3)}$$

$$\vec{v}' = M \vec{v} \qquad \text{(eq. 3.4)}$$

$$\vec{v}' = M_n M_{n-1} \cdots M_1 \vec{v} \qquad \text{(eq. 3.5)}$$

A impossibilidade de se inserir a transformação de translação na matriz M não permite que a formulação 3.5 seja usada, assim, é necessário se encontrar um mecanismo que contorne este problema. A solução usual para este problema é a alteração do espaço de coordenadas de dimensão 3 para 4, de forma controlada, de maneira que a dimensão da matriz M se altere de 3x3 para 3x4 e, assim, ela possa incorporar a transformação de translação (eq. 3.6). Como é inconveniente operar uma matriz não quadrada por não permitir, por exemplo, o cálculo da inversa, a matriz M_T pode ser novamente modificada pela inserção de mais uma linha que não altere o resultado final (eq. 3.7).

(eq. 3.6)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & l \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+l \\ y+m \\ z+n \\ 1 \end{bmatrix}$$

(eq. 3.7)

$$M_T = \begin{bmatrix} 1 & 0 & 0 & l \\ 0 & 1 & 0 & m \\ 0 & 0 & 1 & n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Considerando-se uma matriz M genérica de dimensão 4x4, quando ela é aplicada sobre o vetor $\vec{v}$, em coordenadas homogêneas, gera o vetor $\vec{v}\,*$ descrito a seguir: Para se calcular o vetor desejado, normaliza-se o vetor $\vec{v}\,*$. Assim, supondo-se $H \neq 0$, tem-se $\vec{v}\,'$.

$$\vec{v}\,* = M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} \Rightarrow \quad \vec{v}\,' = \begin{bmatrix} X/H \\ Y/H \\ Z/H \\ 1 \end{bmatrix}$$

Neste contexto, uma propriedade interessante e bastante útil das coordenadas homogêneas se refere a representação de um ponto no infinito. Assim, considere-se o ponto sobre o eixo **x** indicado pelo vetor $\vec{v}\,*$ e $\vec{v}\,'$:

$$\vec{v}\,* = \begin{bmatrix} A \\ 0 \\ 0 \\ H \end{bmatrix} \Rightarrow \quad \vec{v}\,' = \begin{bmatrix} A/H \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Quando $H \to 0$, $A/H \to \infty$, logo a aplicação de uma matriz de transformação em coordenadas homogêneas resulta em um ponto no infinito quando o seu vetor posicional for do tipo $\vec{v}\,*$ descrito a seguir. Em geral, utiliza-se os vetores infinitos $\vec{x}^{\,*}_{\infty}$, $\vec{y}^{\,*}_{\infty}$ e $\vec{z}^{\,*}_{\infty}$.

$$\vec{v}^{\,*} = \begin{bmatrix} A \\ B \\ C \\ 0 \end{bmatrix} \qquad \vec{x}^{\,*}_{\infty} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \vec{y}^{\,*}_{\infty} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad \vec{z}^{\,*}_{\infty} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Estes vetores infinitos serão, em especial, usados na determinação de pontos-de-fuga no caso de projeções planares perspectivas. Outras propriedades das coordenadas homogêneas serão exploradas posteriormente. Também posteriormente será mostrado que a matriz de transformação M de dimensão 4x4, pode ser particionada em 4 quatro partes:

$$M = \begin{bmatrix} 3x3 & 3x1 \\ 1x3 & 1x1 \end{bmatrix}$$

1x3:	produz projeção perspectiva	1x1:	produz escalamento global
3x3:	produz uma transformação afim do tipo, rotação e escalamento	3x1:	produz translação

4. Projeções Planares

Dado que a exibição de um objeto 3D em uma tela de computador ou em uma folha de papel exige o mapeamento de um sistema de coordenadas 3D em um 2D, operações de projeção são requeridas. Em geral, entende-se como projeção, o processo de mapear um sistema de coordenadas de dimensão “n” em um de dimensão menor ou igual a “n-1”.

A humanidade utiliza há séculos o conceito de projeções geométricas. O mais antigo exemplo de uso de desenho técnico na história da humanidade data de aproximadamente 2150 A.C. O desenho contém uma planta de um prédio da cidade de Lagash na Mesopotâmia.

De acordo com alusões literárias, os geômetras e pintores gregos da antiguidade clássica estavam familiarizados com as leis da perspectiva. O pintor Agatharchus foi o primeiro a usar perspectivas em larga escala no período de 5 séculos A.C. e escreveu um livro sobre “pintura de cenas”, o que inspirou os filósofos Anaxagoras e Demócrito a escrever sobre perspectiva.

A primeira evidência real do uso de desenhos para guiar edificações foi encontrado nos textos de Vitruvius, um arquiteto e engenheiro romano do período de Júlio César e Augustus, em torno do ano 14 A.C.

Apesar do estudo de gregos e romanos, uma formalização destas técnicas só surgiu durante a Renascença. Os pintores Duccio (1255-1319), pintor do famoso quadro “A Última Ceia”, e Giotto (1276-1336) empreenderam esforços no sentido de representar a terceira dimensão através da perspectiva. Filippo Brunelleschi (1377-1446) foi o primeiro artista a desenvolver um sistema matemático para a perspectiva. O primeiro tratado sobre perspectiva, *Della Pittura*, foi publicado em 1435 por Leone Battista Alberti (1404-1472). No mesmo período, a técnica da perspectiva continuou a ser aperfeiçoada por Piero della Francesca (1420-1492) através do texto *De Prospettiva Pingendi* e por Leonardo da Vinci que pintou a sua versão de “A Última Ceia”.

Gasparad Monge (1746-1818), um desenhista de fortificações militares francesas, foi o primeiro a descrever de forma organizada o uso de projeções em engenharia, o que lhe valeu o título de “pai da geometria descritiva”. Monge publicou a primeira edição do livro *Geometrie Descriptive* em 1801.

Desta época para a atual, as técnicas de projeções continuaram a ser estudadas e aperfeiçoadas e se popularizaram entre profissionais e estudantes de engenharia, artes e arquitetura. Com o surgimento da computação gráfica e com a popularização de sistemas e bibliotecas gráficas, o número de interessados nas técnicas de projeções se ampliou acentuadamente.

Geralmente usuários de sistemas ou bibliotecas gráficas se deparam com operações relacionadas a posicionamento e especificação de câmeras ou então a ajuste de transformações que permitam o correto posicionamento de um objeto em uma cena. Tão comum quanto estas operações é a dificuldade do usuário em realizá-las de forma totalmente segura e compreensível. Outro problema nesta mesma linha que o usuário se depara é quando ele transporta os objetos criados em um determinado sistema gráfico para outro. Comumentemente falta-lhe um conhecimento mais conceitual do modo de funcionamento destas operações.

O objetivo deste curso é conceituar projeções planares, apresentando a sua classificação e formas algébricas de manipulá-las. Com base nestes conceitos, será ilustrado o funcionamento do esquema de transformações em bibliotecas e sistemas gráficos que manipulam primitivas tridimensionais.

As projeções mais consideradas em CG são as que projetam um sistema de coordenadas 3D em um 2D, realizam a projeção em um plano ao invés de uma superfície curva, como também utilizam raios projetores lineares ao invés de curvos (fig. 4.1). Esta classe de projeções é conhecida como “projeções geométricas planares” e podem ser subclassificadas de acordo com o esquema da figura 4.2

4.1 Classificação das Projeções Planares

As projeções planares paralelas e perspectivas diferem com relação a distância do plano de projeção ao centro de projeção. Se a distância é finita, a projeção é perspectiva, se a distância é infinita, a projeção é paralela (fig. 4.3).

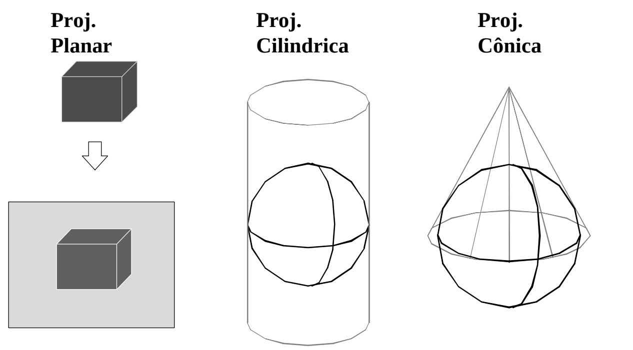


Figura 4.1: Diferentes tipos de projeções

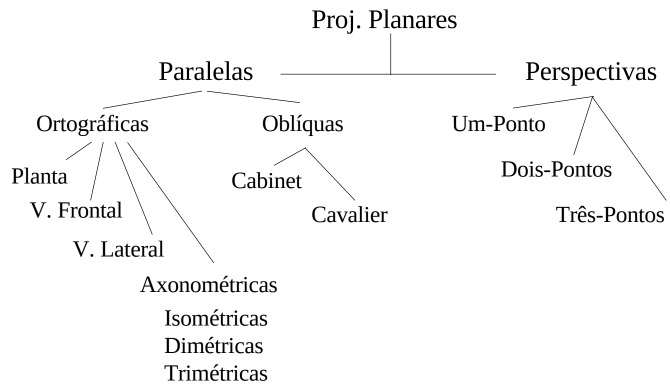


Figura 4.2: Árvore de Classificação das Projeções

4.2 Projeções Planares Paralelas

Projeções paralelas são subclassificadas em ortográficas e oblíquas, dependendo da relação entre a direção dos raios projetores e a normal ao plano de projeção (fig. 4.4). Em projeções ortográficas, as direções são as mesmas. Em projeções oblíquas, são diferentes.

As projeções ortográficas vista lateral, vista frontal e planta constituem as projeções normalmente utilizadas em desenho técnico (fig. 4.5). Elas tem a direção dos raios projetores e a normal ao plano de projeção coincidentes com a direção dos eixos cartesianos. Elas oferecem uma visão parcial do objeto, no entanto, mantém sem alteração as relações de dimensões e ângulos do objeto projetado. Estas projeções são geralmente utilizadas em conjunto, contando também com uma projeção axonométrica ou perspectiva.

As projeções paralelas ortográficas axonométricas tem a direção dos raios projetores e a normal ao plano de projeção coincidentes, porém distintas da direção das normais dos planos dos eixos cartesianos. Desta forma, essas projeções permitem a visualização de várias faces paralelas aos planos cartesianos de uma única vez (fig. 4.6)..

Projeções axonométricas distorcem os objetos, alterando as relações de ângulos e dimensões de lados dos objetos, no entanto, mantém as relações de paralelismo entre eles. A alteração da dimensão dos lados é relacionada com a alteração da dimensão dos versores (vetores unitários) em cada um dos eixos **x**, **y** e **z**, quando projetados no plano. Assim, as projeções axonométricas se subdividem em dimétricas, quando dois versores variam a dimensão igualmente quando projetados no plano; isométricas, quando os três versores variam na mesma proporção (fig. 4.7); e trimétricas, quando os três versores variam de forma diferenciada.

As projeções paralelas oblíquas tem a direção da normal ao plano de projeção distinta da direção dos raios projetores (fig. 4.8). As projeções paralelas oblíquas se subdividem em *cavalier* e *cabinet* (fig. 4.9). Na *cabinet* há um encolhimento na dimensão do versor perpendicular ao plano de projeção para corrigir a ilusão de que o objeto exibido é maior na direção deste versor.

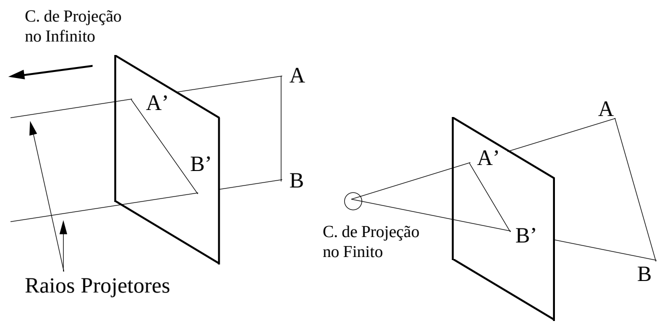


Figura 4.3: Centros de projeções

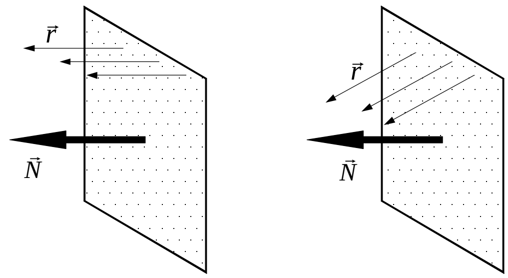


Figura 4.4: Direção da normal ao plano e dos raios projetores.

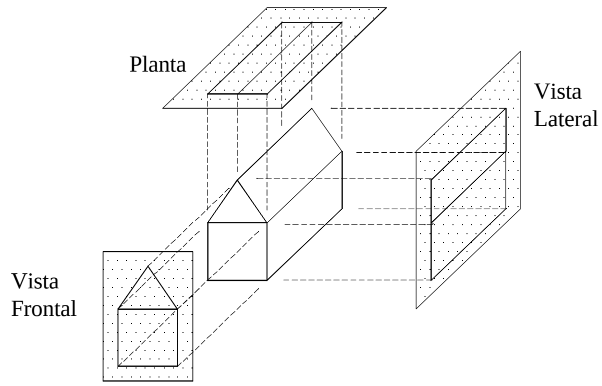


Figura 4.5: Projeções ortográficas planta, vista lateral e frontal.

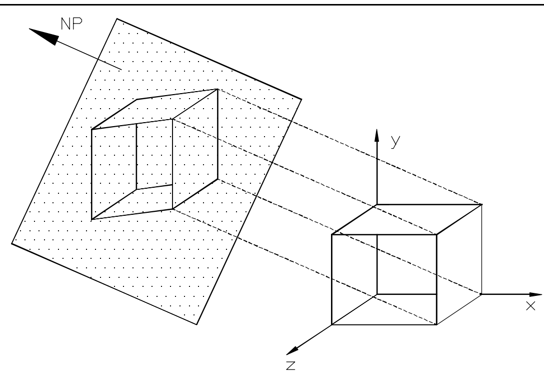


Figura 4.6: Projeções ortográficas axonométricas.

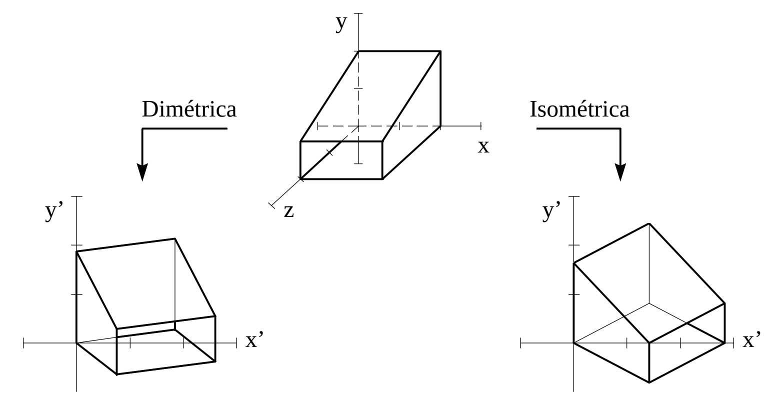


Figura 4.7: Projeções ortográficas axonométricas dimétrica e isométrica.

4.3) Projeções Planares Perspectivas

O efeito visual de uma projeção perspectiva é bastante realista, pois as dimensões de um objeto projetado variam inversamente com relação ao centro de projeção, o que está de acordo com o modo de funcionamento do sistema visual humano. Além disto, como as projeções axonométricas, elas permitem a visualização conjunta de várias faces normais aos eixos **x, y**, e **z**, de um objeto. No entanto, as projeções perspectivas não são úteis para documentar precisamente as formas de um objeto, dado que as dimensões e os ângulos dos seus lados podem sofrer alterações após a projeção. Em especial, pode haver perda do paralelismo entre as linhas.

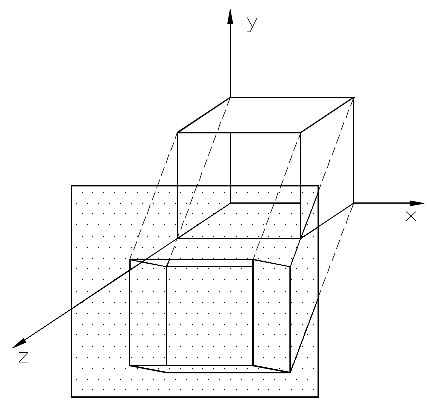


Figura 4.8: Projeções oblíquas.

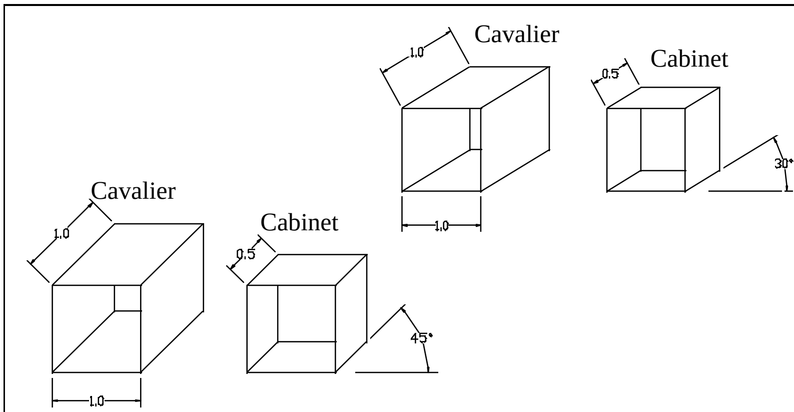


Figura 4.9: Projeções Oblíquas *Cavalier* e *Cabinet*

Como a projeção perspectiva tem o centro de projeção localizado em um ponto finito, ocorre uma distorção no objeto projetado que faz com que qualquer conjunto de linhas que sejam paralelas ao plano de projeção convirjam para um mesmo ponto denominado ponto-de-fuga. O surgimento do ponto-de-fuga pode ser melhor compreendido observando-se as figuras 4.10 e 4.11.

Na figura 4.10, um raio projetor parte do centro de projeção e incide sobre um ponto **p** da linha **l_A**. Quando **p** tende a infinito, o raio projetor vai encontrar este ponto no infinito, o que significa que a linha **RP** vai ficar paralela a linha **l_A**, cruzando, assim, o plano **x** sempre no mesmo ponto **pf**. Nota-se que um ponto sobre a linha **l_B**, paralela a linha **l_A**, vai ser alcançado no infinito de forma similar ao ponto sobre a linha **l_A**, ou seja, o raio projetor corta o eixo **x** no mesmo ponto **pf**.

Quando o raio projetor incide sobre uma linha paralela ao eixo de projeção, não tem ponto-de-fuga. A explicação para este caso é simples. Considere o ponto **A** da figura 4.11 que está sobre a linha **l_A**, paralela ao plano **xy. A**ssim, quando **A** tender a infinito, o raio projetor **RP1** vai encontrá-lo no infinito e, neste caso, estará paralelo ao plano **xy**, o que evita o surgimento do ponto-de-fuga. Ao contrário, o raio projetor **RP2** que incide sobre o ponto **B** sobre a linha **l_B**, perpendicular ao plano **xy**, vai tender à origem quando **B** tender a infinito. Neste caso, a projeção perspectiva do retângulo da figura 4.11 vai apresentar apenas um ponto-de-fuga. Para que a projeção deste retângulo apresente mais de um ponto-de-fuga basta rotacioná-lo (fig. 4.12), o que faz com que as suas linhas paralelas fiquem inclinadas em relação ao plano **xy**.

Para melhor se compreender a relação entre os pontos-de-fuga e o centro de projeção, a figura 4.13 apresenta a forma de se obter a projeção perspectiva de um paralelepípedo com lados paralelos ao eixo **x** e centro de projeção sobre o eixo **z**. O objeto é projetado do sistema de coordenadas 3D **xyz** no sistema de coordenadas 2D **x'y'**. Dado que a dimensão dos lados AB, CD, EF e GH não são visíveis, a dimensão dos lados A'B', C'D', E'F' e G'H' foram definidas arbitrariamente.

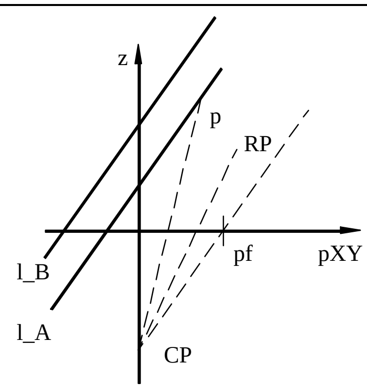


Figura 4.10

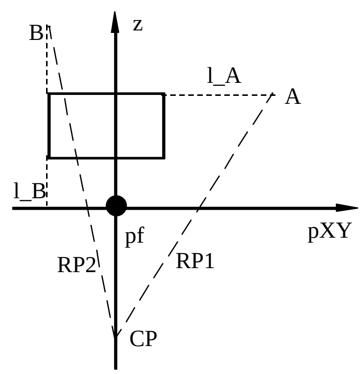


Figura 4.11

Figura 4.10: Centro de projeção e ponto-de-fuga em uma projeção perspectiva.

Figura 4.11: Projeção perspectiva de linhas paralelas e perpendiculares ao plano XY de projeção.

Em função do número de pontos-de-fuga associados às linhas paralelas aos três eixos cartesianos, as projeções perspetivas se subdividem em projeções de um ponto-de-fuga (fig. 4.14), de dois pontos-de-fuga (fig. 4.15) e de três pontos-de-fuga (fig. 4.16). Nota-se que cada conjunto de linhas paralelas no espaço pode ter associado um ponto-de-fuga. Assim, com o objetivo de definir um critério de classificação, somente as linhas paralelas aos eixos são consideradas.

Projeções perspectivas de três pontos-de-fuga são usadas menos frequentemente, dado que elas acrescentam pouco realismo ao já alcançado pelas projeções de dois pontos-de-fuga.

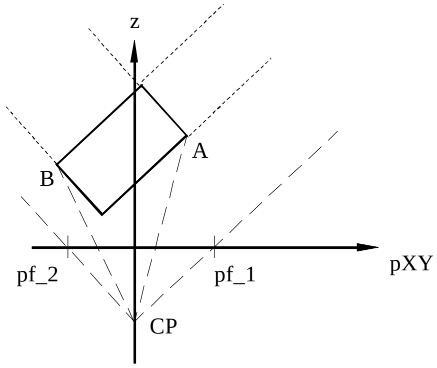


Figura 4.12: Projeção perspectiva de linhas inclinadas em relação ao plano de projeção.

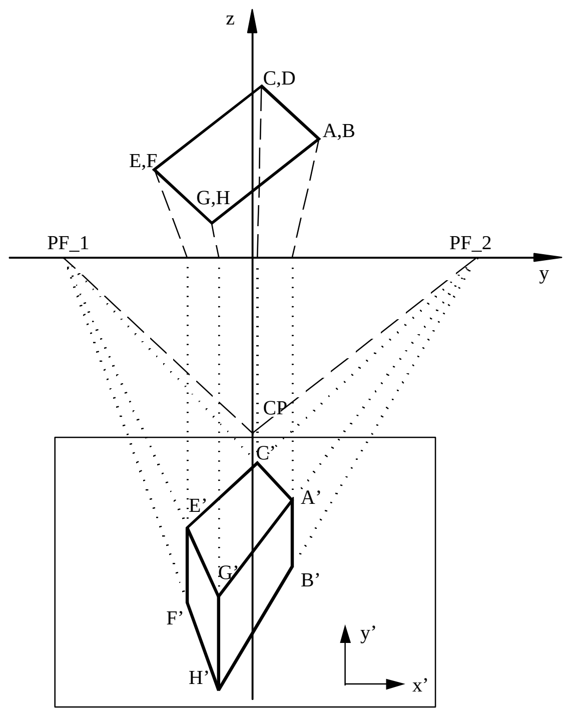


Fig. 4.13: Projeção perspectiva de um cubo com lados paralelos ao eixo **x**.

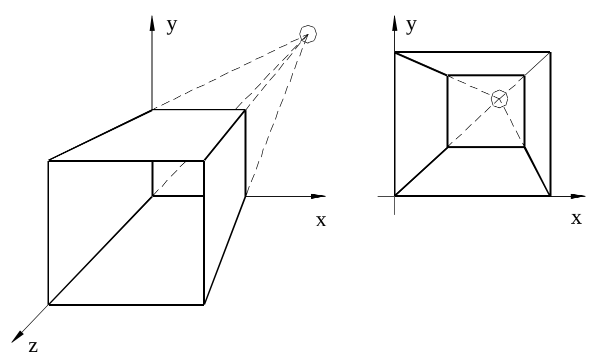


Figura 4.14: Projeção perspectiva com um ponto-de-fuga.

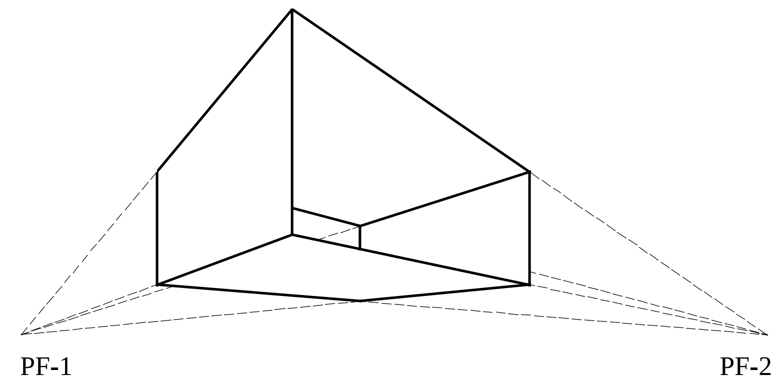


Figura 4.15: Projeção perspectiva com dois pontos-de-fuga.

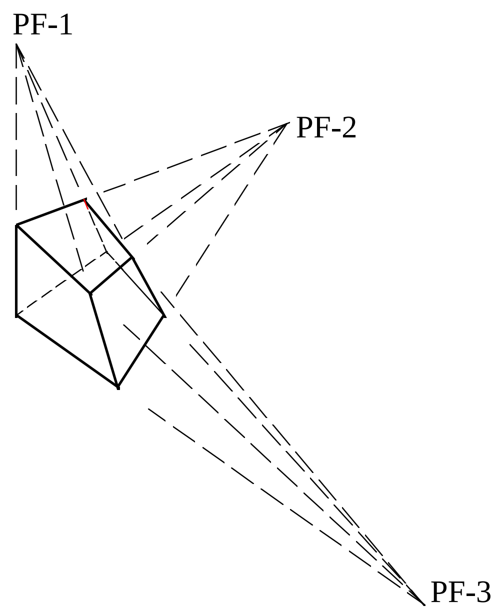


Figura 4.16: Projeção perspectiva com três pontos-de-fuga.

5. Álgebra das Projeções Planares Paralelas

Para que as projeções possam ser geradas em computador é necessário se definir matrizes de transformações que, aplicadas ao conjunto de pontos de um objeto tridimensional, permita a obtenção da figura projetada do objeto. Assim, a seguir, para cada tipo de projeção serão determinadas as matrizes de transformação. Este capítulo cobre a álgebra das projeções planares paralelas.

5.1 Álgebra das Projeções Planares Paralelas Ortográficas

As projeções planares ortográficas vista lateral, frontal e planta são obtidas através de transformações ortogonais, de acordo com as regras abaixo, e projeção no plano **xy** através de raios projetores perpendiculares a este plano (fig. 4.5)

vista lateral- rotação de -90° no eixo **x**, eliminação da coordenada **z**

vista frontal- rotação de -90° no eixo **z**, rotação de –90° no eixo **x**, e eliminação da coordenada **z**

vista superior (planta)- rotação de -90° no eixo **z**, eliminação da coordenada **z,** e translação no eixo **y** no maior valor na coordenada **x**.

A seguir é detalhado apenas a geração da vista lateral. Assim, partindo-se da equação de rotação em torno do eixo **x**, determina-se a matriz de projeção da vista lateral MP_{vl} (eq. 5.2).

$$R_X = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(q) & -\mathrm{sen}(q) & 0 \\ 0 & \mathrm{sen}(q) & \cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.1)}$$

Sendo θ = -90° e considerando-se a projeção no eixo **xy**, a matriz fica:

$$MP_{VL} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.2)}$$

Note-se que o versor sobre o eixo **z**, (0,0,1), será projetado em (0,1), como exemplificado nas figuras 5.1a e 5.1b. Observe a ordem correta dos eixos.

5.2 Álgebra das Projeções Planares Paralelas Ortográficas Axonométricas

As projeções vista lateral, frontal e planta permitem uma observação parcial do objeto projetado, em especial, caso se observe apenas uma delas, não é possível se conceber corretamente a forma do objeto. De forma a solucionar este problema, é usual a folha de projeto de um objeto conter as projeções vista lateral, frontal e planta, mais uma projeção axonométrica, a qual permite que se tenha uma visão mais integrada do objeto.

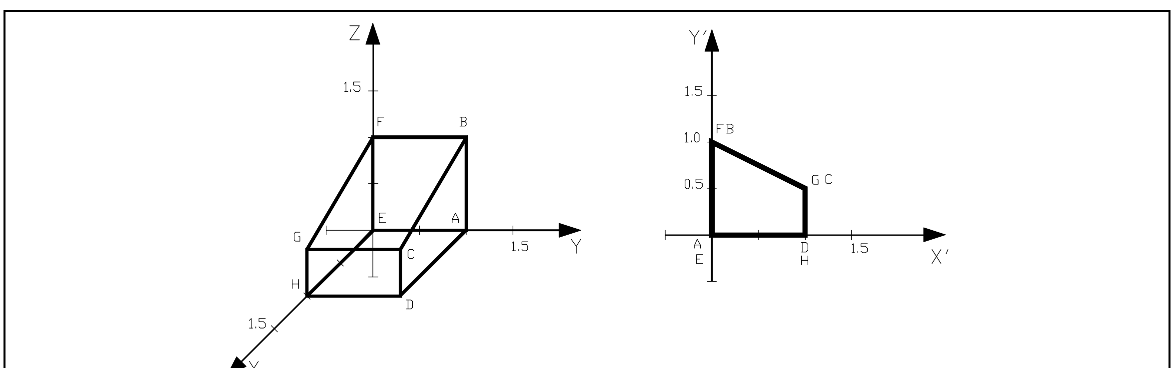


Figura 5.1a e 51.b: Projeção vista lateral.

Como citado no capítulo 4, as projeções paralelas ortográficas axonométricas tem a direção de projeção e a normal ao plano de projeção não coincidentes com a direção de um dos eixos principais (fig. 4.6). Isto é equivalente a se rotacionar adequadamente o objeto e considerar a direção de projeção e a normal ao plano de projeção coincidentes com a direção de um dos eixos principais.

Nas projeções axonométricas há uma alteração da dimensão das faces do objeto quando projetadas sobre o plano. O tipo de rotação do objeto e as considerações sobre a alteração das dimensões permitem que se determine as matrizes de projeção. Neste sentido, o cálculo destas matrizes vai partir de uma rotação do objeto em torno do eixo **y** e posteriormente do eixo **x** (eq. 5.3).

$$R_X \;\; R_Y \;=\; \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(q) & -\operatorname{sen}(q) & 0 \\ 0 & \operatorname{sen}(q) & \cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(f) & 0 & \operatorname{sen}(f) & 0 \\ 0 & 1 & 0 & 0 \\ -\operatorname{sen}(f) & 0 & \cos(f) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.3)}$$

$$R_X \;\; R_Y \;=\; \begin{bmatrix} \cos(f) & 0 & \operatorname{sen}(f) & 0 \\ \operatorname{sen}(f)\operatorname{sen}(q) & \cos(q) & -\cos(f)\operatorname{sen}(q) & 0 \\ -\operatorname{sen}(f)\cos(q) & \operatorname{sen}(q) & \cos(f)\cos(q) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.4)}$$

Aplicando-se esta matriz sobre os vetores unitários (1,0,0), (0,1,0) e (0,0,1), versores nos eixos **x**, **y**, e **z** respectivamente, tem-se:

$$\vec{u}'_X \;=\; \begin{bmatrix} x'_X \\ y'_X \\ z'_X \\ 1 \end{bmatrix} \;=\; R_X \;\; R_Y \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \;=\; \begin{bmatrix} \cos(f) \\ \operatorname{sen}(f)\operatorname{sen}(q) \\ -\operatorname{sen}(f)\cos(q) \\ 1 \end{bmatrix} \qquad \text{(eq. 5.5)}$$

$$\vec{u}'_Y \;=\; \begin{bmatrix} x'_Y \\ y'_Y \\ z'_Y \\ 1 \end{bmatrix} \;=\; R_X \;\; R_Y \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \;=\; \begin{bmatrix} 0 \\ \cos(q) \\ \operatorname{sen}(q) \\ 1 \end{bmatrix} \qquad \text{(eq. 5.6)}$$

$$\vec{u}'_Z \;=\; \begin{bmatrix} x'_Z \\ y'_Z \\ z'_Z \\ 1 \end{bmatrix} \;=\; R_X \;\; R_Y \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \;=\; \begin{bmatrix} \operatorname{sen}(f) \\ -\cos(f)\operatorname{sen}(q) \\ \cos(f)\cos(q) \\ 1 \end{bmatrix} \qquad \text{(eq. 5.7)}$$

Considerando-se apenas as componentes **x** e **y** dos versores rotacionados, tem-se as projeções destes vetores sobre os plano **xy**. De forma a se estipular condições para o encolhimento da dimensão destes vetores, são calculados os seus módulos, como ilustrado a seguir. Assim, para o caso da projeção do versor sobre o eixo **x** tem-se:

$$\left|\vec{u}_x\right|_{XY} \;=\; \sqrt{\left(x'_X\right)^2 \;+\; \left(y'_X\right)^2} \qquad \text{(eq. 5.8)}$$

$$\left|\vec{u}_x\right|_{XY} \;=\; \sqrt{\cos^2(f) \;+\; (\operatorname{sen}^2(f)\operatorname{sen}^2(q))} \qquad \text{(eq. 5.9)}$$

Similarmente, para os versores sobre os eixos **y** e **z** tem-se:

$$\left|\vec{u}_y\right|_{XY} \;=\; \left|\cos(q)\right| \qquad \text{(eq.5.10)}$$

$$\left|\vec{u}_z\right|_{XY} \;=\; \sqrt{(\operatorname{sen}^2(f) \;+\; \cos^2(f)\operatorname{sen}^2(q)} \qquad \text{(eq. 5.11)}$$

Considerando-se que no momento da projeção de um objeto 3D sobre o plano **xy**, ocorra um encolhimento por igual das coordenadas **x** e **y** dos pontos deste objetos, tem-se:

$$|\vec{u}_x|_{XY} = |\vec{u}_y|_{XY} \qquad \text{(eq. 5.12)}$$

$$\cos^2(f) + \text{sen}^2(f)\text{sen}^2(q) = \cos^2(q) \qquad \text{(eq. 5.13)}$$

Da trigonometria tem-se:

$$\cos^2(a) + \text{sen}^2(a) = 1 \qquad \text{(eq. 5.14)}$$

Aplicando-se esta relação na equação 5.13, tem-se:

$$\text{sen}^2(f)\text{sen}^2(q) = \text{sen}^2(f) - \text{sen}^2(q) \qquad \text{(eq. 5.15)}$$

$$\text{sen}^2(f) = \text{sen}^2(q) \ / \ (1 - \text{sen}^2(q)) \qquad \text{(eq. 5.16)}$$

Uma maneira simples de se calcular os ângulos q e f é considerar um valor fixo para o encolhimento no eixo **z**. Assim, inicialmente será suposto que a dimensão do versor projetado será 1/2, ou seja, encolherá pela metade. Logo, da equação 5.11 tem-se:

$$\text{sen}^2(f) + \cos^2(f)\text{sen}^2(q) = (1/2)^2 \qquad \text{(eq. 5.17)}$$

Aplicando-se a equação 5.16 e rearranjando, tem-se:

$$8\text{sen}^4(q) - 9\text{sen}^2(q) + 1 = 0 \qquad \text{(eq. 5.18)}$$

Fazendo-se $sen^2(q) = x$, tem-se:

$$8x^2 - 9x + 1 = 0 \qquad \text{(eq. 5.19)}$$

Resolvendo-se a equação 5.19, tem-se $x=1/8$ e $x=1$. A segunda raiz é inválida porque origina valor zero no denominador da equação 5.16. Usando-se a primeira raiz tem-se:

$$\text{sen}^2(q) = 1/8 \Rightarrow \qquad \text{sen}^2(f) = 1/7 \qquad \text{(eq. 5.20)}$$

$$\therefore \qquad q = 20.705^\circ \quad \text{e} \quad f = 22.208^\circ \qquad \text{(eq. 5.21)}$$

Desta maneira aplicando-se os valores obtidos na equação 5.21, tem-se a seguinte matriz:

$$R_X \ R_Y = \begin{bmatrix} 0.925820 & 0. & 0.377964 & 0 \\ 0.133631 & 0.935414 & -0.327321 & 0 \\ -0.353553 & 0.353553 & 0.866025 & 0 \\ 0. & 0. & 0. & 1 \end{bmatrix} \qquad \text{(eq. 5.22)}$$

$$MP_D = \begin{bmatrix} 0.925820 & 0. & 0.377964 & 0 \\ 0.133631 & 0.935414 & -0.327321 & 0 \\ 0. & 0. & 0. & 0 \\ 0. & 0. & 0. & 1 \end{bmatrix} \qquad \text{(eq. 5.23)}$$

A matriz MP_D anterior permite que se realize, após o zeramento da coordenada **z**, a projeção dimétrica (eq. 5.23), onde há o encolhimento por igual de duas coordenadas.

No entanto, o maior interesse é na projeção isométrica que permite o encolhimento por igual de todas as coordenadas. Logo:

$$|\vec{u}_x|_{XY} = |\vec{u}_y|_{XY} = |\vec{u}_z|_{XY} \qquad \text{(eq. 5.24)}$$

Assim, além da equação 5.13, para a determinação dos ângulos tem-se a seguinte equação:

$$\text{sen}^2(f) + \cos^2(f)\text{sen}^2(q) = \cos^2(q) \qquad \text{(eq. 5.25)}$$

Similarmente a equação 5.16, derivada da 5.12, podemos derivar da 5.25 a equação:

$$\text{sen}^2(f) = (1 - 2\text{sen}^2(q)) / (1 - \text{sen}^2(q)) \qquad \text{(eq. 5.26)}$$

$$\therefore \ \text{sen}^2(q) = 1 - 2\text{sen}^2(q) \qquad \Rightarrow \qquad \text{sen}^2(q) = 1/3$$

$$\text{sen}(q) = \sqrt{1/3} \;\Rightarrow\; \text{sen}(f) = \sqrt{1/2} \qquad \text{(eq. 5.27)}$$

$$q = 35.26429^{\circ} \quad \text{e} \quad f = 45.0^{\circ} \qquad \text{(eq. 5.28)}$$

Logo, substituindo os valores na matriz da equação 5.4, tem-se a matriz que permite a projeção isométrica (5.29), após se zerar a coordenada “z”:

$$R_X \;\; R_Y = \begin{bmatrix} 0.707107 & 0. & 0.707107 & 0 \\ 0.408248 & 0.816597 & -0.408248 & 0 \\ -0.577353 & 0.577345 & 0.577353 & 0 \\ 0. & 0. & 0. & 1 \end{bmatrix} \qquad \text{(eq. 5.29)}$$

$$MP_I = \begin{bmatrix} 0.707107 & 0. & 0.707107 & 0 \\ 0.408248 & 0.816597 & -0.408248 & 0 \\ 0. & 0. & 0. & 0 \\ 0. & 0. & 0. & 1 \end{bmatrix} \qquad \text{(eq. 5.30)}$$

Neste caso, um fato interessante advém da análise do ângulo α que o versor no eixo **x** faz com o eixo **x’**, quando projetado no plano **x’y’** (fig. 5.2a).

Assim, da equação 5.5, tem-se:

$$\vec{u}_X = \begin{bmatrix} \cos(f) \\ \text{sen}(f)\,\text{sen}(q) \end{bmatrix} \qquad \text{(eq. 5.31)}$$

$$\therefore \;\; \tan(a) = (\text{sen}(f)\,\text{sen}(q)) \,/\, \cos(f)$$

$$\tan(a) = (\sqrt{1/2}\sqrt{1/3}) \,/\, \sqrt{1/2} = \sqrt{3}\,/\,3 \qquad \Rightarrow \;\; a = 30.0^{\circ}$$

Este resultado é bem conhecido de desenhistas, pois é com um esquadro de 30° e 60° que eles desenham a projeção de um objeto em uma folha de projeto.

A seguir é esboçado um exemplo de obtenção de uma projeção isométrica de um objeto. Este objeto é o da figura 4.7, e a matriz $\vec{P}_I$ contém a descrição dos vetores de seus pontos (fig. 5.3a). O resultado está na figura 5.3b.

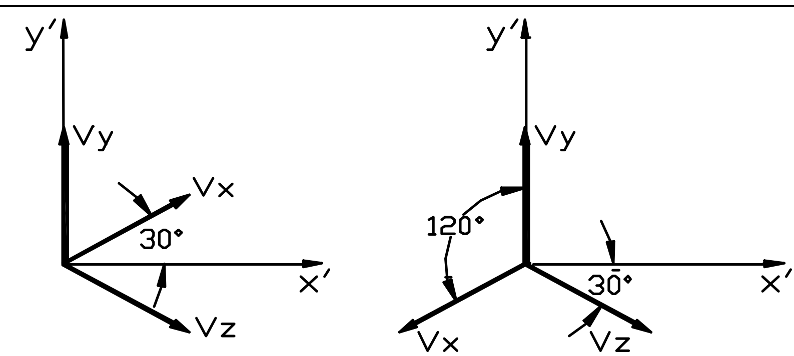


Figura 5.2a e 5.2b: Diferentes estruturas para a projeção isométrica.

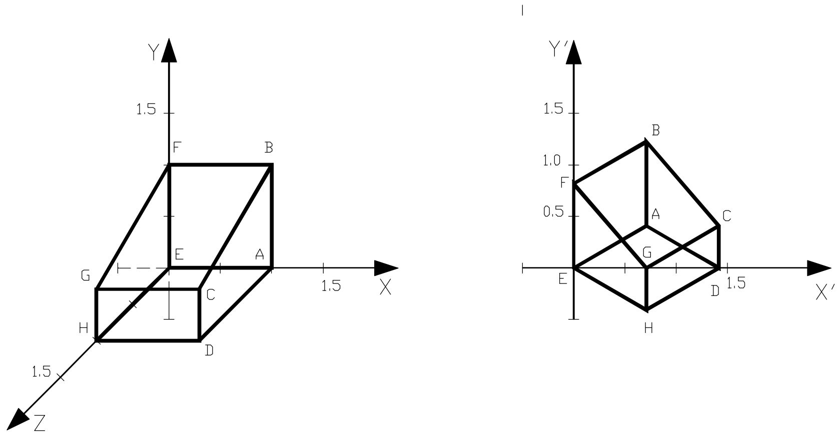


Figura 5.3a: Objeto a ser projetado. Figura 5.3b: Objeto projetado isometricamente.

$$\vec{P}_I \;=\; M_I \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 0 & 1 & 1/2 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ A & B & C & D & E & F & G & H \end{bmatrix} \quad \text{(eq. 5.32)}$$

Uma consideração importante sobre projeções isométricas se refere ao fato que:

$$\mathrm{sen}^2(q) = 1/3 \quad \Rightarrow \quad q = 35.26^\circ \quad \text{ou} \quad q = 144.74^\circ$$

$$sen^2(f) = 1/2 \quad \Rightarrow \quad f = 45.00^\circ \quad \text{ou} \quad f = 135.00^\circ$$

Logo, a projeção isométrica pode ser estruturada de outras formas. Na figura 5.2 é esboçada a estrutura trabalhada anteriormente (5.2a) e outra bastante utilizada (5.2b).

5.3 Álgebra das Projeções Planares Oblíquas

Segundo o que foi exposto no capítulo 4, as projeções paralelas oblíquas tem a direção do plano de projeção distinta da direção dos raios projetores (fig. 4.8). As projeções paralelas obliquas se subdividem em *cavalier* e *cabinet* (fig. 4.9), sendo que na projeção *cabinet* há uma distorção na dimensão do versor perpendicular ao plano de projeção, em geral, um encurtamento de 1/3 ou de 1/2. Além disto, em ambos os casos, o ângulo (α) que este versor forma com o eixo **x** pode ser de 45° ou 30°.

A dedução da matriz de projeção oblíqua é relativamente simples. Considere-se, inicialmente, o ângulo β que a linha entre o ponto P1 e sua projeção P1' forma com o plano **xy** e o ângulo α que é formado pela projeção da linha P1-P1' com o eixo **x** (fig. 5.4). O ângulo β determina o grau de encurtamento ou de dilatação da dimensão do versor, em especial, se β=45.0° não há alteração de dimensão e se β=60° há um encurtamento pela metade. O ângulo α não tem influência sobre o tamanho do versor e, basicamente, o seu valor é uma questão de preferência.

Quando o ponto está sobre o eixo **z** é trivial a dedução da fórmula de projeção, ou seja, sendo:

$$P1 = (0,0,z_1) \;\Rightarrow\; P1' = (d\cos(a), d\,\mathrm{sen}(a))$$

$$P2 = (0,0,z_2) \;\Rightarrow\; P2' = ((z_2/z_1)*d)\cos(a), ((z_2/z_1)*d)\,\mathrm{sen}(a))$$

A generalização da formulação para qualquer ponto pode também ser feita facilmente observando-se a figura 5.5. Assim, supondo-se $\mathbf{z_1}$ igual a **1** e $\mathbf{z_2}$ igual a **z**, tem-se:

$$P2 = (0,b,z) \;\Rightarrow\; P2' = (z*d\cos(a), b + (z*d)\,\mathrm{sen}(a))$$

$$P3 = (a,0,z) \;\Rightarrow\; P3' = (a + (z*d)\cos(a), (z*d)\,\mathrm{sen}(a))$$

Logo:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d\cos(a) & 0 \\ 0 & 1 & d\,\mathrm{sen}(a) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x+((z*d)\cos(a)) \\ y+((z*d)\,\mathrm{sen}(a)) \\ 0 \\ 1 \end{bmatrix} \qquad \text{(eq. 5.33)}$$

Para exemplificar a aplicação da matriz de projeção oblíqua, na matriz da equação 5.33 será adotado d=1, sem alteração da dimensão do versor, o que significa (β=45.0°), e α=45.0°, o que define uma matriz de projeção obliqua *cavalier* (eq. 5.34).

$$M_O = \begin{bmatrix} 1 & 0 & \sqrt{2}/2 & 0 \\ 0 & 1 & \sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.34)}$$

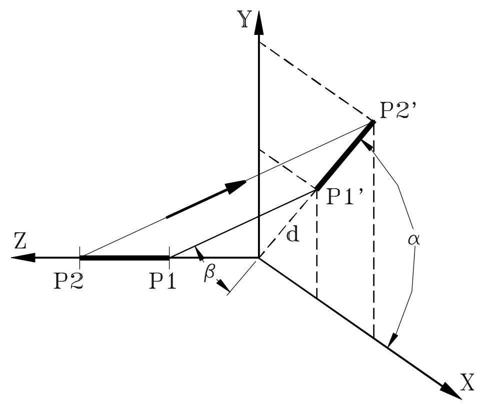


Figura 5.4: Esquema de projeção oblíqua.

Aplicando-se M_O sobre o objeto definido na fig. 5.3a, tem-se:

$$\vec{P}_O = M_O \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1/2 & 0 & 0 & 1 & 1/2 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ A & B & C & D & E & F & G & H \end{bmatrix} \qquad \text{(eq. 5.35)}$$

$$\vec{P}_O = \begin{bmatrix} 1 & 1 & (2+\sqrt{2})/2 & (2+\sqrt{2})/2 & 0 & 0 & \sqrt{2}/2 & \sqrt{2}/2 \\ 0 & 1 & (1+\sqrt{2})/2 & \sqrt{2}/2 & 0 & 1 & (1+\sqrt{2})/2 & \sqrt{2}/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ A & B & C & D & E & F & G & H \end{bmatrix} \text{(eq. 5.36)}$$

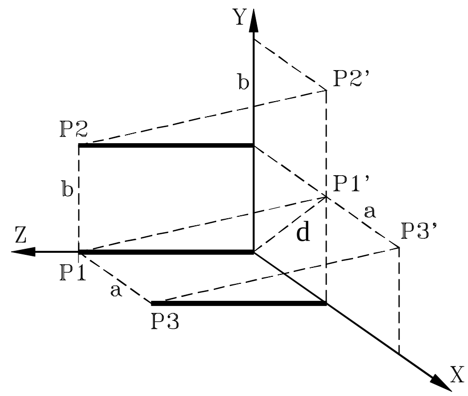


Figura 5.5: Projeção oblíqua de pontos fora do eixo base de projeção.

Note-se que o objeto projetado (fig. 5.6) fica com a face anterior elevada em relação ao eixo x devido ao ângulo α=45° de projeção. Uma outra solução é considerar o ângulo α=(180+45)° (fig. 5.7). Assim, a nova matriz de projeção fica:

$$M_O = \begin{bmatrix} 1 & 0 & -\sqrt{2}/2 & 0 \\ 0 & 1 & -\sqrt{2}/2 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{(eq. 5.33)}$$

Aplicando-se esta nova matriz M_O sobre os pontos do objeto definido na equação 5.30, tem-se os pontos projetados descritos na equação 5.34 e esboçados na figura 5.8.

$$\vec{P}_O = \begin{array}{c} \begin{bmatrix} 1 & 1 & (2-\sqrt{2})/2 & (2-\sqrt{2})/2 & 0 & 0 & -\sqrt{2}/2 & -\sqrt{2}/2 \\ 0 & 1 & (1-\sqrt{2})/2 & -\sqrt{2}/2 & 0 & 1 & (1-\sqrt{2})/2 & -\sqrt{2}/2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \begin{matrix} A & B & C & D & E & F & G & H \end{matrix} \end{array}$$

(eq. 5.34)

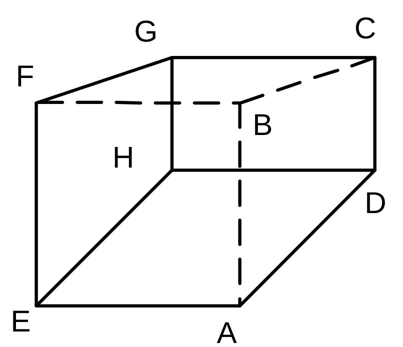


Figura 5.6: Objeto e sua projeção oblíqua *cavalier* com α=45°.

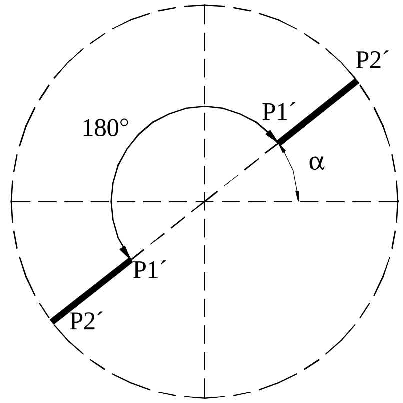


Figura 5.7: Rotação da projeção de 180.0°

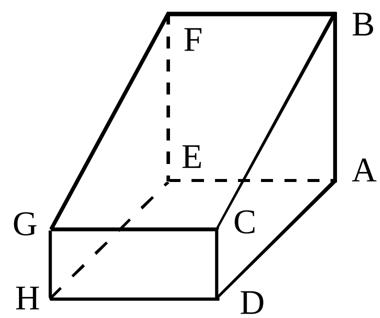


Figura 5.8: Objeto e sua projeção oblíqua *cavalier* com α=(180+45)°.

6. Álgebra das Projeções Planares Perspectivas

Para uma dedução simples e intuitiva da álgebra das projeções perspectivas é interessante se analisar a figura 6.1. Neste caso, o ponto P (x,y,z) é projetado no ponto P' (x',y',0), contido no plano **xy**, a partir de um centro de projeção localizado no ponto C (0,0,k). Aplicando-se semelhança de triângulos a figura produzida e considerando que a coordenada z tem valor negativo obtém-se:

$$x' / k = x / (-z + k) \;\Rightarrow\; x' = x / ((-z / k) + 1) \qquad \text{(eq. 6.1)}$$

$$y' / k = y / (-z + k) \;\Rightarrow\; y' = y / ((-z / k) + 1) \qquad \text{(eq. 6.2)}$$

Note-se que estes valores são os mesmos produzidos pela matriz de transformação abaixo.

$$\begin{bmatrix} X \\ Y \\ Z \\ H \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/k & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ (-z/k)+1 \end{bmatrix} \qquad \text{(eq. 6.3)}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} X/H \\ Y/H \\ Z/H \\ 1 \end{bmatrix} = \begin{bmatrix} x/((-z/k)+1) \\ y/((-z/k)+1) \\ z/((-z/k)+1) \\ 1 \end{bmatrix} \qquad \text{(eq. 6.4)}$$

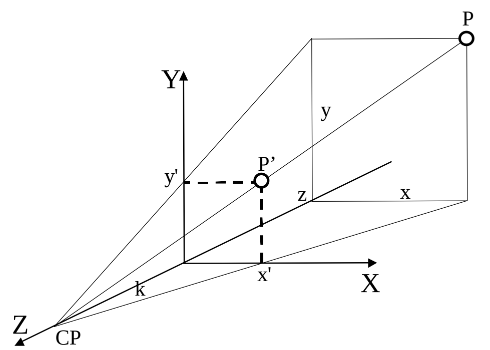


Figura 6.1: Projeção de um ponto no plano a partir de um centro de projeção.

Quando se adota o plano **xy** como o plano de projeção, desconsidera-se o valor de z' na equação 6.4. Note-se que nesta equação se $k \to \infty$, então $x' \to x$ e $y' \to y$, o que origina uma projeção paralela, com raios projetores perpendiculares ao plano **xy**, considerado como o plano de projeção.

Considerando-se x = at, y = bt e z = -ct, com a, b, e c maiores do que zero, tem-se:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} at/((ct/k)+1) \\ bt/((ct/k)+1) \\ -ct/((ct/k)+1) \\ 1 \end{bmatrix}$$

Se $t \to \infty$ tem-se $\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} ka/c \\ kb/c \\ -k \\ 1 \end{bmatrix}$

Logo, o ponto (ka/c, kb/c, -k) constitui um ponto-de-fuga, dado que ele é um ponto finito para o qual converge a projeção de uma linha infinita. Dependendo do valor de a, b e c, dado k fixo, infinitos pontos-de-fuga podem ser determinados.

Em termos de classificação da projeção perspectiva interessa apenas os pontos-de-fuga associados às linhas paralelas aos eixos cartesianos. Assim, considere-se:

$$\vec{v}_1 = \begin{bmatrix} at \\ b \\ c \\ 1 \end{bmatrix}, \quad \vec{v}_2 = \begin{bmatrix} a \\ bt \\ c \\ 1 \end{bmatrix}, \quad \vec{v}_3 = \begin{bmatrix} a \\ b \\ ct \\ 1 \end{bmatrix}$$

$\vec{v}_1$, $\vec{v}_2$ e $\vec{v}_3$ definem, respectivamente, uma linha paralela ao eixo **x**, **y** e **z**. A projeção desses vetores no plano xy, é dada por:

(eq. 6.5) (eq. 6.6) (eq. 6.7)

$$\vec{v}_1' = \begin{bmatrix} at/((-c/k)+1) \\ b/((-c/k)+1) \\ c/((-c/k)+1) \\ 1 \end{bmatrix}, \quad \vec{v}_2' = \begin{bmatrix} a/((-c/k)+1) \\ bt/((-c/k)+1) \\ c/((-c/k)+1) \\ 1 \end{bmatrix}, \quad \vec{v}_3' = \begin{bmatrix} a/((-ct/k)+1) \\ b/((-ct/k)+1) \\ ct/((-ct/k)+1) \\ 1 \end{bmatrix}$$

E quando $t \rightarrow \infty$ resulta em:

$$\vec{v}_1' = \begin{bmatrix} \infty \\ B_1 \\ C_1 \\ 1 \end{bmatrix} \quad \vec{v}_2' = \begin{bmatrix} A_2 \\ \infty \\ C_2 \\ 1 \end{bmatrix} \quad \vec{v}_3' = \begin{bmatrix} 0 \\ 0 \\ -k \\ 1 \end{bmatrix}$$

Mesmo sendo B_1, C_1, A_2 e C_2 constantes finitas, $\vec{v}_1'$ e $\vec{v}_2'$, que são respectivamente a transformação perspectiva de $\vec{v}_1$ e $\vec{v}_2$, resultam em pontos que no espaço tendem a infinito, ou seja, não há pontos-de-fuga associados as linhas paralelas aos eixos **x** e **y**. Este resultado é esperado, pois $\vec{v}_1$ e $\vec{v}_2$ definem linhas perpendiculares ao eixo **z** que contém o centro-de-projeção. A transformação perspectiva de $\vec{v}_3$ resulta em um ponto finito no espaço, ou seja, há ponto-de-fuga pois $\vec{v}_3$ é perpendicular ao eixo **z** que contém o centro de projeção.

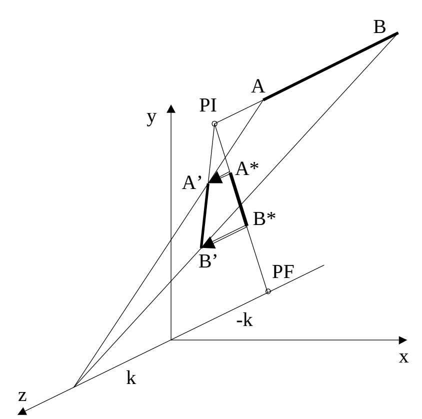


Figura 6.2: Projeção de uma linha paralela ao eixo **z**.

Derivando-se a figura 6.2 da 6.1, pode-se visualizar facilmente o surgimento do ponto-de-fuga. Os pontos A' e B' são as projeções no plano **xy** dos pontos AB da linha A-B perpendicular ao plano de projeção. Note-se que considerando a coordenada **z**, os pontos A* e B* mostram como a transformação perspectiva força a convergência dos pontos transformados para o ponto-de-fuga. Para t=0, $\vec{v}_3$ resulta no ponto PI (a, b, 0), que é o ponto que a linha AB estendida intercepta o plano. Os pontos A* e B* estão na linha PI-PF e, assim, quando B→∞, B*→PF e B'→(0,0), o que faz com que todos os pontos projetados no plano estejam na linha limitada pelos pontos (a,b) e (0,0).

Em especial, observando a figura 6.2 e as equações 6.5, 6.6 e 6.7 compreende-se o mecanismo de funcionamento da transformação perspectiva. O surgimento do ponto-de-fuga no eixo **z** se deve ao fato de que na equação 6.7 o denominador tende a infinito quando t→∞ e, assim, as constantes divididas por um valor infinito resultam em zero e o valor infinito do numerador dividido pelo valor infinito do denominador resulta em um valor finito.

Em termos de computação gráfica, o foco de interesse é a projeção de um ponto no espaço sobre o plano da tela. Neste sentido, para ilustrar uma forma mais intuitiva de se obter as equações anteriores, será deduzido a matriz de projeção perspectiva a partir da consideração de que a tela do computador está sobre o plano **xy** e que o centro de projeção (observador) está fora deste plano e em alguma parte positiva do eixo **z** (fig. 6.3).

Supondo-se conhecido P1 e CP (fig. 6.3), pode-se escrever a equação de reta que passa por estes dois pontos. No caso bidimensional, a equação da reta em função dos pontos pA (xa,ya) e pB (xb,yb) fica:

$$x = x_a + (x_b - x_a)u$$

$$y = y_a + (y_b - y_a)u$$

Assim, no caso da figura 6.5 tem-se:

$$x = x_c + (x_1 - x_c)u \qquad \text{(eq. 6.5)}$$

$$y = y_c + (y_1 - y_c)u \qquad \text{(eq. 6.6)}$$

$$z = z_c + (z_1 - z_c)u \qquad \text{(eq. 6.7)}$$

Logo, o valor de *u* para P2, com coordenadas (x_2,y_2,0), é:

$$u = -z_c / (z_1 - z_c) \qquad \text{(eq. 6.8)}$$

Compondo-se as equações 6.5, 6.6 e 6.8 tem-se:

$$x_2 = x_c - z_c(x_1 - x_c) / (z_1 - z_c) \qquad \text{(eq. 6.9)}$$

$$y_2 = y_c - z_c(y_1 - y_c) / (z_1 - z_c) \qquad \text{(eq. 6.10)}$$

Escrevendo-se este resultado em forma matricial tem-se:

$$MP = \begin{bmatrix} -z_c & 0 & x_c & 0 \\ 0 & -z_c & y_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -z_c \end{bmatrix} \qquad \text{(eq. 6.11)}$$

Dividindo-se todos os elementos da matriz *MP* da equação 6.11 por um mesmo valor não se altera o resultado da sua aplicação sobre um vetor em coordenadas homogêneas. Assim, dividindo-se todos os elementos por $1/z_c$ tem-se:

$$MP = \begin{bmatrix} 1 & 0 & -x_c / z_c & 0 \\ 0 & 1 & -y_c / z_c & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/z_c & 1 \end{bmatrix} \qquad \text{(eq. 6.12)}$$

Considerando-se x_c e y_c iguais a zero na *MP* da equação 6.12, tem-se:

$$MP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/z_c & 1 \end{bmatrix} \qquad \text{(eq. 6.13)}$$

O que está de acordo com a matriz de transformação perspectiva *MTPz*, deduzida anteriormente, no caso do centro de projeção estar no ponto $(0,0,z_c)$ (fig. 6.2).

A matriz *MP* da equação 6.12 permite um controle explícito do ponto-de-fuga em função do centro de projeção, no entanto, gera apenas um ponto-de-fuga. Para a geração de mais pontos-de-fuga pode-se associar *MP* com matrizes de rotação. Note-se que, como exposto no capítulo 4.3 e esboçado nas figuras 4.10, 4.11 e 4.12, caso se rotacione o objeto é possível obter um número maior de pontos-de-fuga. Assim, fazendo-se a multiplicação da matriz *MP* pela matriz *RxRy* (equação 5.4) tem-se a equação 6.14:

$$MPR = \begin{bmatrix} \cos(f) + (xc/zc)\,\text{sen}(f)\cos(q) & (-xc/zc)\,\text{sen}(q) & \text{sen}(f) - (xc/zc)\cos(f)\cos(q) & 0 \\ \text{sen}(f)\,\text{sen}(q) + (yc/zc)\,\text{sen}(f)\cos(q) & \cos(q) - (yc/zc)\,\text{sen}(q) & -\cos(f)\,\text{sen}(q) - (yc/zc)\cos(f)\cos(q) & 0 \\ 0 & 0 & 0 & 0 \\ (1/zc)\,\text{sen}(f)\cos(q) & (-1/zc)\,\text{sen}(q) & (-1/zc)\cos(f)\cos(q) & 1 \end{bmatrix}$$

Supondo-se z_c, *sen(f)*, *sen(q)*, *cos(f)*, *cos(q)* diferentes de zero, deriva-se os seguintes pontos-de-fuga:

$$PFx = \begin{bmatrix} x_c + z_c(\cos(f)/(\text{sen}(f)\cos(q))) \\ y_c + z_c(\text{sen}(q)/\cos(q)) \end{bmatrix} \qquad \text{(eq. 6.15)}$$

$$PFy = \begin{bmatrix} x_c \\ y_c - z_c(\cos(q)/\text{sen}(q)) \end{bmatrix} \qquad \text{(eq. 6.16)}$$

$$PFz = \begin{bmatrix} x_c - z_c(\text{sen}(f)/(\cos(f)\cos(q))) \\ y_c + z_c(\text{sen}(q)/\cos(q)) \end{bmatrix} \qquad \text{(eq. 6.17)}$$

Assim, utilizando-se a matriz *MPR* tem-se o controle simultâneo do centro de projeção e dos pontos-de-fuga. Esta matriz pode ser modificada para incorporar rotações no eixo z.

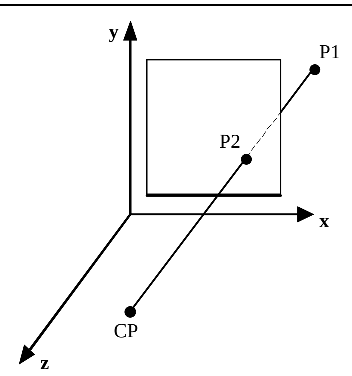


Figura 6.3: Centro de projeção genérico.

7. Uso de Projeções em Sistemas CAD e Modeladores

Atualmente é possível se encontrar no mercado vários sistemas CAD bem como modeladores. Como não é possível se analisar todos estes produtos, optou-se por estudar apenas os recursos de geração de objetos tridimensionais no AutoCAD e no 3D Studio, isto em função da grande popularidade destes produtos. Os recursos considerados utilizam os conceitos abordados anteriormente.

O AutoCAD permite o trabalho em um sistema de coordenadas tridimensional que pode ser reconfigurado através do comando UCS (*User Coordinates System*). O sistema de coordenadas inicial é denominado de *World Coordinate System* e identificado pela letra W no ícone de coordenadas (fig. 7.1). Definido um novo sistema de coordenadas 3D é possível se deslocar a origem do sistema de coordenadas corrente para a origem do novo (fig. 7.2). Além disto, é possível se rotacionar no espaço o sistema de eixos em relação ao do sistema anterior (fig. 7.3). A direção do eixo **z** é sempre calculada com base na "regra da mão direita". Manipulando o sistema de coordenadas é possível criar um objeto tridimensional utilizando-se, por exemplo, um recurso avançado de extrusão, o qual permite que, definido um caminho, uma forma seja replicada ao seu redor (fig. 7.4). Este tipo de recurso exige que a forma e o caminho não estejam contidos no mesmo plano, assim para definí-los se utiliza o recurso de rotação do sistema de coordenadas.

Outra forma para gerar objetos tridimensionais é utilizar operações booleanas de sólidos pré-definidos. Este recurso foi utilizado para criar o sólido da figura 7.5.

Em termos de diferentes forma de visualização de um objeto, o comando ***vpoint*** do AutoCAD permite a definição de um ponto-de-vista a partir do qual o objeto será visto segundo uma projeção planar paralela ortográfica. O ponto-de-vista define um vetor que parte da origem do sistema de coordenadas corrente e é perpendicular ao plano de projeção (fig. 7.5). O sentido do vetor é do plano para o observador (fig. 7.6). Neste caso, quando *vpoint* é igual a (0,-1,0) o desenho é exibido de forma que o eixo **y** esteja apontando no sentido contrário ao do observador e vice-versa quando *vpoint* é igual a (0,1,0).

Além disto, o AutoCAD fornece uma maneira alternativa para se definir o ponto-de-vista de um objeto. Considerando-se uma esfera no espaço é possível se definir um vetor que vai do centro da esfera até a sua borda através de dois ângulos de rotação. O primeiro é medido no plano XY e o segundo no plano XZ. Uma representação única para estes dois ângulos pode ser dada através de duas circunferências e um trípode (fig. 7.7), onde deslocamentos circulares em torno do centro do trípode indicam rotações ao redor da origem dos eixos no plano XY e deslocamentos do cursor do centro do trípode para a circunferência mais externa implica na rotação norte-sul do vetor, ou seja, caracterizam rotações no plano XZ. As figuras 7.8, 7.9 e 7.10 exemplificam diferentes vistas de rotações tridimensionais realizadas através do comando *vpoint* e do tripóide.

Observe-se que uma forma mais intuitiva de realizar rotações é o esquema de movimentos horizontais e verticais do *mouse*, como o utilizado pelo software de visualização tridimensional de dados meteorológicos VIS-5D. Neste processo, movimentos horizontais ou verticais do *mouse* indicam rotações horizontais ou verticais do objeto, respectivamente. Como as rotações são sempre realizadas sobre a última posição do objeto, este processo se torna bastante simples e intuitivo.

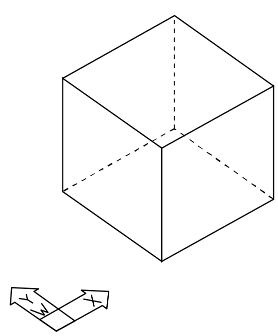


Figura 7.1

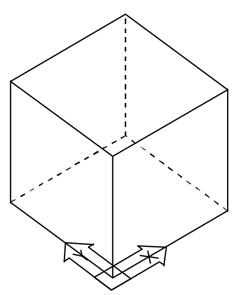


Figura 7.2

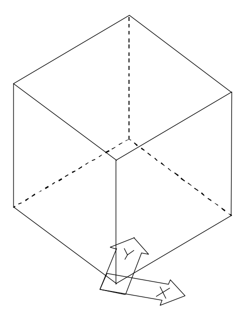


Figura 7.3

Figura 7.1: Sistema de coordenadas no AutoCAD (UCS).

Figura 7.2: Mudança de origem dos eixos.

Figura 7.3: Rotação dos eixos.

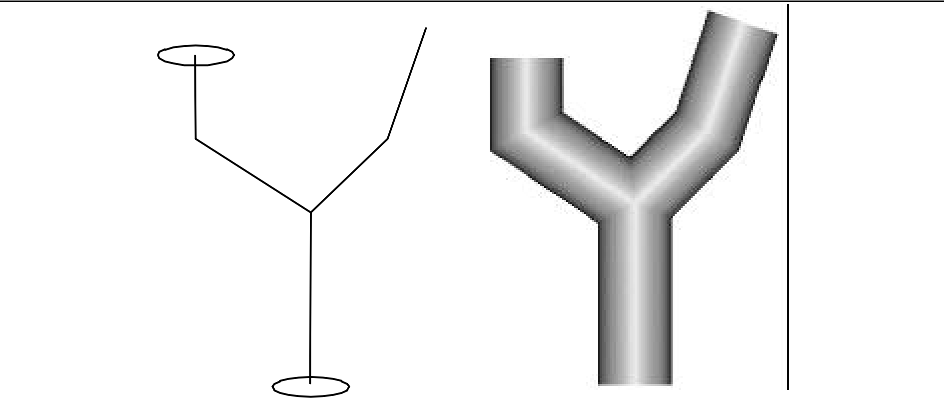


Figura 7.4: Processo de extrusão no AutoCAD.

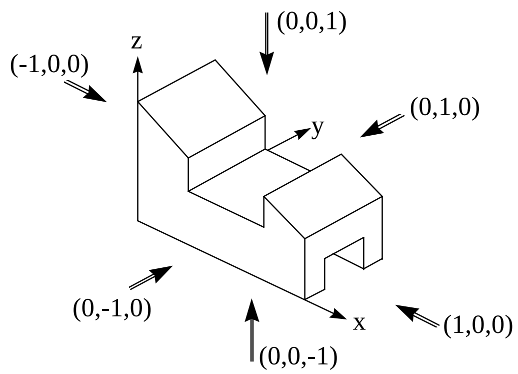


Figura 7.5: Diferentes especificações de ponto-de-vista (*vpoint*).

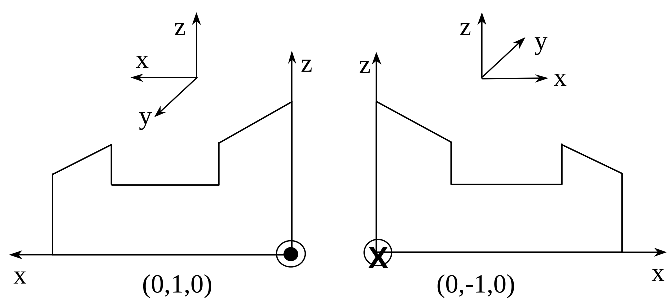


Figura 7.6: Especificações do ponto-de-vista em relação ao eixo **y**.

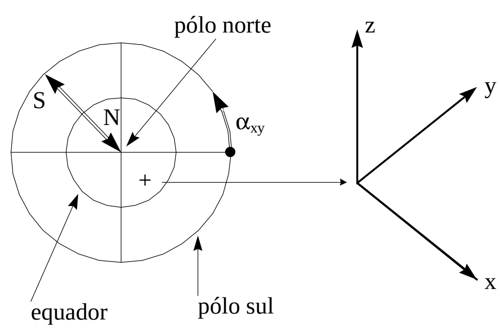


Figura 7.7: Esquema interativo de especificação de ponto-de-vista.

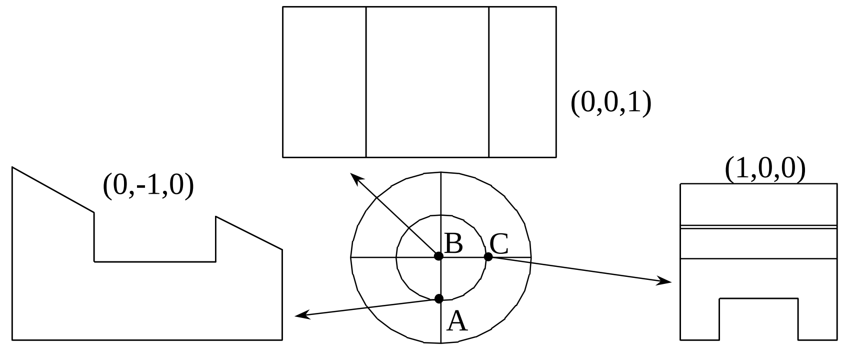


Figura 7.8: Especificações de ponto-de-vista para planta, vista lateral e frontal.

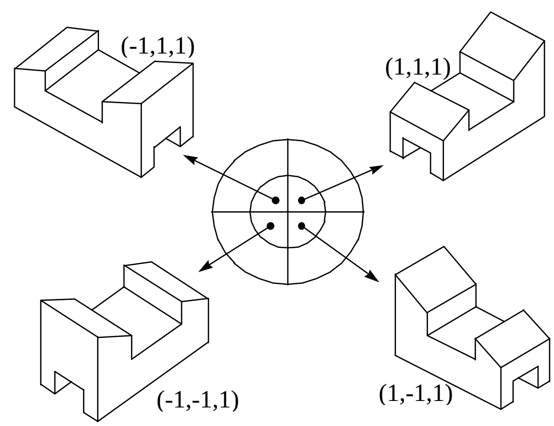


Figura 7.9: Diferentes especificações de ponto-de-vista.

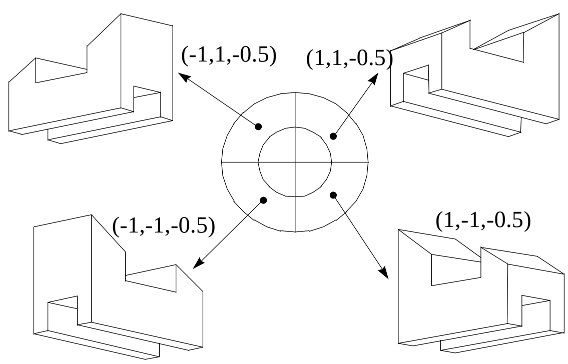
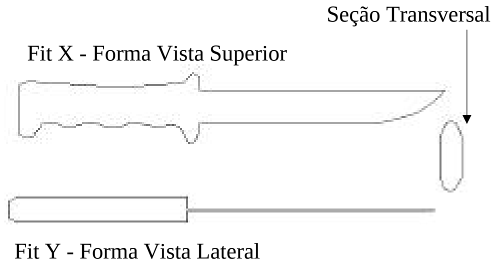


Figura 7.10: Diferentes especificações de ponto-de-vista.

Um uso interessante do conceito de projeções para a geração de objetos tridimensionais está disponível no 3D Studio Max da AutoDesk. Este sistema permite que o usuário defina e ajuste formas que constituirão as projeções paralelas ortográficas planta, vista lateral e vista frontal de um objeto 3D.

A construção do objeto parte da definição de uma forma base denominada de seção transversal (fig. 7.11). Considerando o objeto 3D a perna torneada de uma mesa, a forma base é a circunferência que define o tubo a ser torneado. Através da definição de um caminho para a replicagem da forma base (fig. 7.12) é gerado um objeto 3D denominado no 3D Studio de *LOFT* (objeto com elevação).

O “torneamento” do objeto *LOFT* é realizado através de um recurso denominado de *Fit Deformation*. Através desta deformação a forma superior (Fit X) e a lateral (Fit Y) do objeto desejado são associadas ao objeto *LOFT* base para se obter a forma 3D composta (fig. 7.13). A edição das formas é bem simples dado que cada uma é um conjunto de pontos interligados por linhas retas ou curvas e, assim, podem ser modificadas por alteração de posição dos pontos, pela inserção de mais pontos ou pelo aumento/diminuição das formas em relação aos seus eixos. Definida a forma final do objeto, ele pode ser renderizado (fig. 7.14).



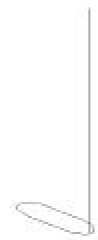


Figura 7.11 Figura 7.12

Figura 7.11: Vistas do objeto Faca.

Figura 7.12: Definição da forma base do objeto Faca.

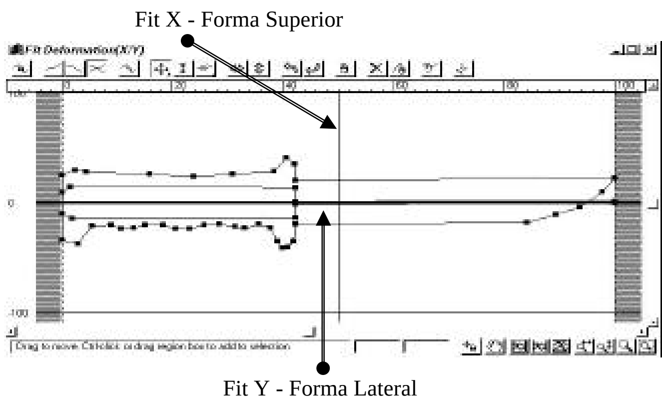


Figura 7.13: Deformações do objeto *LOFT* base.

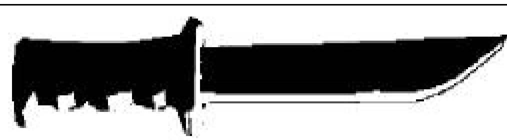


Figura 7.14: Objeto final renderizado.

8. Transformações e Mapeamento de Coordenadas em Bibliotecas Gráficas

Diferentes bibliotecas gráficas utilizam esquemas diversos de transformações e mapeamento de coordenadas. No caso de uma biblioteca gráfica que utiliza primitivas gráficas bidimensionais, como o GKS (*Graphical Kernel System*), por exemplo, o modo de mapeamento é realizado através da definição de duas transformações.

No GKS, especificado os limites da área de trabalho do usuário, cujo espaço é denominado de coordenadas de mundo, um retângulo em seu interior define a área de interesse para a visualização (fig. 7.1). Um segundo retângulo define o espaço das coordenadas normalizadas, na qual o retângulo é mapeado através da transformação de normalização. Nesta etapa pode se habilitar ou não operações de recorte (*clipping*) bem como de preservação ou não de relações de aspecto (*aspect ratio*), ou seja, o objeto original a ser exibido pode ser achatado ou esticado ou mantido com suas proporções iniciais inalteradas. A transformação de dispositivo mapeia as coordenadas de um retângulo definido no espaço das coordenadas normalizadas no das coordenadas de dispositivo. Nesta fase, não são permitidas operações de recorte e de alteração da relação de aspecto.

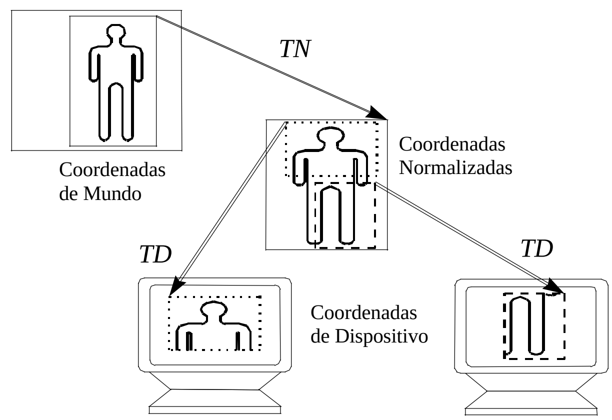


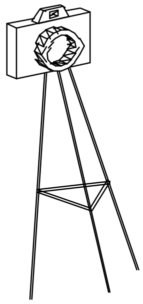
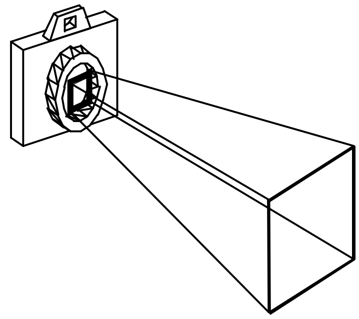
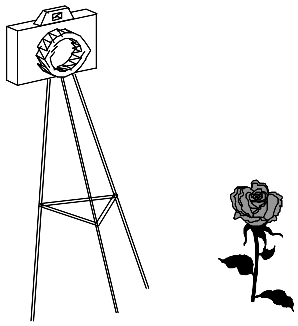
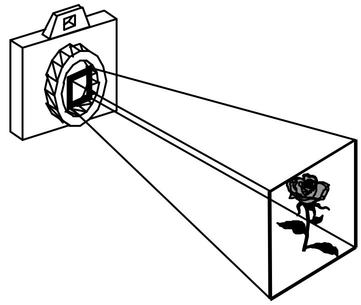
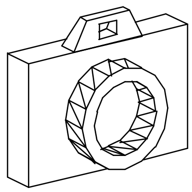
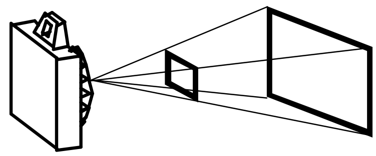
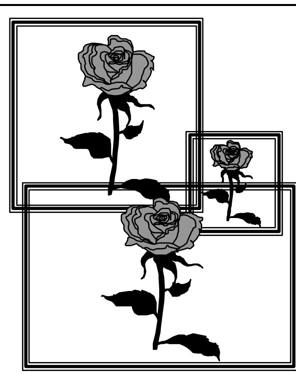
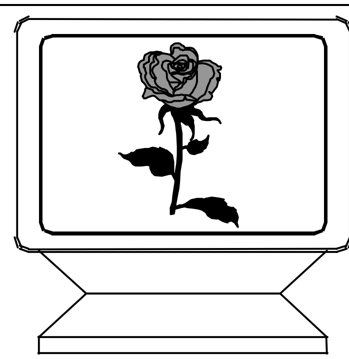
Figura 8.1: Mapeamento de coordenadas no GKS.

8.1 Esquema de Transformações na Biblioteca Gráfica OpenGL

O processo de transformação na OpenGL é baseado em uma analogia com as etapas usualmente seguidas para se fotografar um objeto. Inicialmente, há o posicionamento da câmera, ou seja, regulagem do trípode e da direção para a qual a câmera aponta, de forma a se definir uma região de interesse para a foto (fig. 8.2.a). Na OpenGL, isto significa definir o ponto-de-observação, o qual determina o volume de visualização no mundo em questão (fig. 8.2.b), e está relacionado com a transformação de visualização (*viewing transformation*).

A segunda etapa no processo de fotografar é o posicionamento do objeto (fig. 8.2.c), o que significa determinar qual parte dele deverá aparecer mais acentuadamente na fotografia. Para tanto, o que se faz é mover ou rotacionar o objeto. Na OpenGL, esta etapa é também relacionada a definição de posicionamento do modelo (fig. 8.2.d), sendo denominada de transformação de modelagem (*modeling transformation*), que se caracteriza por transformações de rotação, translação e escalamento.

Note-se que estas duas etapas no processo da fotografia se confundem, dado que é possível se obter a mesma foto rotacionando ou transladando o objeto ou a câmera. Neste sentido, a figura 8.3.a exemplifica como pode-se afastar um objeto do ponto-de-observação, cuja posição não se altera e a figura 8.3.b como afastar o ponto-de-observação do objeto, cuja posição permanece inalterada. Assim, na OpenGL as transformações de visualização e modelagem são tratadas de forma combinada como *modelview transformation*.

 Figura 8.2.a: Posicionamento de câmera.	 Figura 8.2.b: Posicionando o volume de visualização no mundo.
 Figura 8.2.c: Posicionando o objeto (modelo).	 Figura 8.2.d: Posicionando o modelo no mundo.
 Figura 8.2.e: Ajustando a lente para o enquadramento.	 Figura 8.2.f: Determinando a forma do volume de visualização.
 Figura 8.2.g: Fotografia	 Figura 8.2.h: *Viewport*

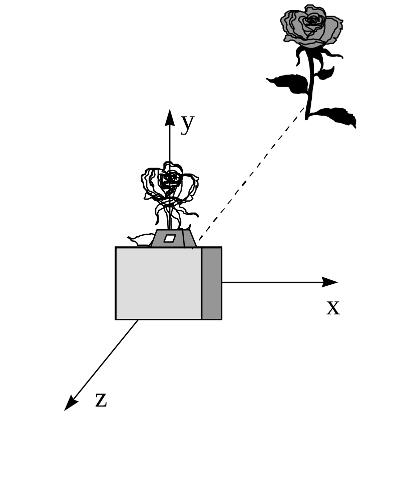


Figura 8.3.a: Translação do objeto.

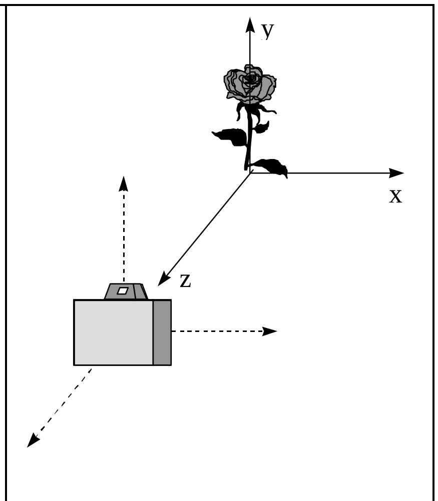


Figura 8.3.b: Translação do ponto-de-observação.

A terceira etapa do processo de fotografar é o ajuste das lentes para definir o enquadramento desejado. Neste sentido, ao se usar uma grande angular, pode-se aumentar o ângulo de enquadramento, possibilitando que um espaço maior ao redor do objeto alvo se enquadre na área da foto (fig. 8.2.e). Na OpenGL, esta etapa corresponde a definição do volume de visualização (fig. 8.2.f), ou seja, o que vai ser realmente exibido na tela, bem como do tipo de projeção utilizado para se projetar o conteúdo do volume de visualização na tela.

Após se bater a foto, a última etapa se refere a revelação e produção da fotografia. Neste etapa, não é possível mais se proceder a alterações na cena, no entanto, pode-se escolher toda região ou uma sub-parte dela para ser mapeada na área da foto (fig. 8.2.g). Na OpenGL, o processo é similar (fig. 8.2.h), no entanto, também é possível se proceder a alterações de proporções entre a altura e a largura do retângulo onde está enquadrada a cena, conforme comentado no tópico sobre transformação de *viewport*.

Concluindo, o processo de transformação na OpenGL está dividido em quatro etapas (fig. 8.4), cada um explicado em maiores detalhes a seguir.

8.1.1. Transformações de Visualização e Modelagem

A OpenGL oferece uma função utilitária para se definir uma transformação de visualização (fig. 8.5), baseando-se na posição do observador (ox,oy,oz), na posição do centro da cena (cx,cy,cz) e em um vetor que indica o eixo vertical do observador (vx,vy,vz), cujo valor é em geral (0, 1, 0). Essa função tem a seguinte sintaxe:

```
gluLookAt (GLdouble ox, GLdouble oy, GLdouble oz, GLdouble cx, GLdouble cy, GLdouble cz,
GLdouble vx, GLdouble vy, GLdouble vz);
```

Note-se que gluLookAt é uma função utilitária porque ela é implementada pela combinação das transformações básicas de visualização e modelagem da OpenGL, ou seja, translação, rotação e escalamento.

A função de translação que aplica sobre a matriz *modelview* tem a seguinte sintaxe:

```
glTranslatef (GLfloat x, GLfloat y, GLfloat z);
glTranslated (GLdouble x, GLdouble y, GLdouble z);
```

Onde x, y e z indicam as distâncias transladadas nos eixos x, y ou z.

Também aplicada sobra a matriz *modelview*, a função para se realizar uma rotação tem a seguinte sintaxe:

```
glRotatef (GLfloat angulo, GLfloat x, GLfloat y, GLfloat z);
glRotated (GLdouble angulo, GLdouble x, GLdouble y, GLdouble z);
```

Onde o parâmetro angulo indica o ângulo de rotação, e x, y e z indicam em quais dos eixos será aplicada a rotação.

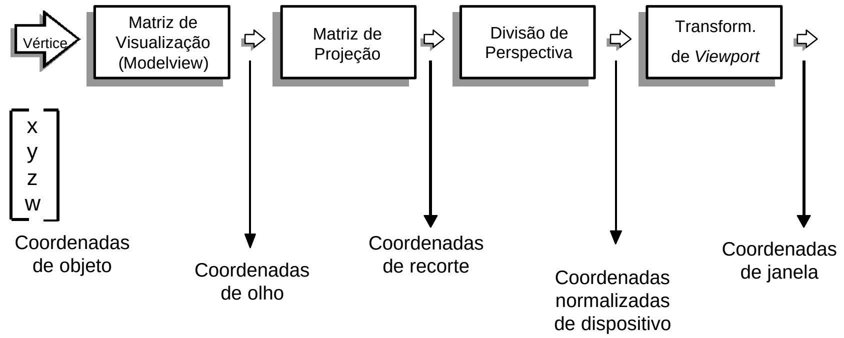


Figura 8.4: Esquema de transformação de coordenadas na OpenGL.

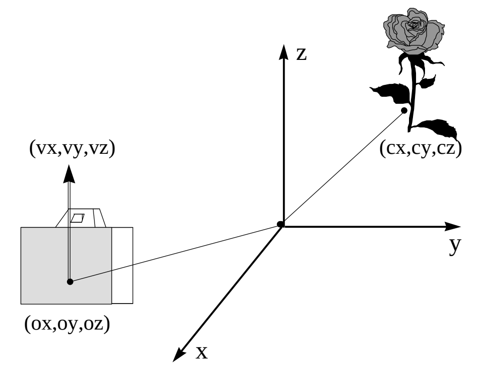


Figura 8.5: Transformação de visualização.

A sintaxe da função de escalamento é:

glScalef (GLfloat x, GLfloat y, GLfloat z);
glScaled (GLdouble x, GLdouble y, GLdouble z);

Onde os parâmetros x, y e z indicam o valor a ser multiplicado sobre cada eixo.

8.1.2. Transformação de Projeção

Esta transformação é utilizada para definir o volume de visualização, os planos de corte e, como o próprio nome diz, determinar o modo de projeção do espaço tridimensional no plano. Em termos de projeção perspectiva, existem duas funções que realizam tal tipo de operação. A primeira é:

glFrustum (GLdouble esquerda, GLdouble direita,
GLdouble topo, GLdouble base,
GLdouble perto, GLdouble longe);

Onde os parâmetros esquerda e direita indicam os limites em **x**, topo e base indicam os limites em **y**, e perto e longe indicam os limites em **z** (fig. 8.6). A palavra *Frustum* se refere a piramide truncada que fica entre os planos perto e longe.

A segunda função, mais usada que a primeira por ser operacionalmente mais intuitiva, é:

gluPerspective (GLdouble acdvy, GLdouble ra, GLdouble perto,GLdouble longe);

Onde acdvy define o ângulo do campo de visão no eixo **y**, ra (razão de aspecto) define a relação **x/y**, para se determinar o campo de visão em **x**, e perto e longe definem os limites ao longo do eixo **z** (fig. 8.7).

A projeção da cena no plano também pode ser realizada de acordo com uma projeção ortográfica. A sintaxe da função que realiza esta operação é:

glOrtho (GLdouble esquerda, GLdouble direita,
GLdouble topo, GLdouble base,
GLdouble perto, GLdouble longe);

Os parâmetros desta rotina funcionam de forma similar ao da função glFrustum.

Note-se que, como explicado anteriormente, as projeções perspectivas permitem que se tenha uma visão mais realística de uma cena, dado que os objetos da cena mais distante do observador aparecerão menores, e, assim, alteram as relações de tamanho e ângulo de lados do objeto. As projeções ortográficas, onde os raios projetores são perpendiculares ao plano de projeção, evitam estas distorções.

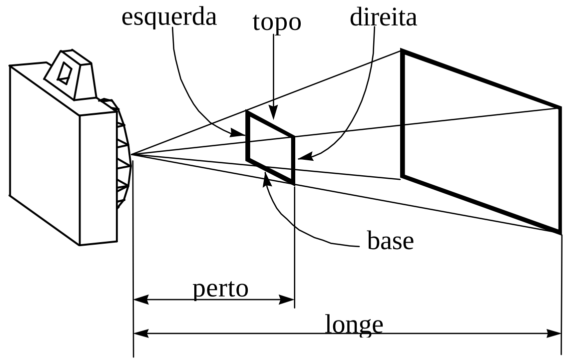


Figura 8.6: Definição do volume de visualização.

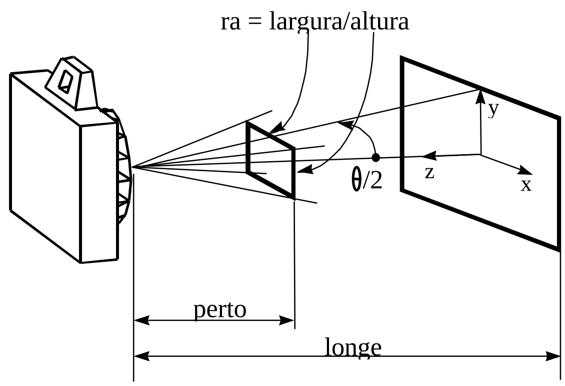


Figura 8.6: Definição do volume de visualização pela rotina gluPerspective.

8.1.3. Transformação de *Viewport*

Depois que todos os vértices sofreram as transformações de *modelview* e de projeção, bem como os vértices fora do volume de visualização foram eliminados, os vértices serão mapeados na janela de exibição através da transformação de *viewport*. A função que define o mapeamento na *viewport* é:

glViewport (GLint x, GLint y, GLsizei largura, GLsizei altura);

Onde (x,y) definem o ponto esquerdo-abaixo da janela de exibição e largura e altura definem as suas dimensões. Caso a relação de aspecto (largura/altura) não for igual a do volume de visualização, a imagem poderá ser exibida com distorção (fig. 8.8).

8.1.4. Matrizes de Transformação

A OpenGL possibilita duas maneiras de se obter transformações: a primeira através do uso de funções específicas para rotação, translação e escala, e a segunda através do cálculo de uma matriz resultante de todas as transformações e a sua aplicação sobre os objetos da cena. Esta biblioteca oferece uma série de funções destinadas para a manipulação direta de matrizes de transformação. Pode-se carregar uma matriz da memória e armazená-la diretamente, sem o uso das funções descritas anteriormente.

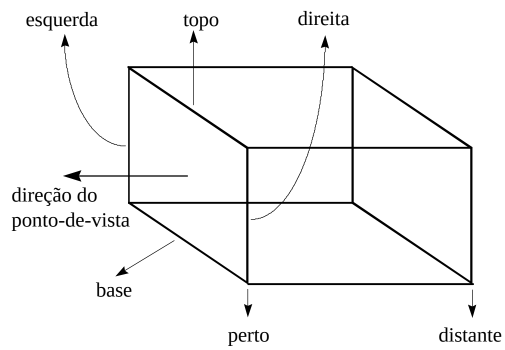


Figura 8.7: Projeção ortográfica.

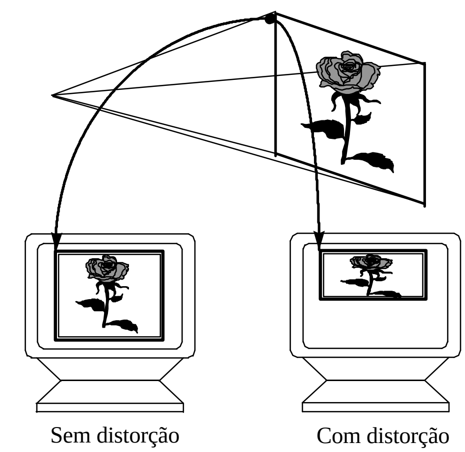


Figura 8.8: Mapeamento do volume de visualização para a *viewport*.

Para se especificar qual tipo de matriz será feita a transformação, existe uma função cuja sintaxe é:

```
glMatrixMode (Glenum modo);
```

Onde o parâmetro modo é uma constante, que pode ser GL_MODELVIEW, GL_PROJECTION ou GL_TEXTURE.

Para se carregar uma matriz, usa-se a seguinte função:

```
glLoadMatrixf (const GLfloat *m);
glLoadMatrixd (const GLdouble *m);
```

Onde o parâmetro m é um vetor de 16 posições, contendo os valores da matriz. Outra função permite que se multiplique a matriz corrente por uma especificada como parâmetro:

```
glMultMatrixf (const GLfloat *m);
glMultMatrixd (const GLdouble *m);
```

Neste caso, m também representa a matriz de transformação. Com isso, a matriz de transformação corrente será a multiplicação da antiga com a especificada pelo parâmetro m.

Há também uma função para zerar a matriz corrente de transformação. Sua sintaxe é:

```
glLoadIdentity ();
```

Desse modo, todas as transformações são perdidas e os próximos desenhos serão feitos na posição (0, 0, 0).

Como exemplo do uso destas matrizes, o código abaixo carrega a matriz identidade e depois aplica uma escala nos três eixos:

```
glMatrixMode (GL_MODELVIEW);
GLfloat m1[]     = {     1.0, 0.0, 0.0, 0.0,
                         0.0, 1.0, 0.0, 0.0,
                         0.0, 0.0, 1.0, 0.0,
```

```
                    0.0, 0.0, 0.0, 1.0 };
glLoadMatrix (m1);
GLfloat m2[]        = {     2.0, 0.0, 0.0, 0.0,
                    0.0, 2.0, 0.0, 0.0,
                    0.0, 0.0, 2.0, 0.0,
                    0.0, 0.0, 0.0, 1.0 };
glMultMatrix (m2);
```

Este outro exemplo mostra como se desenha um cubo rotacionado com transformação perspectiva:

```
glMatrixMode (GL_PROJECTION);
gluPerspective (90.0, 1.0, 0.0, 100.0);
glMatrixMode (GL_MODELVIEW);
glRotatef (30.0, 0.0, 1.0, 0.0);
glTranslatef (0.5, 0.0, 0.0);
auxSolidCube (0.5);
```

Duas funções em OpenGL permitem que se guarde temporariamente a matriz corrente de transformação e que se recupere-a posteriormente. As matrizes podem ser empilhadas e desempilhadas, recuperando-se os seus valores. A grande diferença entre o método de zerar a matriz e empilhar/desempilhar é que no primeiro as transformações anteriores são perdidas, e neste último elas podem ser recuperadas.

Para empilhar a matriz usa-se a função glPushMatrix () e para recuperá-la usa-se a função glPopMatrix ().

Como exemplo, este código desenha uma esfera no centro da tela, outra à sua direita, e outra acima da primeira:

```
glLoadIdentity ();
auxSolidSphere (0.2);
glPushMatrix ();
    glTranslatef (0.5, 0.0, 0.0);
    auxSolidSphere (0.2);
glPopMatrix ();
glTranslatef (0.0, 0.5, 0.0);
auxSolidSphere (0.2);
```

Se não fossem usadas as funções de empilhar e desempilhar na matriz de transformação, seriam desenhadas uma esfera no centro, outra à direita e outra acima desta última, pois as transformações seriam calculadas sucessivamente.

8.2 Esquema de Transformações na Biblioteca Gráfica VRML

O esquema de transformação em VRML é mais simples do que o da OpenGL. Ele se baseia em um eixo de coordenadas 3D fixo e centrado no centro da janela de exibição. Objetos sólidos (cilindros, cubos, etc) são posicionados de forma centralizada em relação a este eixo. Os objetos podem ser deslocados no espaço em relação a este eixo, bem como é possível se definir um ponto-de-observação de forma semelhante ao da OpenGL.

Não há transformação de *viewport*, o que implica que relações de aspecto não podem ser alteradas. Além disto, a VRML adota uma unidade padrão de medida que é o metro, isto para possibilitar a compatibilização de diferentes mundos.

As transformações em VRML são realizadas através do *node* (bloco fundamental de construção de um arquivo VRML) *Transform* que possui alguns campos que definem rotação, translação, escalamento e qualquer combinação das transformações citadas acima. Um *node Transform* afeta todos os nodes que estão dentro dele e ainda possui um efeito acumulativo.

A translação é feita usando-se um campo *translation* de um node *Transform* que especifica três valores de movimentação do objeto ao longo dos eixos **x**, **y** e **z**. A VRML segue a convenção da regra da mão direita para a orientação dos eixos.

O exemplo abaixo descreve como se realizar a translação de -2.4 unidades no eixo **x**, 2 unidades no eixo **y** e 3 unidades no **z**, de um cilindro com 3m de raio da base e 6m de altura em VRML 2.0.

```
#VRML V2.0 utf8
Transform {
```

```
    translation -2.4 2 3
    children[
        Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder {
                radius 3
                height 6
                side TRUE
                top FALSE
                bottom TRUE
            }
        }
    ]
}
```

O campo *rotation* de um *Transform* especifica a rotação de um objeto ao redor de um determinado eixo. Este campo consiste de quatro valores. Os três primeiros definem o eixo de rotação segundo a sequência **x y z**. O último define quantos radianos o objeto deve ser rotacionado em torno do eixo de rotação. Por exemplo, o trecho de código abaixo descreve o mesmo cilindro do exemplo anterior rotacionado de 1.57 radianos ao redor do eixo x.

```
#VRML V2.0 utf8
Transform {
    rotation 1 0 0 1.57
    children[
        Shape {
            ***
        }
    ]
}
```

A rotação pode ser realizada no sentido horário (valor do ângulo de rotação negativo) e no sentido anti-horário (ângulo positivo). Como mencionado anteriormente, deve-se atentar para que quando for se rotacionar um objeto fora da origem dos eixos, deve-se primeiro transladar esta origem para um ponto estabelecido do objeto, realizar a rotação e depois retornar a origem dos eixos a sua posição inicial.

Escalamento aumenta ou diminui o tamanho dos objetos que estiverem dentro de um *node Transform*. Esta transformação é especificada pelo campo *scale*, o qual possui três números que representam a escala nos eixos **x**, **y** e **z** respectivamente. *scale* 1 1 1 significa nenhuma modificação no objeto. *scale* com valor 0 em qualquer direção torna o objeto infinitamente pequeno na direção. Como ilustração, se for alterado parte do último exemplo tem-se:

```
#VRML V2.0 utf8
Transform {
    scale 1.23 1 1
    children[
        Shape {
            ***
        }
    ]
}
```

Nesta situação o objeto será aumentado em 23% na direção do eixo x.

A combinação dos campos do *node Transform* podem realizar as mais complicadas transformações. Quando usa-se uma combinação de transformações deve-se considerar que:

- A ordem em que uma série de transformações é aplicada faz diferença.
- A transformação mais recente é aplicada primeiro.

- A ordem em que uma série de transformações é aplicada dentro de um único node *Transform* é fixa. Primeiro é aplicado o escalamento, depois a rotação e por fim a translação. Se for necessário que isto ocorra em ordem diferente da especificada acima, deve-se usar vários *nodes Transform*.

Abaixo tem-se um exemplo de um *node Transform* que combina todas as transformações possíveis em VRML.

```
#VRML V2.0 utf8
Transform {
    scale           1.23 1 1
    rotation                0 1 0 1.69
    translation     4.5 1 0.4
    children[
        Shape {
            appearance Appearance {
                material Material { }
            }
            geometry Cylinder {
                radius 3
                height 6
                side TRUE
                top FALSE
                bottom TRUE
            }
        }
    ]
}
```

Em relação ao posicionamento do ponto-de-observação, o *node ViewPoint* descreve uma posição potencial e uma orientação para a observação de uma cena. Criar um *ViewPoint* é como posicionar uma câmera em um determinado ponto para que o cenário seja visto a partir dele. Pode-se especificar, também, um campo de visão, indicando o quanto a cena é visível. Isto funciona de forma similar ao parâmetro acdvy na rotina gluPerspective da OpenGL e um pequeno *field of view* deixa parte de um cenário visível, ao contrário de um grande *field of view* que deixa grande parte do cenário visível.

Segundo a especificação da VRML 2.0, o node *ViewPoint* é definido da seguinte maneira:

```
ViewPoint {
    position                0 0 10
    orientation     0 0 1 0
    fieldOfView     0.785398
    description     “ “
    jump            TRUE
}
```

onde,

position - especifica a posição relativa do *node ViewPoint* em relação ao sistema de coordenadas.

orientation - especifica a rotação relativa a uma orientação padrão. A orientação padrão é aquela que o usuário observa a cena percorrendo-a no sentido do eixo -z , com +x à direita e +y para cima. Uma simples rotação da orientação (rotação realizada sobre um eixo arbitrário) é suficiente para especificar qualquer combinação de direções de visão. Pode-se colocar dois campos *orientation* para um *node ViewPoint*. Os campos *position* e *orientation* são afetados pela hierarquia das transformações.

fieldOfView - especifica um campo de visão em radianos. Ele pode variar de 0 a P, sendo que o ângulo pré-assumido é 45º.

description - descreve o ponto de vista que o usuário está utilizando.

jump - indica se o *browser* pode “saltar” para um outro *ViewPoint* que foi definido.

9. Referências

Cap. 1: [CARL78], [MARR82]

Cap. 2: [FOLEY90], [ROGER90]

Cap. 3: [FOLEY90], [ROGER90], [BLINN92a], [BLINN92b]

Cap. 4, 5, 6: [CARL78], [FOLEY90], [GLASS90], [ROGER90], [SNYDE84]

Cap. 7: [PETER96], [KALAM96]

Cap. 8: [NEID93], [BLINN93], [HART96], [SCHR96], [RICH96]

Cap. 9: [SOUTH92], [HODGE92], [ROGER90], [BALLA82], [BAYA95], [SLATER93], [KELSO92]

[BALLA82] Computer Vision. Dana H.Ballard, Christopher M.Brown. Prentice-Hall, 1982.

[BAYA95] Computing non-planar perspectives in real time. Salvador Bayarri, Computers & Graphics 19(3), 1995.

[BLINN93] A Trip Down the Graphics Pipeline: The Homogeneous Perspective Transformation. James F. Blinn, IEEE Computer Graphics & Applications, Março, 1992.

[BLINN92a] Uppers and Downers. James F. Blinn, IEEE Computer Graphics & Applications, Março, 1992.

[BLINN92b] Uppers and Downers II. James F. Blinn, IEEE Computer Graphics & Applications, Maio, 1992.

[CARL78] Planar Geometric Projections and Viewing Transformations. I. Carlbom, J. Paciorek, Computing Surveys, v.10, n.4, dezembro de 1978.

[EZZI93] The Scaling Behavior of Viewing Transformations. S. S. Abi-Ezzi, L. A. Shirman, IEEE Computer Graphics & Applications, Maio, 1993.

[FOLEY90] Computer Graphics - Principles and Practice. J. D. Foley, A. van Dam, S. K. Feiner, J. F. Hughes, Addison-Wesley, 1990, segunda-edicao.

[GLASS90] Graphics Gems. S. Glassner, Academic Press Professional, 1990, paginas 485 a 493.

[HART96] The VRML 2.0 Handbook: Building Moving Worlds on the Web. Jed Hartman, Josie Wernecke, Addison-Wesley Pub Co, 1996.

[HODGE92] Tutorial: time-multiplexed stereoscopic computer graphics. Larry F.Hodges, IEEE Computer Graphics & Applications, 12(2), 92.

[KALAM96] AutoCAD para Desenhos de Engenharia. Alan Kalameja, Makron Books, 1996

[KELSO92] Perspective projection: artificial and natural. Patterson Kelso, Engineering Design Graphics Journal, 56(3), 1993.

[NEID93] OpenGL Programming Guide. Jackie Neider, Tom Davis, Mason Woo, Addison Wesley, 1993.

[PETER96] 3D Studio MAX Fundamentals. Michael Todd Peterson, New Riders Publishing, 1996.

[RICH96] Opengl Superbible: The Complete Guide to Opengl Programming for Windows Nt and Windows 95. Richard S., Jr Wright, Michael Sweet, Waite Group Pr, 1996.

[ROGER90] Mathematical Elements for Computer Graphics. David F. Rogers, J. Alan Adams, McGraw-Hill, segunda edicao, 1990

[SCHR96] The Visualization Toolkit: An Object-Oriented Approach to 3-D Graphics. Will Schroeder, Bill Lorenson, William Schroeder, Prentice Hall Computer Books, 1996

[SLATER93] Simulating peripheral vision in immersive virtual environments. Mel Slater, Martin Usoh, Computers & Graphics 17(6), 1993

[SOUTH92] Transformations for stereoscopic visual simulation. David A.Southard, Computers & Graphics 16(4), 1992

[SNYDE84] Map projections used by the US Geological Survey. J. P. Snyder, US Government printing Office, Washington, segunda edicao, 1984