MANIPULAÇÃO 3D

Todas as manipulações de gráficos podem ser representadas em forma de equação[ARTWICK, 1984]. O problema é que manipulações de gráficos normalmente envolvem muitas operações de aritmética simples. Matrizes são usadas porque são mais fáceis de manipular e entender do que as equações que representam, e isso explica por que os programadores e engenheiros as usam extensivamente [ARTWICK, 1984].”
Devido ao padrão de coordenadas usualmente adotado para representação de pontos no espaço tridimensional, (x,y,z) pode ser conveniente manipular estes valores em pequenas entidades de 3 elementos. A representação por matriz, apresenta-se como um modo mais simples e mais limpo do que a forma algébrica. As matrizes são também parecidas com o modelo organizacional da memória dos computadores. Assim, suas representações se correlacionam diretamente com estas estruturas de armazenamento, facilitando muito o trabalho de transposição do programador.

ARITIMÉTICA DE VETORES

A forma mais simples de matriz é o vetor em linha com dois ou mais elementos colocados lado a lado e envolvidos por chaves [x y]. Os elementos são independentes um do outro e as chaves marcam o começo e o fim dos elementos. Usualmente nos referimos a um par de coordenadas (x,y) para representar a posição de um ponto num plano. Assim sendo, passaremos a representar o ponto no plano pelo vetor [x y] em lugar de seu usual par de coordenadas (x,y). Podemos estender esta representação para 3 dimensões e descrever o ponto num sistema de coordenadas (x,y,z) como o vetor [x y z].

Pontos definidos num espaço multi-dimensional podem ser transladados (movimentados) pela adição ou subtração de valores de translação às suas coordenadas. Assim, o ponto A definido por (x,y,z) pode ser "reposicionado" pelo uso de fatores de translação, como se segue:

$$x1 = x + Tx;$$
$$y1 = y + Ty;$$
$$z1 = z + Tz; \quad (1)$$

Onde Tx, Ty, e Tz são valores de translação nos respectivos eixos e (x1,y1,z1) é novo posicionamento do ponto A. Se utilizarmos a notação matricial, esta translação se apresentará da seguinte forma:

$$[x1\ y1\ z1] = [x\ y\ z] + [Tx\ Ty\ Tz] \quad (2)$$

Multiplicação por vetor

Em gráficos, e em situações do mundo real, é freqüentemente necessário calcular-se somas de produtos. A multiplicação de vetores é realmente mais que só uma multiplicação, é uma operação de soma de produtos. Note-se também que o resultado não é um vetor, mas um único valor (escalar). Assim sendo, a definição oficial de multiplicação de vetores é:

$$a = [b1\ \ b2\ \ b3] * \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix}$$

$$a = (b1 * c1) + (b2 * c2) + (b3 * c3) \quad 3)$$

Multiplicação por Escalar

A transformação de escala, como a translação, é uma operação comumente executada em pontos. Para se fazer com que uma imagem definida por um conjunto de pontos mude de tamanho requer-se apenas que se multiplique todos os valores de suas coordenadas pelos apropriados valores de escala. A representação matricial desta função é:

$$[x'\ \ y'\ \ z'] = S * [x\ \ y\ \ z] \quad (4)$$

onde S é o fator de escala para os 3 eixos, [x' y' z'] é o ponto resultado e [x y z] é o ponto original.

Esta operação é equivalente a:

$$x' = S * x,$$
$$y' = S * y,$$
$$z' = S * z, \qquad (5)$$

ARITIMÉTICA DE MATRIZES RETANGULARES

Matrizes Retangulares são números ordenados em "m" filas por "n" colunas. Se "m" é iguala "n", a matriz é quadrada. A própria matriz é simplesmente um retângulo cheio de números. As operações algébricas realizadas sobre as matrizes é que dão a elas o seu significado.

A adição e subtração de matrizes retangulares são executadas somando-se os elementos de uma matriz aos elementos correspondentes da outra matriz. Ambas as matrizes devem ser do mesmo tamanho. A adição de matrizes em computação gráfica é normalmente limitada a matrizes de linha ou vetores; adições de matrizes retangulares são raramente utilizadas [ARTWICK, 1984].

Existem dois tipos de multiplicação de matrizes retangulares: multiplicação de escalar por matriz e multiplicação de matriz por matriz. Na **multiplicação por escalar** um único número é multiplicado por todos os elementos da matriz, gerando uma nova matriz.

Em computação gráfica a multiplicação por um escalar traduz-se na movimentação de um ponto no espaço tridimensional, para longe ou para perto da origem do sistema de coordenadas (0,0,0).

A **multiplicação de matriz por matriz** é uma operação de soma de produtos. O resultado é sempre uma matriz. O multiplicador e o multiplicando podem ser matrizes de tamanhos diferentes, no entanto o número de colunas do multiplicando deve coincidir com o número de linhas do multiplicador, e a matriz resultado possui o número de linhas do multiplicando e o número de colunas do multiplicador, com se segue:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} * \begin{bmatrix} G & H \\ I & J \\ K & L \end{bmatrix}$$
$$A = (a * G) + (b * I) + (c * K)$$
$$B = (a * H) + (b * J) + (c * L)$$
$$C = (d * G) + (e * I) + (f * K)$$
$$B = (d * H) + (e * J) + (f * L) \qquad (6)$$

Definições importantes relativas à multiplicação de matrizes retangulares:

a) a multiplicação de matrizes é uma operação **não comutativa**, isto é, o produto M*N é geralmente diferente de N*M.

b) a multiplicação de matrizes é uma operação **associativa**, isto é:(M*N)*O = M*(N * O).

c) a multiplicação de matrizes é **distributiva**, isto é: A*(B+C) = AB + AC.

TRANSFORMADAS TRIDIMENSIONAIS

A **translação** de um ponto no espaço tridimensional pode ser obtida pela adição de um vetor de deslocamento à posição atual do ponto. A seguir ilustra-se os benefícios da visualização desta operação pelo uso da representação matricial. Seja o ponto P definido pelas coordenadas (x,y,z). A translação de P em função do vetor deslocamento D definido por (x1, y1, z1) é:

Forma algébrica:

$$\begin{aligned} x1 &= x + Tx, \\ y1 &= y + Ty, \\ z1 &= z + Tz, \end{aligned} \quad (7)$$

Forma matricial:

$$P1[x1 \;\; y1 \;\; z1] = P[x\,y\,z] + D[Tx\;Ty\;Tz] \quad (8)$$

Ou simplesmente:

$$P1 = P + D \quad (9)$$

A **escala** de um ponto no espaço tridimensional pode ser obtida pela multiplicação dos fatores de escala ao ponto. O uso de um pequeno artifício matemático ilustra a transposição da forma algébrica à forma matricial.

Forma algébrica:

$$\begin{aligned} x' &= (x * Sx), \\ y' &= (y * Sy), \\ z' &= (z * Sz), \end{aligned} \quad (10)$$

Aplica-se um pequeno "truque" matemático:

$$\begin{aligned} x' &= (x * Sx) + (y * 0) + (z * 0), \\ y' &= (x * 0) + (y * Sy) + (z * 0), \\ z' &= (x * 0) + (y * 0) + (z * Sz), \end{aligned} \quad (11)$$

Obtendo-se a forma matricial:

$$P'[x' \;\; y' \;\; z'] = P[x \;\; y \;\; z] * Sxyz \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & Sz \end{bmatrix} \quad (12)$$

Ou simplesmente:

$$P' = P * Sxyz \quad (13)$$

A **rotação** de um ponto no espaço tridimensional pode ser obtida pela multiplicação dos fatores de rotação ao ponto. A aplicação da matriz de rotação é dependente do eixo sobre o qual se efetua a rotação.

Rotação ao redor do **eixo Z**

Forma algébrica:

$$\begin{aligned} x' &= (x * \cos(ang)) + (y * (-sen(ang))), \\ y' &= (x * sen(ang)) + (y * \cos(ang)), \\ z' &= z, \end{aligned} \quad (14)$$

Nosso pequeno "truque" matemático:

$$\begin{aligned} x' &= (x * \cos(ang)) + (y * (-sen(ang)) + (z * 0), \\ y' &= (x * sen(ang)) + (y * \cos(ang)) + (z * 0), \\ z' &= (x * 0) + (y * 0) + (z * 1), \end{aligned} \quad (15)$$

Forma matricial:

$$P'[x' \quad y' \quad z'] = P[x \quad y \quad z] * Rz \begin{bmatrix} \cos(ang) & -sen(ang) & 0 \\ sen(ang) & \cos(ang) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (16)$$

Ou simplesmente:

$$P' = P * Rz \quad (17)$$

Rotação ao redor do **eixo X**

Forma algébrica:

$$\begin{aligned} x' &= x, \\ y' &= (y * \cos(ang)) + (z * -sen(ang)), \\ z' &= (y * sen(ang)) + (z * \cos(ang)), \end{aligned} \quad (18)$$

Nosso pequeno "truque" matemático:

$$\begin{aligned} x' &= (x * 1) + (y * 0) + (z * 0), \\ y' &= (x * 0) + (y * \cos(ang)) + (z * -sen(ang)), \\ z' &= (x * 0) + (y * sen(ang)) + (z * \cos(ang)), \end{aligned} \quad (19)$$

Forma matricial:

$$P'[x' \quad y' \quad z'] = P[x \quad y \quad z] * Rx \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(ang) & -sen(ang) \\ 0 & sen(ang) & \cos(ang) \end{bmatrix} \quad (20)$$

Ou simplesmente:

$$P' = P * Rx \quad (.21)$$

Rotação ao redor do **eixo Y**

Forma algébrica:

$$\begin{aligned} x' &= (x * \cos(ang)) + (z * -sen(ang)), \\ y' &= y, \\ z' &= (x * sen(ang)) + (z * -\cos(ang)), \end{aligned} \quad (22)$$

Nosso pequeno "truque" matemático:

$$\begin{aligned} x' &= (x * \cos(ang)) + (y * 0) + (z * -sen(ang)), \\ y' &= (x * 0) + (y * 1) + (z * 0), \\ z' &= (x * sen(ang)) + (y * 0) + (z * -\cos(ang)), \end{aligned} \quad (23)$$

Forma matricial:

$$P'[x' \quad y' \quad z'] = P[x \quad y \quad z] * Ry \begin{bmatrix} \cos(ang) & 0 & -sen(ang) \\ 0 & 1 & 0 \\ sen(ang) & 0 & -\cos(ang) \end{bmatrix} \quad (24)$$

Ou simplesmente:

$$P' = P * Ry \quad (25)$$

O processo de combinar duas matrizes é chamado "**concatenação**" e é executado multiplicando as duas matrizes multiplicadoras ANTES de aplicá-la aos multiplicando. Este processo é especialmente produtivo quando deseja-se aplicar uma certa escala e translação a um conjunto de pontos. Deve-se no entanto ter em mente que a ordem de aplicação das transformações afeta o produto afinal, como bem se ilustra [ARTWICK, 1984] na Figura 3-1 e Figura 3-2 onde foram aplicados os mesmos fatores de escala e translação, em ordens inversas, obtendo-se resultados completamente diferentes.

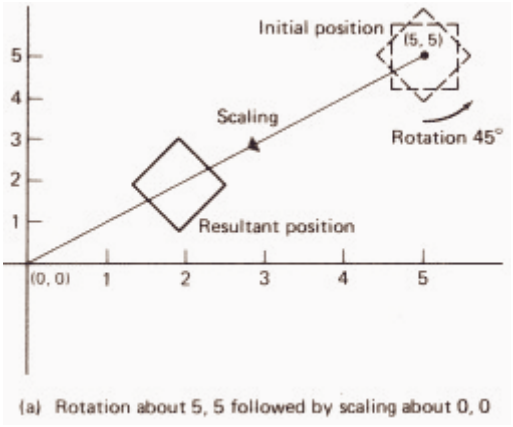

Figura 1Seqüência de rotação e escala [ARTWICK, 1984]

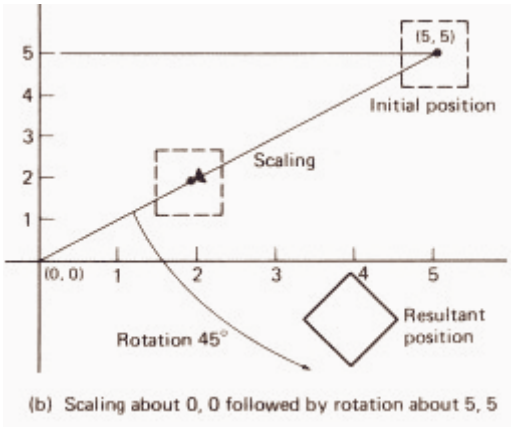

Figura -2Seqüência de escala e rotação: [ARTWICK, 1984]

Até mesmo a ordem da aplicação dos fatores de rotação influencia no resultado final conforme ilustra [ARTWICK, 1984] a Figura 3

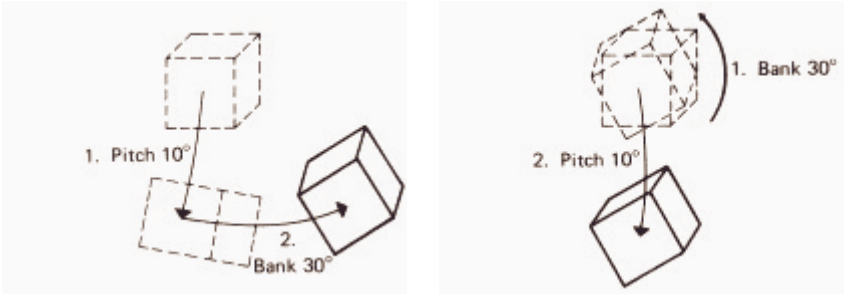

Figura 3 -A inversão na seqüência de rotação provoca resultados diferentes[ARTWICK, 1984]

Rotações são usadas de duas maneiras em computação gráfica: rotacionar objetos no espaço, e girar ou rotacionar o próprio espaço. A rotação de um único objeto é muito usada em animação, onde objetos são manipulados e movidos. Nestes casos, são aplicadas translação, escala e rotação de objetos. Em aplicações típicas de simulação, os objetos permanecem estáticos no espaço e o observador (o piloto de um simulador de vôo, por exemplo) manipula a sua visão do espaço. Numa analogia simples podemos dizer que o observador permanece parado e movimentamos o espaço a seu redor. Em um sistema de CAD ou mesmo num jogo onde os objetos da cena se movem deve-se manter uma tabela de translação, escala e rotação para cada um destes objetos.

COORDENADS HOMOGÊNEAS

As operações de translação, rotação e escala podem ser facilmente executadas com o uso de matrizes. No entanto, enquanto as operações de rotação e escala podem ser concatenadas numa única matriz (pela multiplicação prévia de suas respectivas matrizes) as operações de translação ainda têm de ser conduzidas em separado, uma vez que sua aplicação depende de uma soma matricial.

Com o objetivo de otimizar a aplicação destas operações transformadas foi concebido um sistema de coordenadas denominado **coordenadas homogêneas**. Quando tratamos de representar um ponto no espaço 3D no sistema cartesiano fazemos uso das coordenadas (x,y,z) que posicionam o ponto em relação ao centro de coordenadas. O sistema de coordenadas homogêneas utiliza-se de 4 valores para expressar um ponto P(x,y,z,M). A transposição do sistema homogêneo para o cartesiano se dá pela seguinte relação: (x,y,z)=(x/M, y/M, z/M).

Existem infinitas coordenadas homogêneas para um mesmo ponto. Os pontos onde M=0 são, por definição, os pontos fora do espaço dimensional. Tudo isso poderia parecer muito pouco útil não fosse uma pequena característica das operações matriciais que permite expressar a aplicação da matriz de translação como uma multiplicação quando se expressa o ponto no sistema de coordenadas homogêneo.

Define-se todo ponto como (x,y,z,1) no sistema homogêneo, fazendo com que sua transposição para o espaço cartesiano seja direta (divisão por 1). A aplicação da translação por multiplicação de matrizes se dá então da seguinte forma:
Forma matricial

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Txyz \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (26)$$

Forma algébrica:

$$\begin{aligned} x' &= (x*1)+(y*0)+(z*0)+(1*Tx); \\ y' &= (x*0)+(y*1)+(z*0)+(1*Ty); \\ z' &= (x*0)+(y*0)+(z*1)+(1*Tz); \\ M &= (x*0)+(y*0)+(z*1)+(1*1); \end{aligned} \quad (27)$$

Assim o resultado algébrico é equivalente ao obtido anteriormente com o uso da adição matricial mas permite agora concatenar a matriz de translação aos demais componentes de escala e rotação que assumem então as seguinte formas para o sistema homogêneo.

Escala no eixo X, Y e Z:

Forma matricial

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Sxyz \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (28)$$

Forma algébrica:

$$\begin{aligned} x' &= (x*Sx)+(y*0)+(z*0)+(1*0); \\ y' &= (x*0)+(y*Sy)+(z*0)+(1*0); \\ z' &= (x*0)+(y*0)+(z*Sz)+(1*0); \\ M &= (x*0)+(y*0)+(z*0)+(1*1); \end{aligned} \quad (29)$$

Rotação no eixo Z:

Forma matricial

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Rz \begin{bmatrix} \cos(rz) & -sen(rz) & 0 & 0 \\ sen(rz) & \cos(rz) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (30)$$

Forma algébrica:

$$\begin{aligned} x' &= (x * \cos(rz)) + (y * (-sen(rz))) + (z * 0) + (1 * 0); \\ y' &= (x * sen(rz)) + (y * \cos(rz)) + (z * 0) + (1 * 0); \\ z' &= (x * 0) + (y * 0) + (z * 1) + (1 * 0); \\ M &= (x * 0) + (y * 0) + (z * 0) + (1 * 1); \end{aligned} \quad (31)$$

Rotação no eixo X:

Forma matricial

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Rx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(rx) & -sen(rx) & 0 \\ 0 & sen(rx) & \cos(rx) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (32)$$

Forma algébrica

$$\begin{aligned} x' &= (x * 1) + (y * 0) + (z * 0) + (1 * 0); \\ y' &= (x * 0) + (y * \cos(rx)) + (z * (-sen(rx))) + (1 * 0); \\ z' &= (x * 0) + (y * sen(rx)) + (z * \cos(rx)) + (1 * 0); \\ M &= (x * 0) + (y * 0) + (z * 0) + (1 * 1); \end{aligned} \quad (33)$$

Rotação no eixo Y:

Forma matricial:

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Ry \begin{bmatrix} \cos(ry) & 0 & sen(ry) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(ry) & 0 & \cos(ry) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (34)$$

Forma algébrica:

$$\begin{aligned} x' &= (x * \cos(ry)) + (y * 0) + (z * sen(ry)) + (1 * 0); \\ y' &= (x * 0) + (y * 1) + (z * 0) + (1 * 0); \\ z' &= (x * (-sen(ry))) + (y * 0) + (z * \cos(rx)) + (1 * 0); \\ M &= (x * 0) + (y * 0) + (z * 0) + (1 * 1); \end{aligned} \quad (35)$$

Utilizando-se de rotinas eficientes de multiplicação de matrizes pode-se agora concatenar todas as matrizes de transformação numa única matriz global de transformação que será aplicada a todos os pontos do desenho.A rotina em Java a seguir ilustra a implementação computacional de uma multiplicação de matrizes de tamanho (4 x 4) intensamente utilizada quando se trabalha com coordenadas homogêneas.A matriz 4x4 "A[][]" é o multiplicando, "B[][]" é o multiplicador e "C[][]" armazena o resultado.

```
void MultiplyMatrix(double A[][], double B[][], double C[][])
   {
   int i, j, k;
   double Acc;
```

```
for(i=0; i<4; i++)
   {
   for(j=0; j<4; j++)
      {
      Acc=0;
      for(k=0; k<4; k++)
         Acc+=A[i][k]*B[k][j];
      C[i][j]=Acc;
      }
   }
}
```

As rotinas em Java a seguir ilustram o fluxo computacional utilizado na preparação das matrizes de translação, na sua concatenação e na sua aplicação aos pontos do espaço tridimensional.

1) Definição das estruturas de armazenamento de Translação:

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Txyz \begin{bmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (36)$$

```
void SetTransMatrix(double tx, double ty, double tz)
   {
   int i;
   ZeroMatrix(T); // Preenche matriz [4 x 4] com zeros
   for(i=0; i<4; i++) // Preenche diagonal principal com 1
      T[i][i]=1.0;
   T[0][3]=-tx;
   T[1][3]=-ty;
   T[2][3]=-tz;
   }
```

2) Definição das estruturas de armazenamento de Escala:

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Sxyz \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (37)$$

```
void SetScaleMatrix(double sx, double sy, double sz)
   {
   ZeroMatrix(S); // Preenche matriz [4 x 4] com zeros
   S[0][0]=sx;
   S[1][1]=sy;
   S[2][2]=sz;
   S[3][3]=1;
   }
```

3) Definição das estruturas de armazenamento de Rotação em x, y e z

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Rx \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(rx) & -sen(rx) & 0 \\ 0 & sen(rx) & \cos(rx) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (38)$$

```
void SetRotateMatrix(int Eixo, double Theta)
   {
   double c, s;
```

```
    c=CosD(Theta);
    s=SinD(Theta);

            if(Eixo==EIXO_X)
        {
        ZeroMatrix(RX);
        RX[0][0]=1.0;
        RX[3][3]=1.0;
        RX[1][1]=RX[2][2]=c;
        RX[2][1]=-s;
        RX[1][2]=s;
        }
```

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Ry \begin{bmatrix} \cos(ry) & 0 & sen(ry) & 0 \\ 0 & 1 & 0 & 0 \\ -sen(ry) & 0 & \cos(ry) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (39)$$

```
else if(Eixo==EIXO_Y)
        {
        ZeroMatrix(RY);
        RY[1][1]=1.0;
        RY[3][3]=1.0;
        RY[0][0]=RY[2][2]=c;
        RY[2][0]=RY[0][2]=-s;
        }
```

$$P'[x' \quad y' \quad z' \quad M] = P[x \quad y \quad z \quad 1] * Rz \begin{bmatrix} \cos(rz) & -sen(rz) & 0 & 0 \\ sen(rz) & \cos(rz) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (40)$$

```
//
 else if(Eixo==EIXO_Z)
        {
        ZeroMatrix(RZ);
        RZ[2][2]=1.0;
        RZ[3][3]=1.0;
        RZ[0][0]=RZ[1][1]=c;
        RZ[1][0]=-s;
        RZ[0][1]=s;
        }   }
```

4) Preparação da matriz "M" de transformação combinada de escala, rotação (x,y,z) e translação

```
void PrepareMatrix()
    {
    double M1[][] = new double[4][4];
    double M2[][] = new double[4][4];
    double M3[][] = new double[4][4];
     // Concatena Rotação em X e Escala e armazena em
     MultiplyMatrix(RX, S, M1); M1
    // Concatena Rotação em Y e armazena em M2
    MultiplyMatrix(RY, M1, M2);
    // Concatena Rotação em Z e armazena em M3
    MultiplyMatrix(RZ, M2, M3);
    // Concatena Translação e armazena em M
MultiplyMatrix( T, M3, M);
}
```

5) Sendo o ponto definido por:

```
class POINT3D
   {
   double x,y,z;

   POINT3D(double X, double Y, double Z)
      {
      x=X; y=Y; z=Z;
      }
   }
```

6) Finalmente, multiplica-se todos os pontos pela matriz composta de transformação

```
void Transform(POINT3D A, POINT3D B)
   {
  // Multiplica a matriz concatenada de transformações M ao ponto A e armazena em B
   B.x=M[0][0]*A.x+M[0][1]*A.y+M[0][2]*A.z+M[0][3];
   B.y=M[1][0]*A.x+M[1][1]*A.y+M[1][2]*A.z+M[1][3];
   B.z=M[2][0]*A.x+M[2][1]*A.y+M[2][2]*A.z+M[2][3];
   }
```

PROJEÇÕES

PROJEÇÕES ORTOGONAL

Projeções são provavelmente a classe mais importante de transformações em computação gráfica pois permitem a visualização planar de objetos tridimensionais. O tipo mais simples de projeção é a projeção ortogonal ou paralela, onde a imagem de um ponto é definida como a projeção normal deste ponto no plano de projeção. A Figura 4 ilustra a projeção de um sólido no plano (x,y), note-se que as coordenadas (x,y) do sólido projetado permanecem idênticas às coordenadas do sólido no espaço. A projeção ortográfica é obtida pela simples eliminação do componente "z".

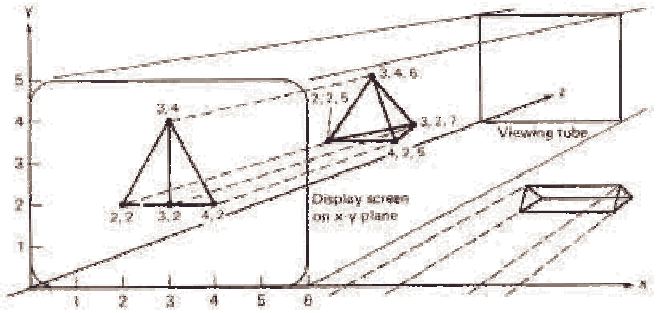

Figura 4 - Projeção ortogonal de um sólido no plano (x,y): [ARTWICK, 1984]

Projeções ortográficas são simples, porém algumas de suas características nos levam, quase sempre, a adotar esquemas mais sofisticados de projeção. Embora projeções ortogonais sejam fáceis de obter-se, seus resultados carecem de realismo, a menos que o observador esteja a uma distância muito grande dos objetos em cena. Quando a ordem de grandeza desta distância é equivalente às dimensões dos objetos, é recomendável o uso de projeções em perspectiva.

PROJEÇÕES EM PERSPECTIVA

A projeção em perspectiva considera a profundidade como elemento relevante na sua formação e por isso apresenta um resultado mais familiar ao observador humano. O sistema de coordenadas mais favorável às projeções em perspectiva adotam um "ponto de fuga" no centro da tela, isto é, todas as linhas do desenho apontam para este ponto de fuga, criando uma "sensação" de profundidade ao observador humano. A Figura 5 ilustra a visualização de uma caixa através da projeção ortogonal, assinalando a utilização da técnica de “ponto de fuga” no traçado da projeção.

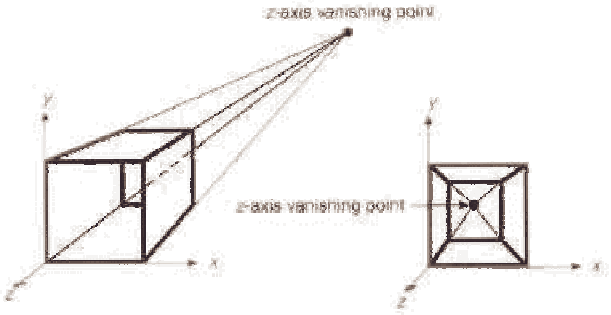

Figura 5 - Projeção em perspectiva de um sólido: [FOLEY, 1993]

A projeção em perspectiva é feita através de cálculos simples de "semelhança de triângulos". Objetos no mundo real refletem a luz ambiente em todas as direções. Parte desta luz refletida atinge a córnea e é projetada na retina ocular. A Figura 6 ilustra a geometria utilizada para geração da projeção em perspectiva de um ponto no espaço (*point to be projected*) no plano de projeção (*projection plane*).

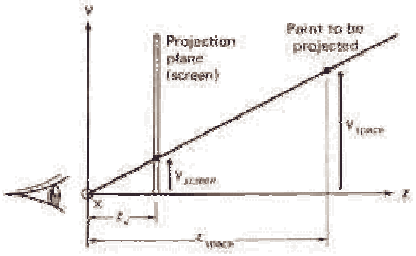

Figura 6 - Semelhança de triângulos na projeção em perspectiva: [ARTWICK, 1984]

As equações simplificadas para o cálculo da projeção ilustrada na Figura 6 são:

$$Xscreen = \frac{Xspace}{Zspace} * Zv$$
$$Yscreen = \frac{Yspace}{Zspace} * Zv \quad (1)$$

Note que na projeção ilustrada pela Figura 6 o parâmetro *Zspace* foi incluído no cálculo da projeção. Assim quanto mais um objeto se distancia do plano de projeção, menor o valor resultante da divisão, assim sendo os pontos localizados a uma grande distância do plano de projeção serão levados ao centro do sistema de coordenadas. Em virtude desta característica costuma-se adotar o centro do plano de projeção como origem do sistema de coordenadas de tela, ao contrário da usual origem inferior à esquerda.

Os limites do plano de projeção limitam o ângulo de visão do observador. A distância do observador ao plano de projeção determina o seu ângulo de visão. A prática demonstra que ângulos de visada entre 40 e 60 graus proporcionam experiências visuais bastante próximas da realidade. Projeções mais "impressionantes" podem ser obtidas com ângulos superiores a 90 graus. Ao contrário, quando a distância do observador à tela aumenta, o seu campo de visão se estreita e a projeção em perspectiva dos pontos no espaço aproxima-se dos resultados obtidos pela projeção ortogonal.

Muitas vezes é desejável que o observador possa observar o espaço tridimensional sob diferentes ângulos de visão. Neste caso um movimento do observador equivalente à uma câmara cinematográfica em movimento é adequado. No entanto os cálculos necessários à movimentação do plano de projeção e do observador no espaço são complexos e demandam processamento intensivo. Uma opção com efeito visual equivalente ao recalculo da nova posição do plano de projeção é a aplicação do movimento da câmara inversamente nos objetos no espaço. Desta maneira o movimento aplicado aos objetos apresenta-se ao observador como um movimento de câmara em virtude da ausência de outros pontos de referência. Afortunadamente o controle e o cálculo deste tipo de movimento é muito menos custoso do que o controle da movimentação do observador e do plano de projeção.

Considerando-se um sistema de coordenadas (x,y,z) conforme ilustrado na Figura 4 podemos simular movimentos de câmara através de rotações em torno do eixo x, y ou z aplicando-se as mesmas matrizes de rotação apresentadas na Seção que falamos de rotações.

Utilizando-se da **concatenação** das matrizes de rotação chega-se a uma complexa representação algébrica equivalente ao seguinte algoritmo, implementado em Java, que calcula a projeção de um ponto determinado por (P1.x, P1.y, P1.z) em função da distância do observador à tela (Offset Z) e dos ângulos de **azimute**, **elevação** e **rolagem** da câmara:

```
double ProjectToScreen(POINT3D P1,POINT3D P2)
   {
   double XA,YA,ZA;
   double Xt,Yt,Zt;
   double OneOverYt;

   XA=CosAzim*P1.x-SinAzim*P1.y;
   YA=SinAzim*P1.x+CosAzim*P1.y;
   ZA=CosRool*P1.z-SinRool*XA;
   Xt=CosRool*XA+SinRool*P1.z;
   Yt=CosElev*YA-SinElev*ZA;
   Zt=OffSetZ+SinElev*YA+CosElev*ZA;
   if(Yt != 0) // Distancia à tela
      {
      OneOverYt=1.0/Yt;
      P2.x=FocalDist*Xt*OneOverYt;
      P2.y=FocalDist*Zt*OneOverYt;
      }
   return (Yt);
   }
```