Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #4: Transformações

Objetivos

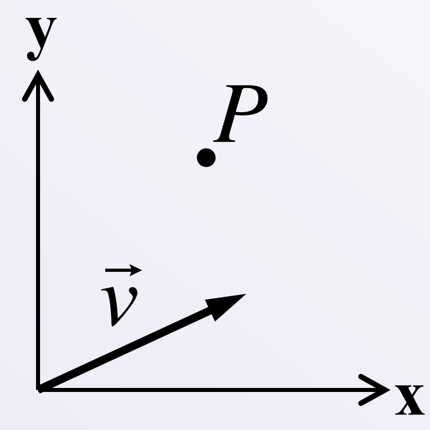
- Apresentar transformações padrões
  - Translação
  - Rotação
  - Escala
  - Cisalhamento
- Sistema de coordenadas homogêneas
- Aprender a montar matrizes de transformação arbitrárias

Pontos e Vetores (2D)

- Ponto
  - Denota posição no plano
- Vetor
  - Denota deslocamento, isto é, inclui a noção de direção e magnitude
- Ambos são normalmente expressos por pares de coordenadas (em 2D) mas <u>não são a "mesma coisa"</u>

$$P = (x_P, y_P)$$

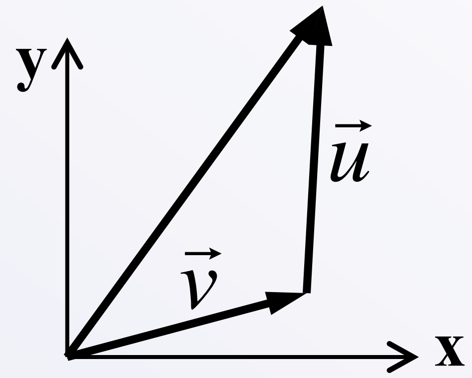
$$\vec{v} = (x_v, y_v)$$

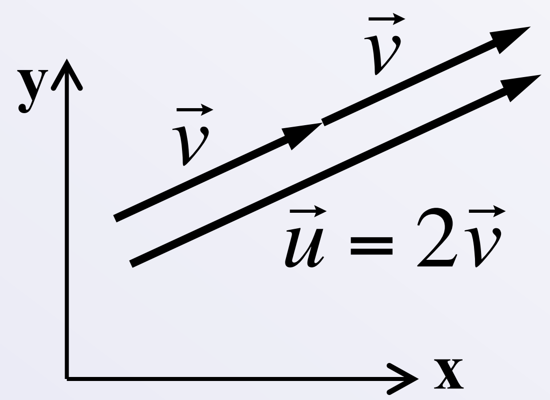


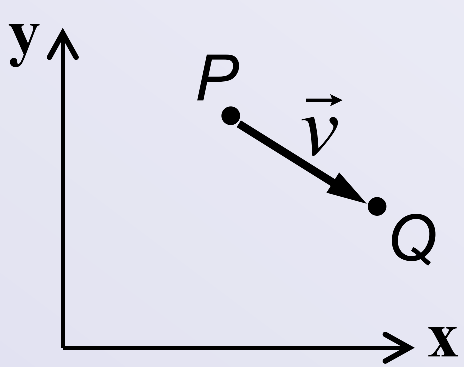
Operações com Pontos e Vetores

- Soma de vetores
- Multiplicação de vetor por escalar
- Subtração de pontos
- Soma de ponto com vetor

$$\vec{t} = \vec{v} + \vec{u}$$

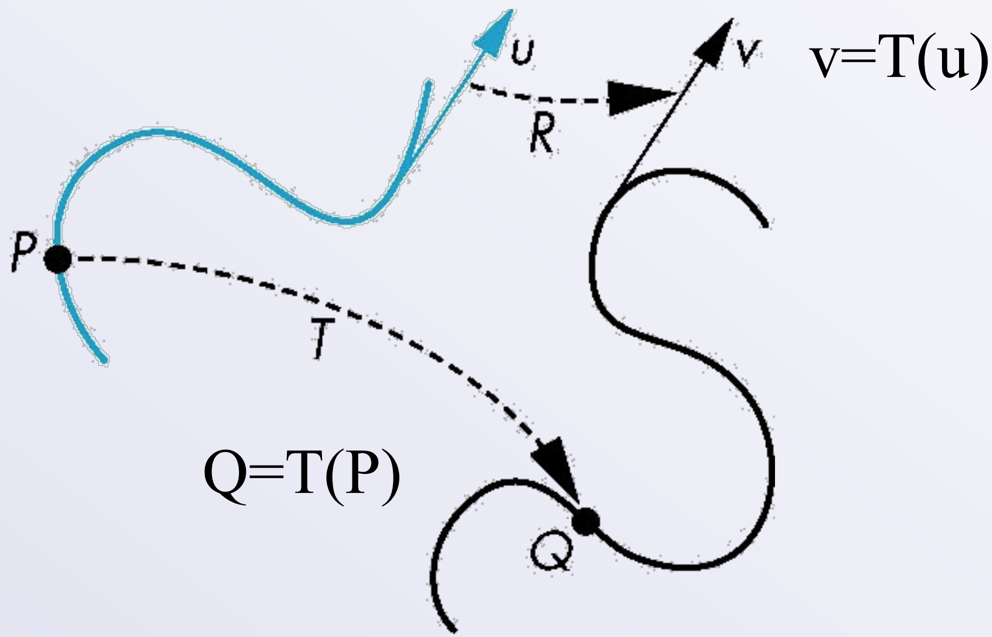






Transformações

- Transformação é uma função que mapeia pontos (vetores) entre espaços vetoriais



Transformações lineares

- Tipo de transformação que preserva as operações de adição vetorial e multiplicação por escalar

$$(\forall v, w \in V) : T(v + w) = T(v) + T(w)$$

$$(\forall \alpha \in K)(\forall v \in V) : T(\alpha v) = \alpha T(v)$$

$$T(x) = 3x$$
$$T(x, y) = x + y$$
$$T(x, y) = (3x + y, 2x - 2y)$$
$$T(x) = x + a$$

Transformações lineares

- Se uma transformação é linear, então
  - Se um conjunto de pontos está contido em uma reta, depois de transformados eles também estarão contidos sobre uma reta.
  - Se um ponto P guarda uma relação de distância com dois outros pontos Q e R, então essa relação de distância é mantida pela transformação.
- Transformação mapeia origem na origem?
  - Sim: Transformação *Linear*
  - Não: Transformação *Linear Afim*: Translações são permitidas

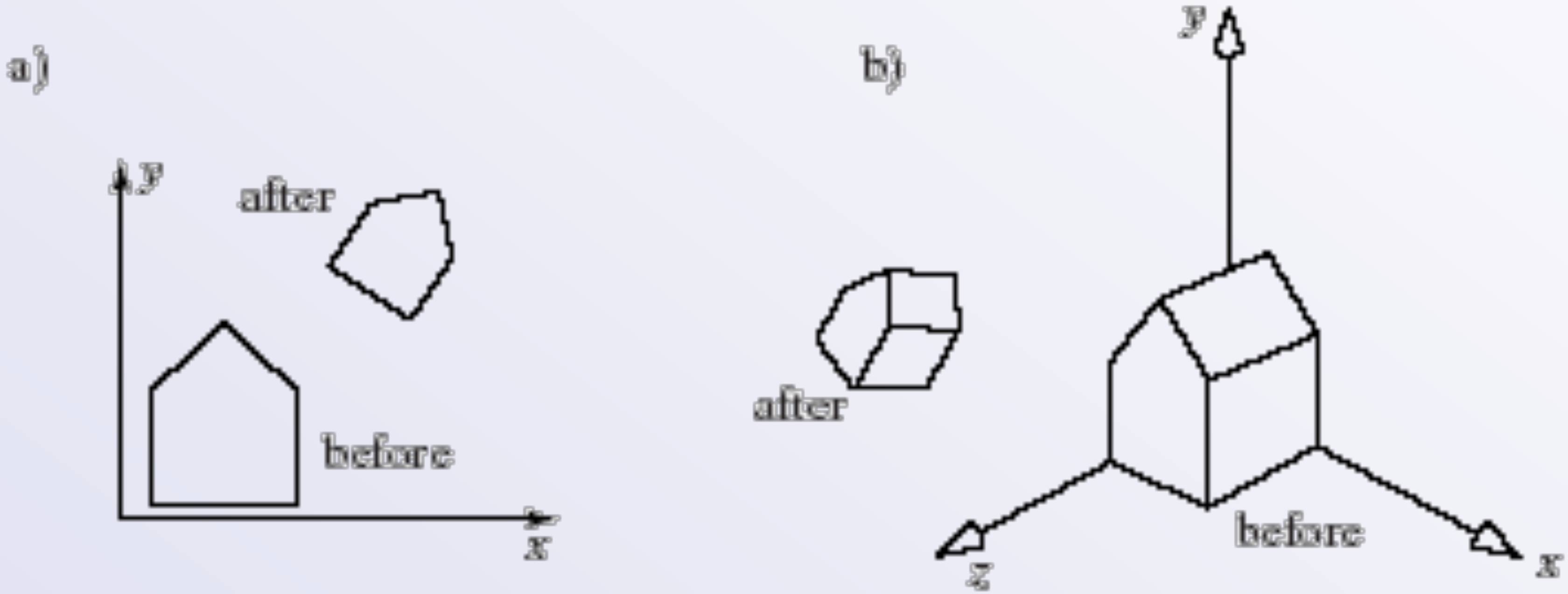
Transformações afins

$$x \mapsto Ax + b$$

- Preservam colinearidade
- Preservam razões entre distâncias
- Representam muitas das transformações físicas:
  - Transformações de corpo rígido: rotações, translações
  - Escalas, cisalhamento
- Precisamos somente transformar os vértices dos segmentos de linha e deixar que a implementação desenhar o restante;

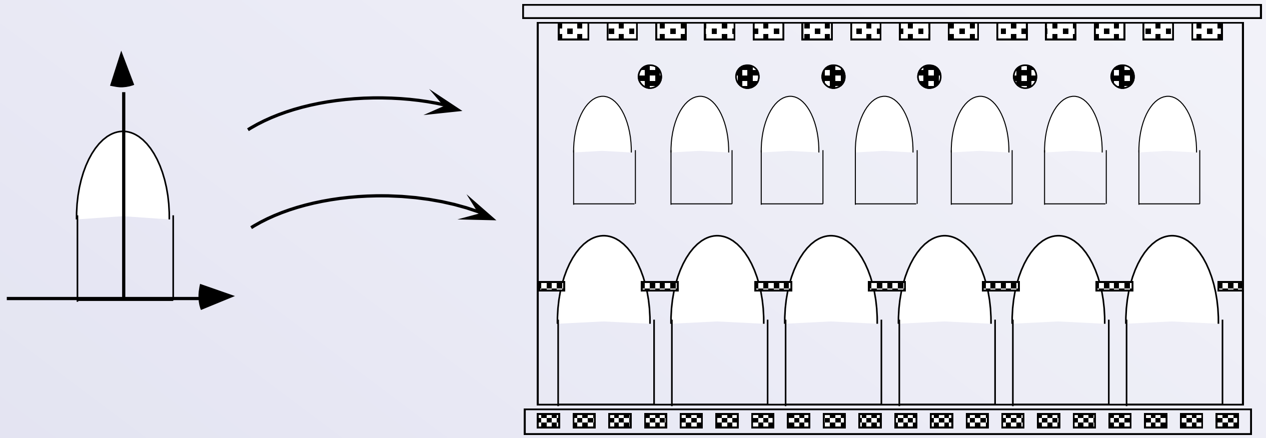
Exemplo de transformação afim

- A casa foi escalada, rotacionada e transladada, em 2D e 3D.



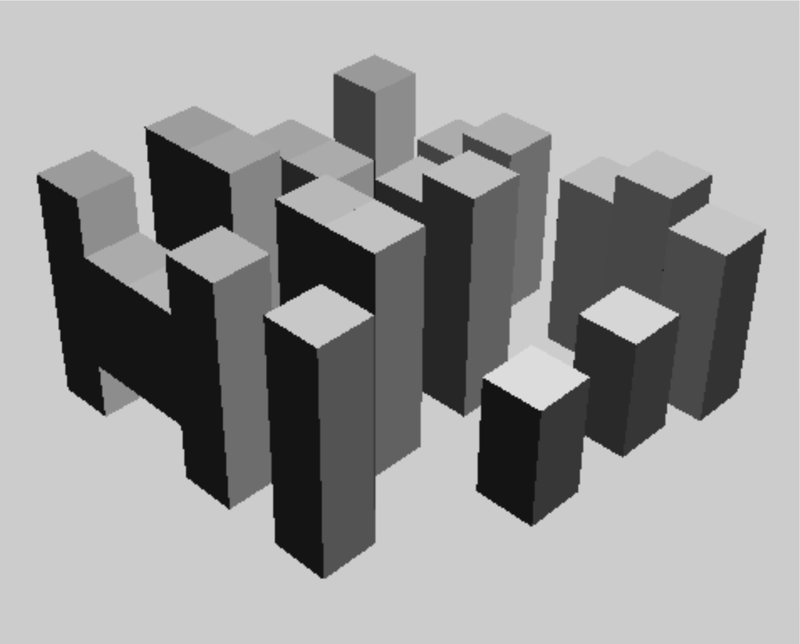
Usando transformações

- Cada arco foi desenhado no seu próprio sistema de coordenadas.
- A cena abaixo é criada através do desenho de várias instâncias do arco em posições e tamanhos diferentes.



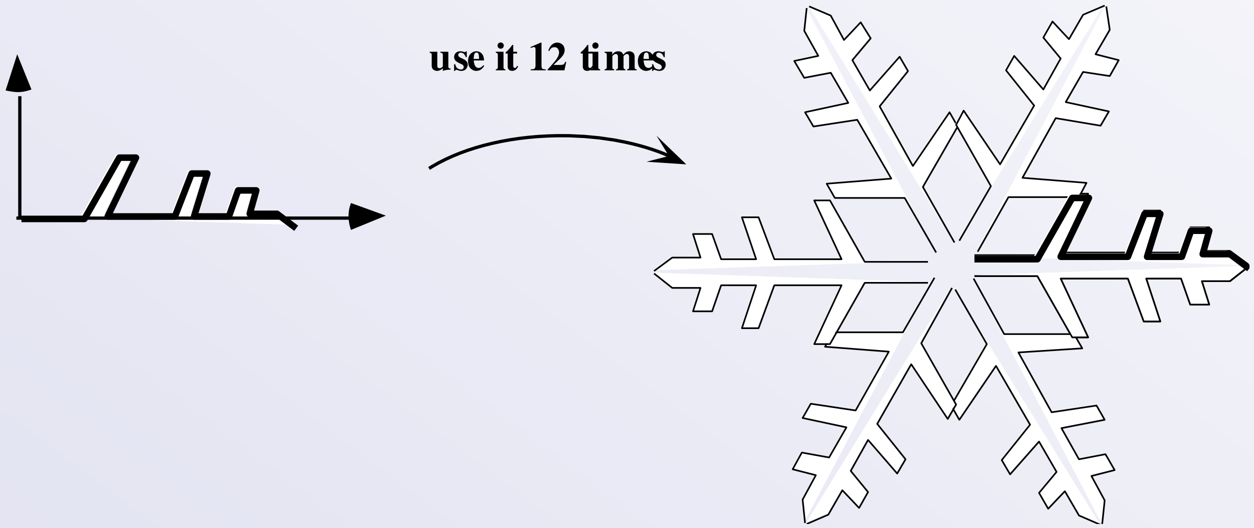
Usando transformações

- Em 3D, um conjunto de cubos pode se assemelhar à uma cidade;



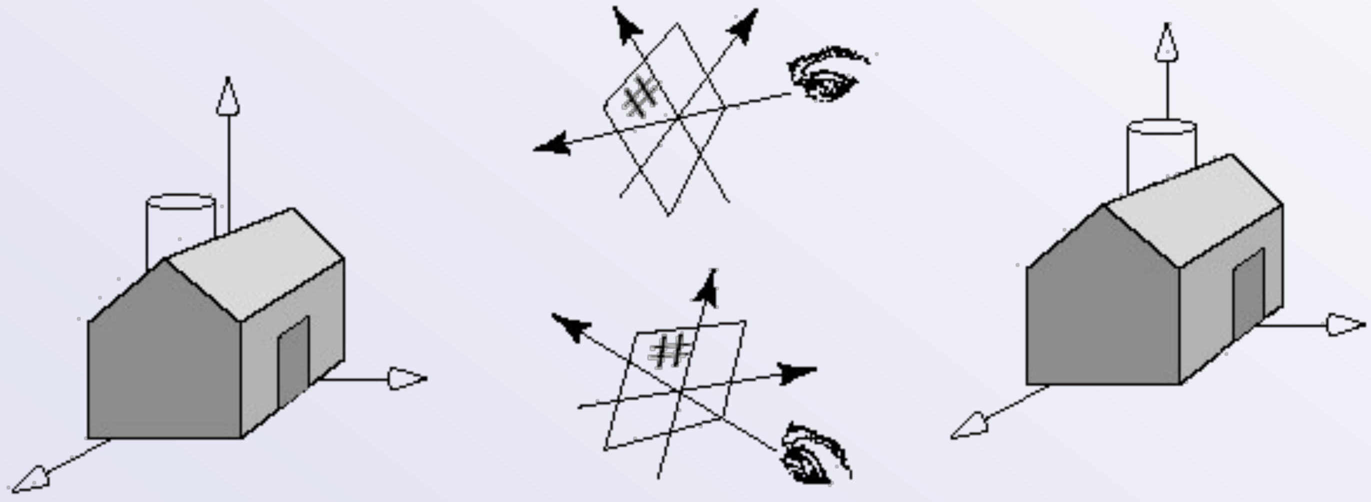
Usando transformações

- Um floco de neve é simétrico.
- Podemos iniciar com um simples perfil e desenhar toda a figura usando reflexões, rotações e translações deste perfil.



Usando transformações

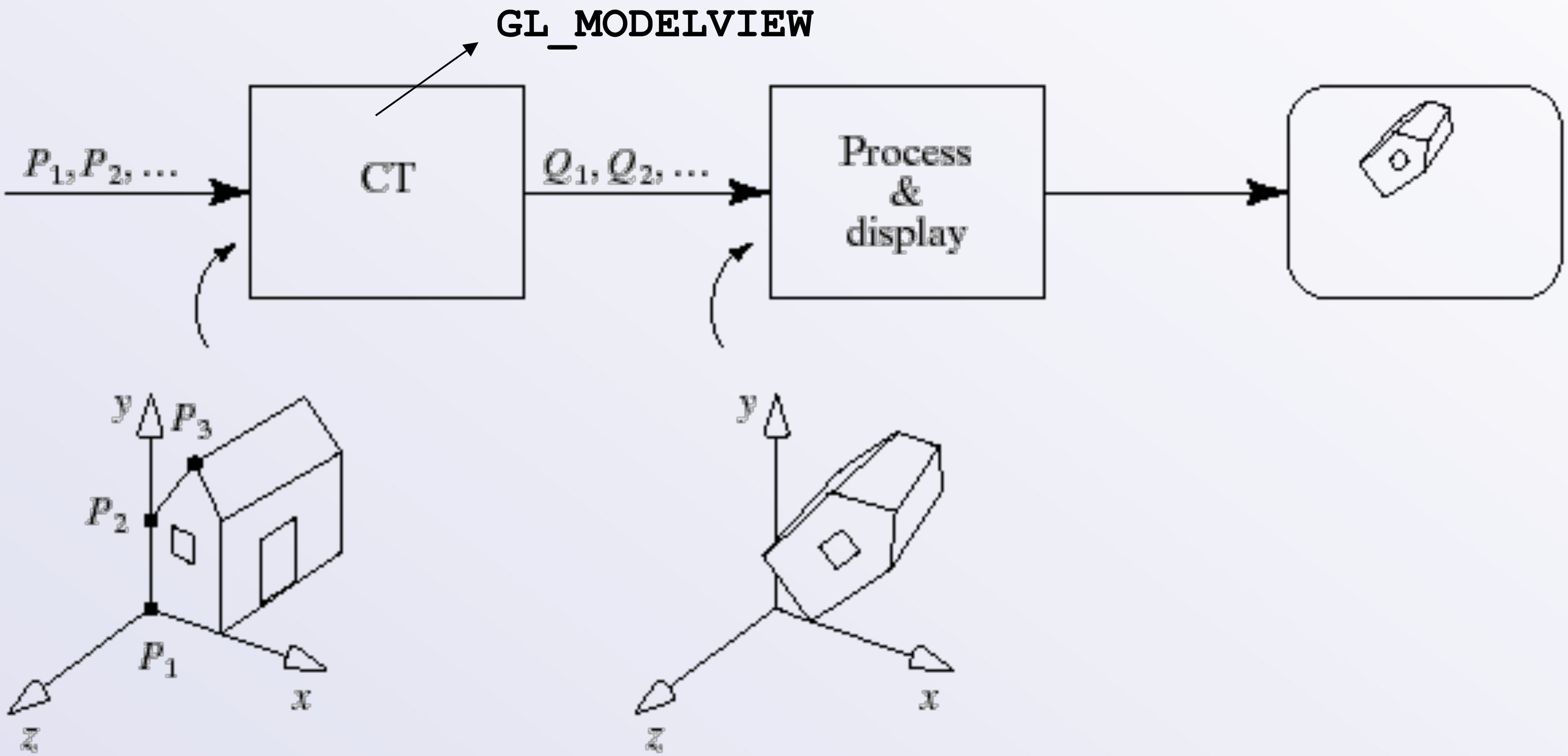
- Um desenhista pode desejar visualizar um objeto de diferentes perspectivas.
- O posicionamento e reorientação da câmera pode ser feito através de transformações afins em 3D.



Usando transformações

- Em uma animação, objetos em uma cena se movimentam
- Esta movimentação é criada através de translações e rotações des seus sistemas de coordenadas a medida que a animação prossegue
- Em OpenGL, podemos aplicar uma série de operações que serão aplicadas em todos os pontos de um objeto
- O objeto é desenhado depois das transformações de seus pontos

Transformações em OpenGL



Pipeline de transformação

- Uma aplicação manda para o pipeline uma sequência de pontos P_1, P_2, ... através de comandos como:

```
glBegin(GL_LINES);
    glVertex3f(...); // manda P1
    glVertex3f(...); // manda P2
    ...
glEnd();
```

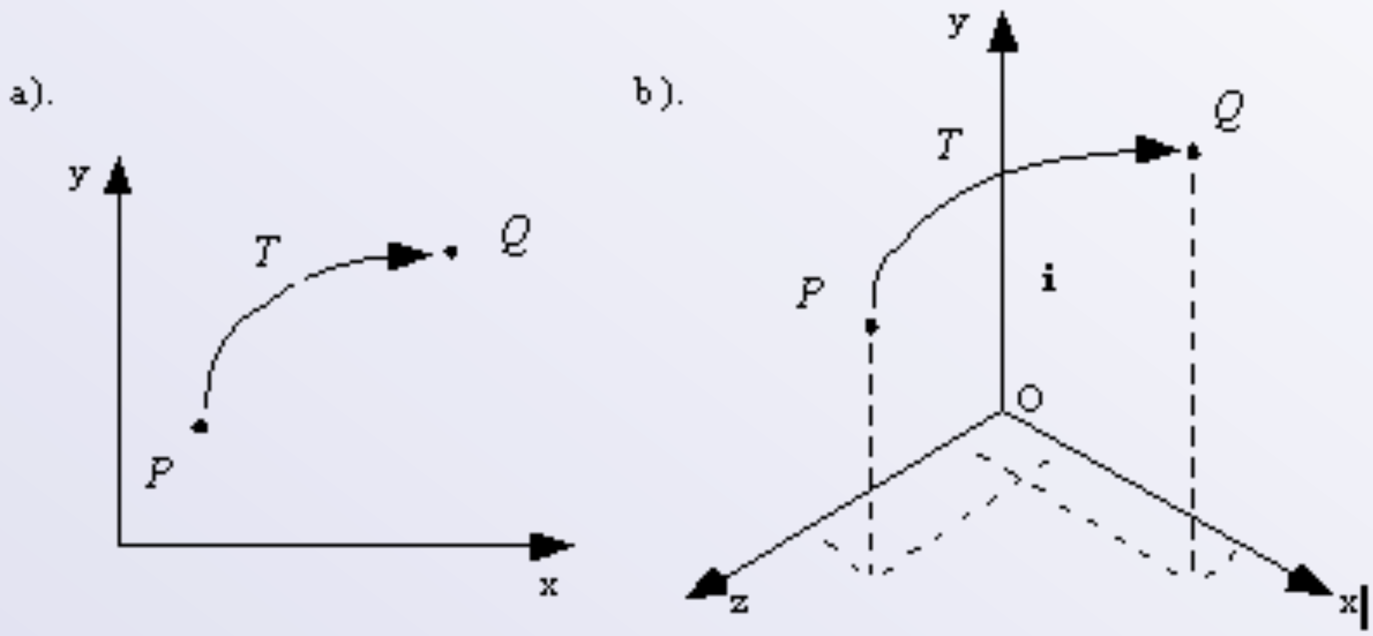
- Tais pontos encontram uma primeira transformação chamada de **transformação corrente** (TC), que altera seus valores para um diferente conjunto de pontos Q_1, Q_2, Q_3.

Transformações

- Transformações mudam pontos ou vetores em 2D e 3D, ou mudam sistemas de coordenadas.
  - A transformação de um objeto <u>altera as coordenadas de cada ponto</u> do objeto deixando o sistema local de coordenadas intacto.
  - Uma transformação entre sistemas de coordenadas define <u>um novo sistema de coordenadas</u> baseado no antigo, e representa todos os pontos do objeto neste novo sistema.

Transformações

- Uma transformação (2D ou 3D) *T*(.) altera cada ponto, *P* em um novo ponto, *Q*, usando uma fórmula: *Q*=*T*(*P*).



Transformações

- Porém, T poderá ter qualquer forma:

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(P_x)e^{-P_x} \\ \dfrac{\ln(P_y)}{1+P_x^2} \\ 1 \end{pmatrix}$$

- Nos restringiremos às transformações **afins**, ou seja, que sejam lineares em P_x e P_y.

Transformações Lineares em 2D

- Uma transformação linear

$$\begin{cases} x' = ax + by \\ y' = cx + dy \end{cases}$$

- Uma transformação linear afim

$$\begin{cases} x' = ax + by + e \\ y' = cx + dy + f \end{cases}$$

Forma Matricial

- Mais conveniente para uso em um computador. Sejam

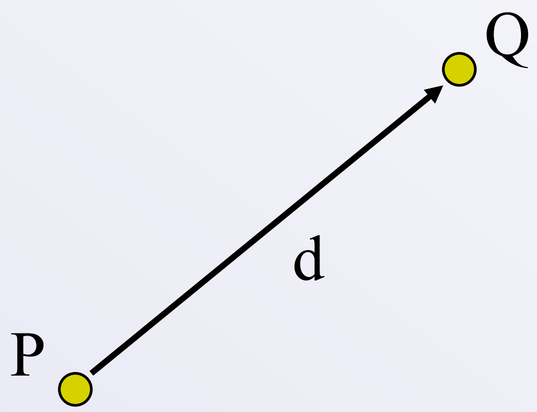
$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad D = \begin{bmatrix} e \\ f \end{bmatrix}$$

- Então uma transformação linear afim pode ser escrita $T(P) = P'$ onde

$$P' = A \times P + D$$

Translação

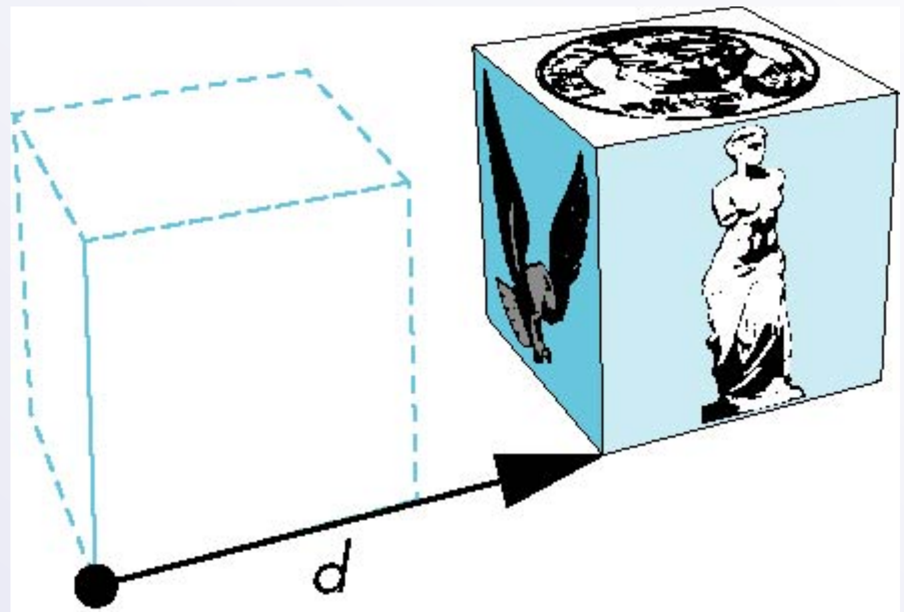
- Mover um ponto para uma nova posição



- Translação determinada pelo vetor d
  - Q=P+d

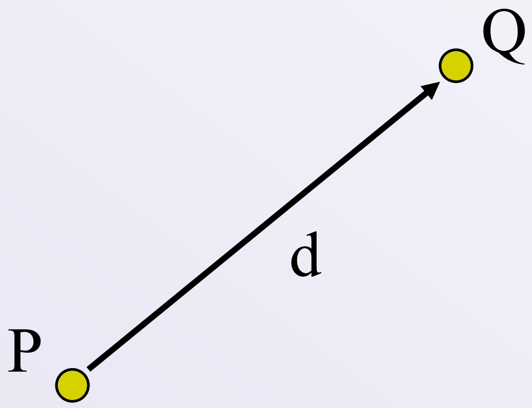
Movendo vários pontos

objeto



cada ponto é transladado
pelo mesmo vetor

Translação



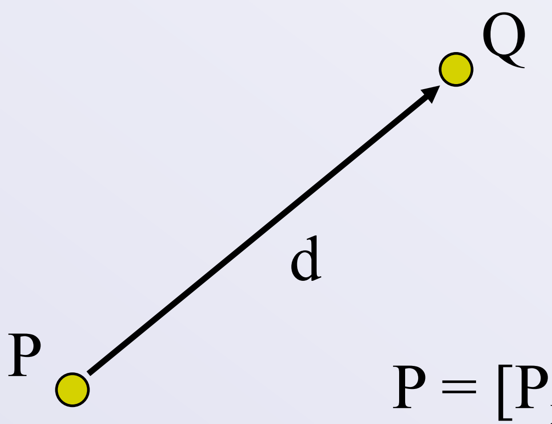
$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix}$$

- Podemos representar uma translação em termos de uma multiplicação entre matriz e vetor ?

$$\begin{bmatrix} Q_x \\ Q_y \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

Coordenadas Homogêneas

- Que tal se aumentássemos a dimensionalidade ?



$$\begin{bmatrix} Q_x \\ Q_y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ 1 \end{bmatrix}$$

$P = [P_x, P_y, \mathbf{1}]^T$

$Q = [Q_x, Q_y, \mathbf{1}]^T$

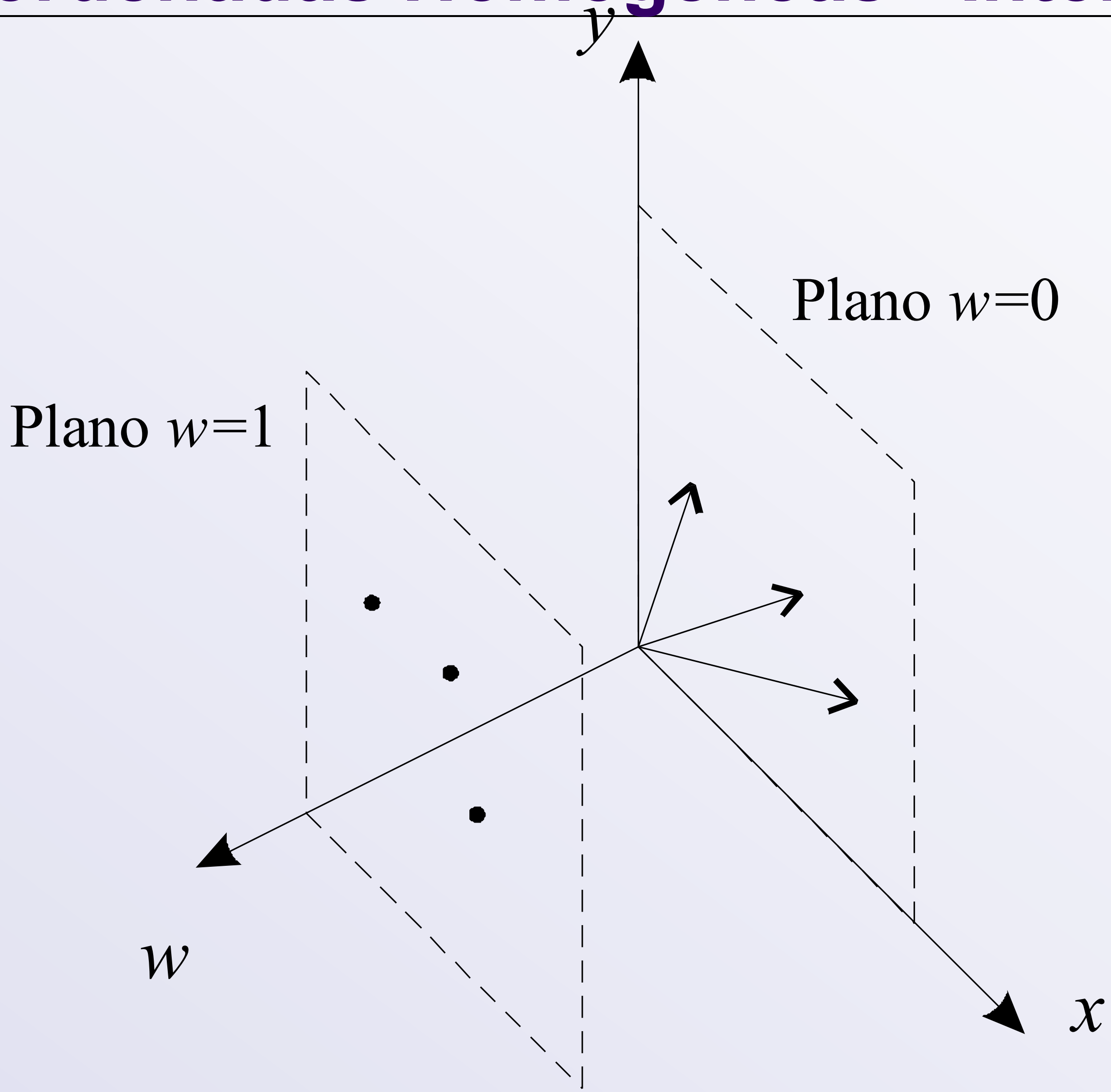
$d = [d_x, d_y, \mathbf{0}]^T$

Coordenadas Homogêneas

- A transformação de vetores é operacionalmente diferente da de pontos
- Coordenadas homogêneas permitem unificar o tratamento
- Problema é levado para uma dimensão superior:
  - Coordenada extra *w*=0 para vetores e w=1 p/ pontos
  - Termos independentes formam uma coluna extra na matriz de transformação

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Coordenadas Homogêneas - Interpretação



Transformações afins

- Matriz geral de transformação 2D:

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

- Para uma transformação afim em 2D, a terceira linha será sempre (0, 0, 1).

Transformando vetores

- Quando o vetor **V** é transformado pela mesma transformação afim que foi usada para o ponto P, o resultado será

$$\begin{pmatrix} W_x \\ W_y \\ 0 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} V_x \\ V_y \\ 0 \end{pmatrix}$$

- Importante: para transformar o ponto *P* em um ponto *Q*, *pós-multiplique P* por *M*: Q = M P.

Pré-multiplicar e pós-multiplicar

- Pré-multiplicar (pós-concatenar) M_2 com M_1:
  - **$p' = M_2 \,.\, M_1 \,.\, p$**
- Pós-multiplicar (pré-concatenar) M_2 com M_1:
  - **$p' = M_1 .\, M_2 \,.\, p$**

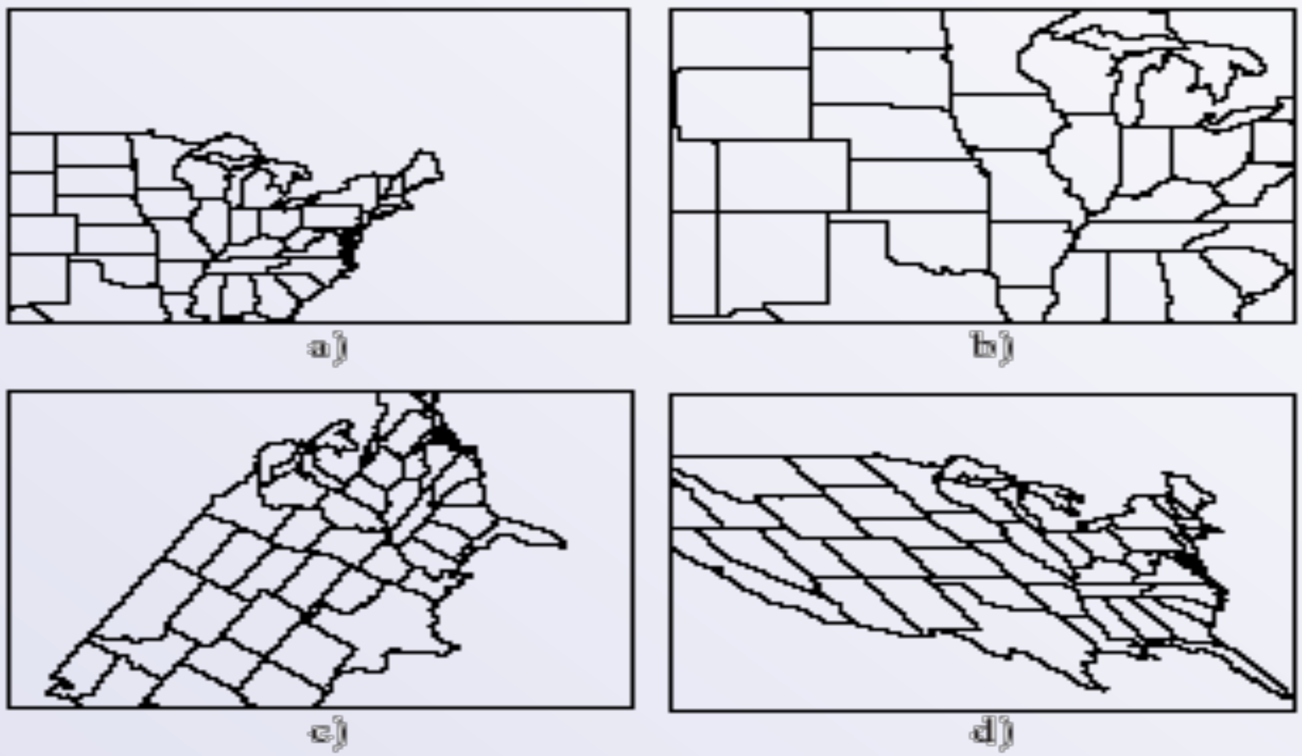
Exemplo de transformação

- Ache a imagem *Q* do ponto *P* = (1, 2, 1) usando a transformação afim *M*:

$$M = \begin{pmatrix} 3 & 0 & 5 \\ -2 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}; Q = \begin{pmatrix} 8 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 & 0 & 5 \\ -2 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$$

Efeitos geométricos das transformações afins

- Exemplos: (a) translação, (b) escala, (c) rotação, e (d) cisalhamento



Translações

- A distância que *P* é transladado não depende da posição de P
- Não faz sentido transladar vetores
- Matriz de translação:

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = \begin{pmatrix} Q_x + a \\ Q_y + b \\ 1 \end{pmatrix}$$

- Somente através do uso de coordenadas homogêneas, podemos definir a translação como uma transformação afim.

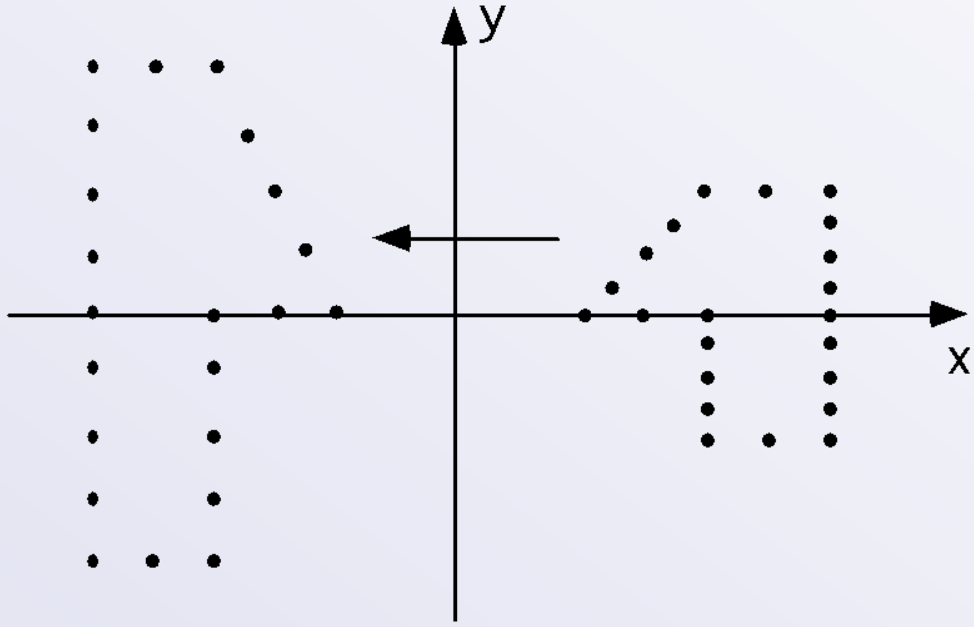
Escala

- A operação de escala é feita com base na origem. Se $S_x = S_y$ a escala é uniforme; caso contrário ela distorcerá a imagem.
- Se S_x ou $S_y < 0$, a imagem é refletida no eixo x ou y.
- A matriz de transformação de escala é:

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

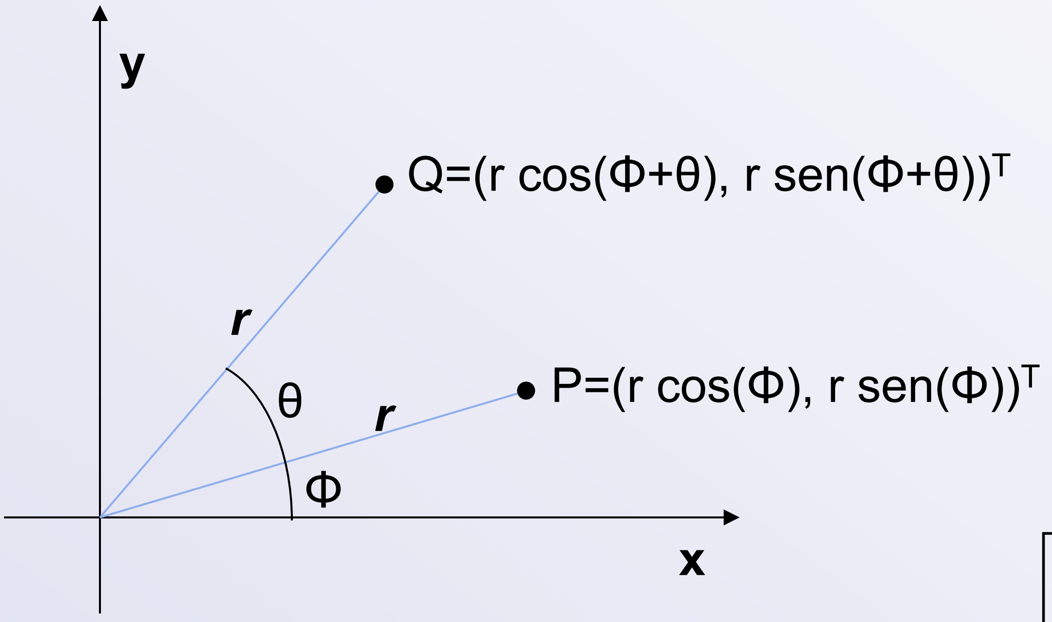
Exemplo de Escala

- A escala (Sx, Sy) = (-1, 2) é aplicada à um conjunto de pontos. Cada ponto é refletido pelo eixo x e escalado por 2 no eixo y.



Rotação

- Sob a origem, ângulo θ:



$$Q_x = P_x \cos\theta - P_y\,\mathrm{sen}\,\theta$$
$$Q_y = P_x\,\mathrm{sen}\,\theta + P_y \cos\theta$$

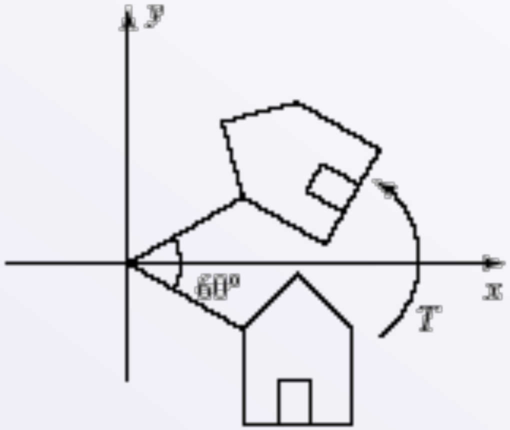
$\cos(\theta + \Phi) = \cos(\theta)\cos(\Phi) - \mathrm{sen}(\theta)\,\mathrm{sen}(\Phi);$

$\mathrm{sen}(\theta + \Phi) = \mathrm{sen}(\theta)\cos(\Phi) + \cos(\theta)\,\mathrm{sen}(\Phi).$

Matriz de rotação

- Sentido antihorário sob a origem com ângulo θ:

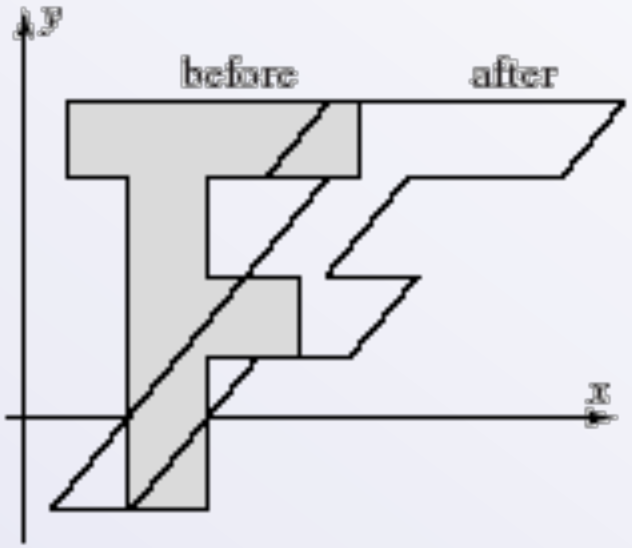
$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$



Cisalhamento

- Cisalhamento em x: h ≠ 0, e P_x depende de P_y.
- Cisalhamento em y: g ≠ 0, e P_y depende de P_x.

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & h & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$



Transformações inversas

- Inversa da translação (T^{-1}):

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

- Inversa da escala (S^{-1}):

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

Transformações inversas

- Inverso da rotação $R^{-1} = R(-\theta)$:

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & sen(\theta) & 0 \\ -sen(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

- Inverso do cisalhamento (H^{-1}):

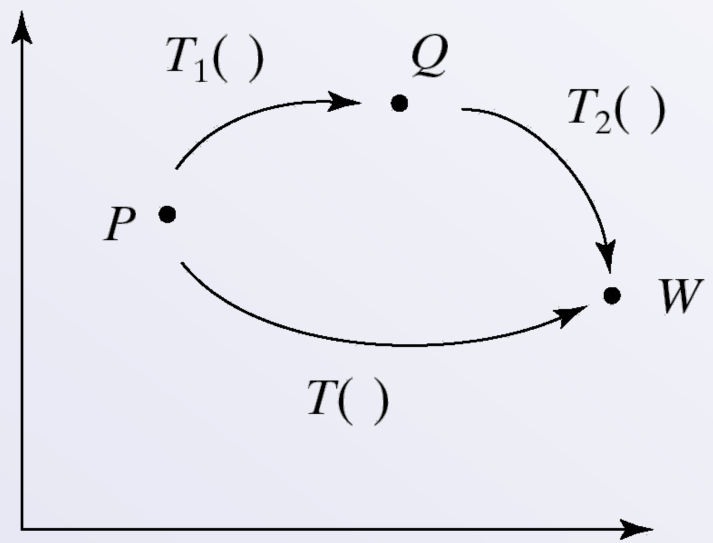
$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & -h & 0 \\ -g & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}$$

Composição de transformações

- Normalmente queremos aplicar uma série de transformações em uma determinada ordem:
  - transladar por (3, -4)
  - então rotacionar por 30^o
  - então escalar por (2, -1) e assim por diante.
- A aplicação de sucessivas transformações afins é chamada de **composição**.

Composição

- $T_1(.)$ mapeia P em Q, e $T_2(.)$ mapeia Q no ponto W.



Exemplos de composição

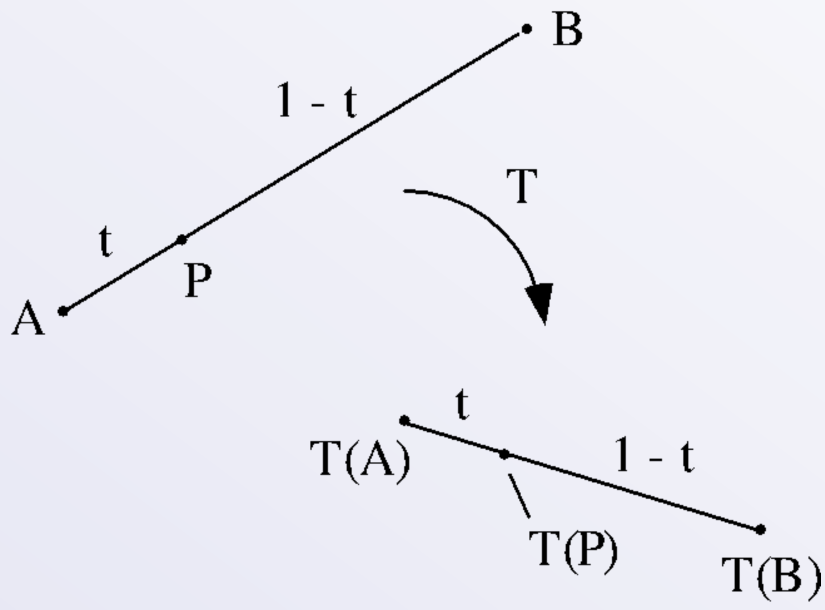
- Rotacionar um objeto sob um ponto arbitrário:
  - Transladar P para origem,
  - Rotacionar
  - Transladar P para a sua posição original
  - $Q = T_P \, R \, T_{-P} \, P$
- Escala sob um ponto arbitrário:
  - $Q = T_P S T_{-P} \, P$

Fatoração

- Cada transformação afim é composta por operações elementares.
- Uma matriz pode ser fatorada em um produto de matrizes elementares. Uma das maneiras:
  - M = [cisalhamento]x[escala]x[rotação]x[translação]
- M é uma matriz 3x3 que representa uma transformação 2D afim escrita como um produto de transformações elementares (direita para esquerda).

Propriedades

- Razões são preservadas
- $L(t)=(1-t)A + tB$
- $Q(t)=(1-t)T(A)+tT(B)$
- O ponto transformado *T*(*P*), está a mesma distância *t* entre as imagens *T*(*A*) e *T*(*B*).



Transformações em 3D

- Vetores e pontos em 3D

$$\vec{V} = \begin{bmatrix} V_x \\ V_y \\ V_z \\ 0 \end{bmatrix} \qquad P = \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix}$$

- Transformação linear afim

$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformações Rígidas

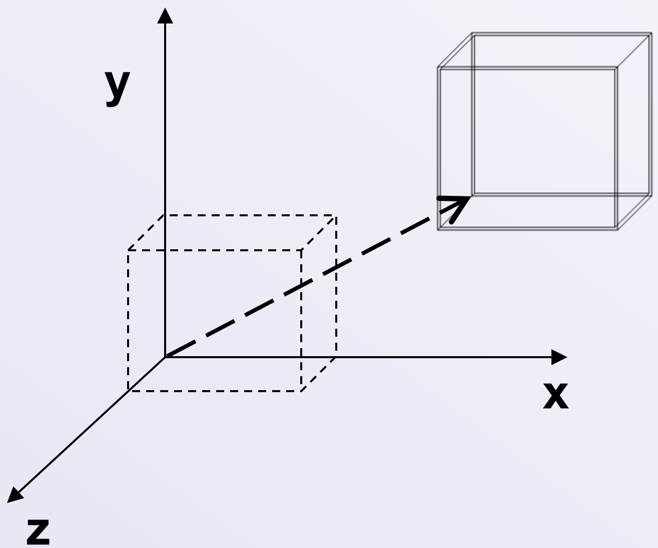
- Não modificam a forma (dimensensões /ângulos) do objeto
- São compostas de uma rotação e uma translação

Submatriz de Rotação

Vetor de Translação

$$T = \begin{bmatrix} a & b & c & j \\ d & e & f & k \\ g & h & i & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Translação



$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I_3 & t \\ 0 & 1 \end{bmatrix}$$

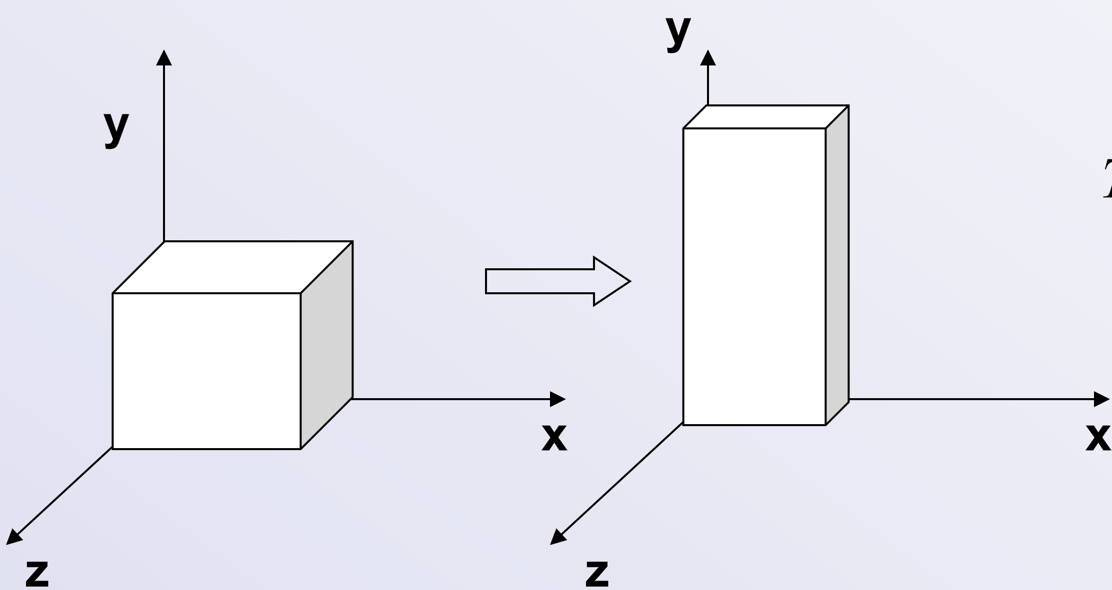
$$P' = T \times P = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} P_x + t_x \\ P_y + t_y \\ P_z + t_z \\ 1 \end{bmatrix} = P + t$$

- **Observe que translações são comutativas:**

  P + t + v = P + v + t

Escala

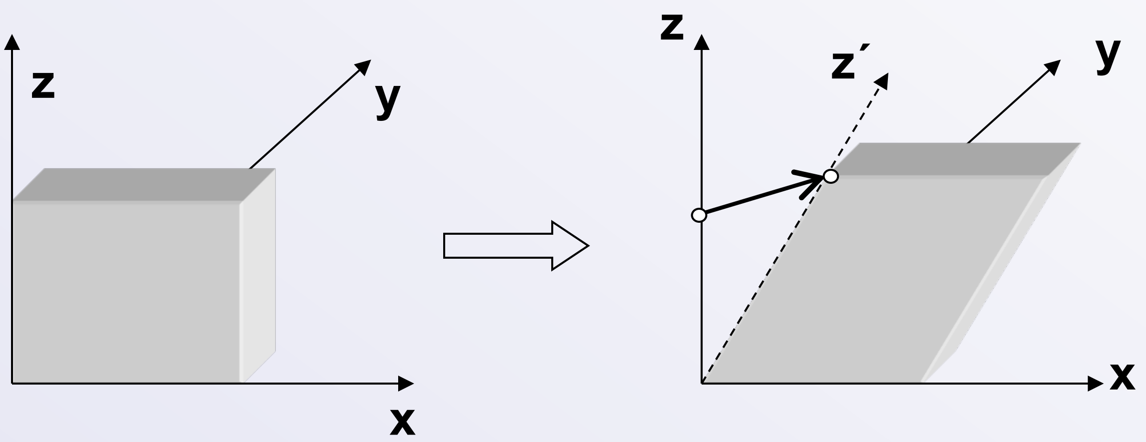
- Especificada por três fatores (S_x, S_y , S_z) que multiplicam os vetores unitários *x*, *y*, *z*
- Escala não é uma transformação rígida
- Escala uniforme ($S_x = S_y = S_z$) entretanto, é uma operação ortogonal ou homotética, isto é, preserva os ângulos
- Para obter reflexão em torno do plano *z*=0, usar fatores de escala (1, 1, -1)



$$T_{escala} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Cisalhamento ("*shear*")

- É uma transformação onde um eixo é "entortado" em relação aos demais



$$T_{inclinação} = \begin{bmatrix} 1 & 0 & Sh_x & 0 \\ 0 & 1 & Sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

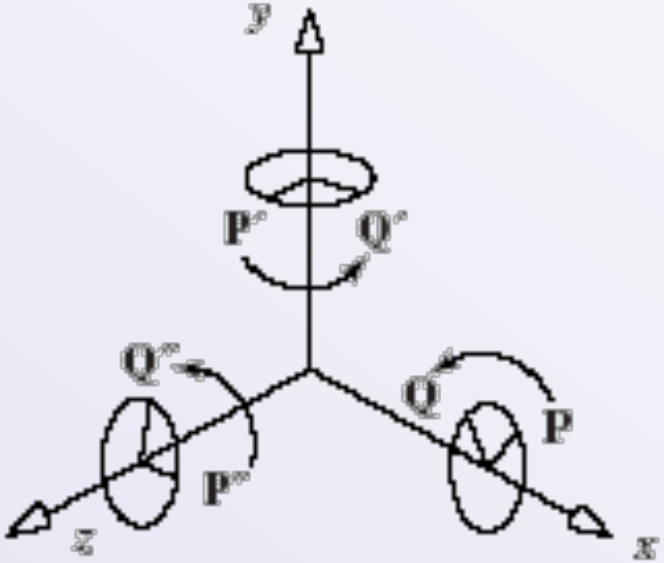
Cisalhamento

- Matriz genérica de cisalhamento:
  - a: x depende de y
  - b: x depende de z
  - c: y depende de x
  - d: y depende de z
  - e: z depende de x
  - f: z depende de y

$$H = \begin{pmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotações em 3D

- Sentido antihorário, sob um determinado eixo:



Rotações

- *Rotação em z*: eixo *x* rotaciona para o eixo *y*.
- *Rotação em x*: eixo *y* rotaciona para o eixo *z*.
- *Rotação em y*: eixo *z* rotaciona para o eixo *x*.

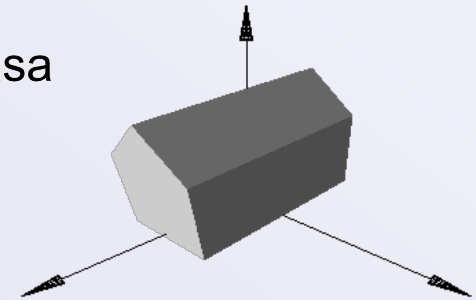
$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta & 0 \\ 0 & \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, R_y = \begin{pmatrix} \cos\vartheta & 0 & \sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_z = \begin{pmatrix} \cos\vartheta & -\sin\vartheta & 0 & 0 \\ \sin\vartheta & \cos\vartheta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

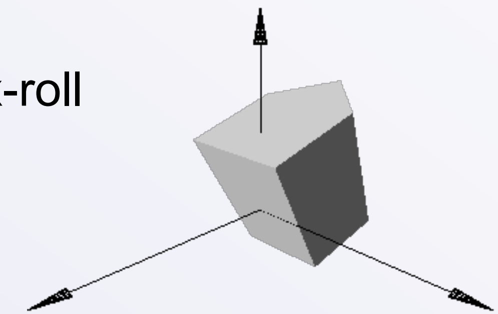
Exemplo

- Casa na sua orientação original, e depois de uma rotação de -70° em *x*, 30° em *y*, e -90° em *z*, separadamente.

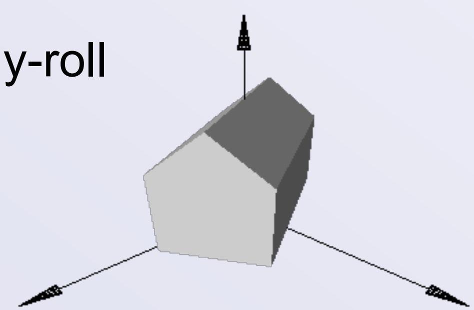
a) casa



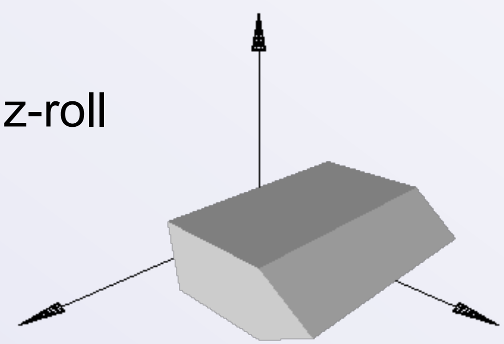
b) -70^0 x-roll



c) 30^0 y-roll



d) -90^0 z-roll

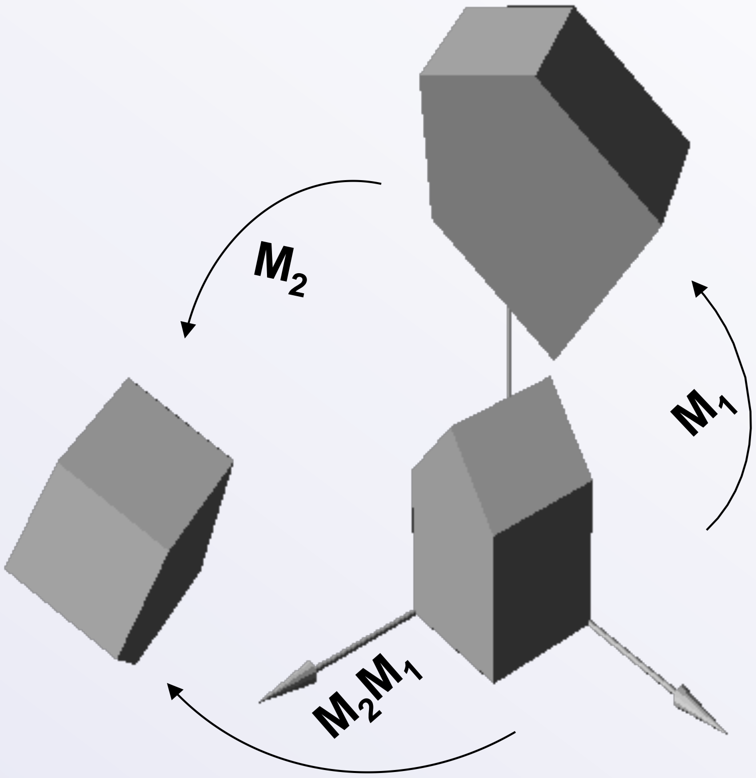


Composição

- Transformações afins em 3D também podem ser compostas, e o resultado é outra transformação 3D afim.
- A matriz final de transformação é o produto de matrizes individuais M_1 e M_2 que fazem suas transformações elementares, com M_2 *pré-multiplicando* M_1: $M = M_2M_1$
- Qualquer número de transformações afins podem ser compostas desta maneira, resultando em uma única matriz final de transformação.

Exemplo

- A casa é primeiramente transformada por M_1, e novamente transformada por M_2.
- O resultado é o mesmo se transformada uma única vez por M_2M_1.

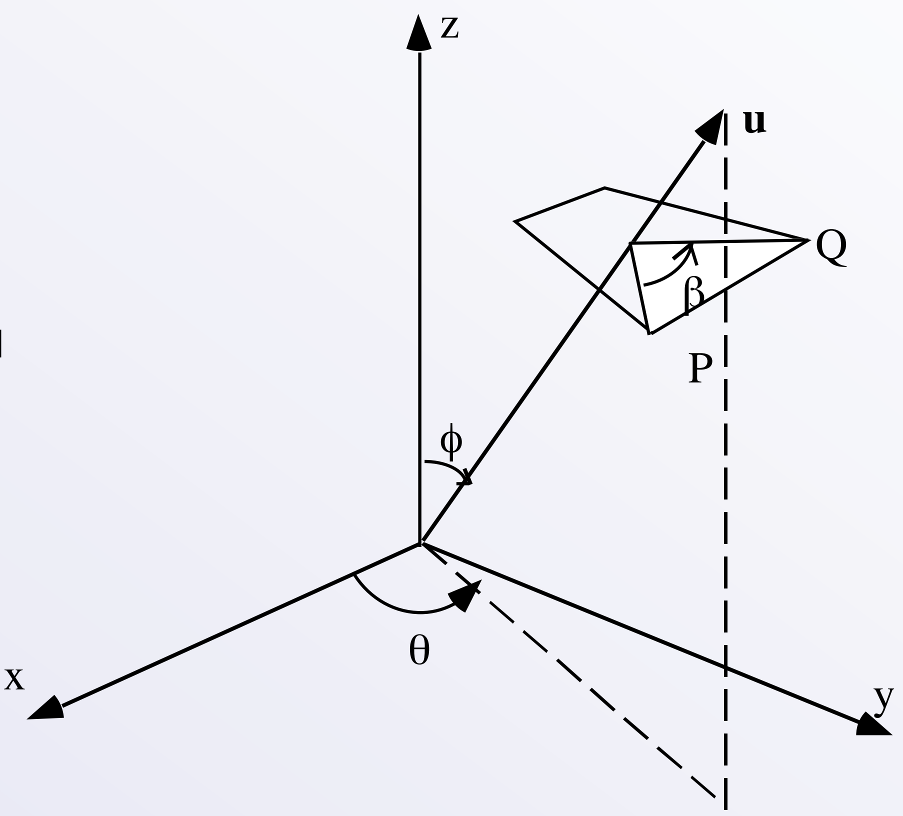


Rotações em 3D

- Em 2D, duas rotações combinam em uma rotação através da soma dos âgulos de rotação. E as matrizes comutam.
- Em 3D, as rotações são mais complicadas, pois podem ser feitas em diferentes eixos.
- A ordem em que duas rotações são realizadas **tem importância**: matrizes de rotação em 3D **não** comutam.
- Uma rotação em 3D pode ser composta através de 3 rotações elementares $M = R_z(\beta_3)R_y(\beta_2)R_x(\beta_1)$.
- Os ângulos β_1, β_2, and β_3 são chamados de **ângulos de Euler**

Rotação sob um eixo arbitrário

- Desejamos rotacionar **u** para transformar P em Q.
- Decompor a transformação em passos:
  - Duas rotações para alinhar **u** com o eixo z.
  - Rotacionar *β* sob o eixo z
  - Desfazer as duas primeiras rotações.



$$R_u(\beta) = R_z(-\theta)\ R_y(-\Phi)\ R_z(\beta)\ R_y(\Phi)\ R_z(\theta)$$

Sobre rotações

- ***Teorema de Euler: Qualquer rotação (ou sequência de rotações) sob um ponto é equivalente a uma única rotação ao redor de um eixo através daquele ponto.***
- *Qualquer* rotação em 3D ao redor de um eixo (que passa pela origem) pode ser obtido através do produto de 5 matrizes com ângulos de Euler apropriados.
- Isso implica que somente três valores são necessários para especificar qualquer rotação!

Sumarizando

- Transformações preservam linhas e planos
- Razões relativas são preservadas
- O paralelismo de linhas e planos também é preservada
- As colunas da matriz revelam o sistema de coordenadas
- Cada transformação afim é composta por operações elementares.

Ordem das transformações

- A matriz mais à direita é a primeira a ser aplicada
- Matematicamente, as expressões abaixo são equivalentes

$$\mathbf{p}' = \mathbf{ABCp} = \mathbf{A(B(Cp))}$$

- Muitas referências usam vetores colunas para representar pontos. Neste caso:

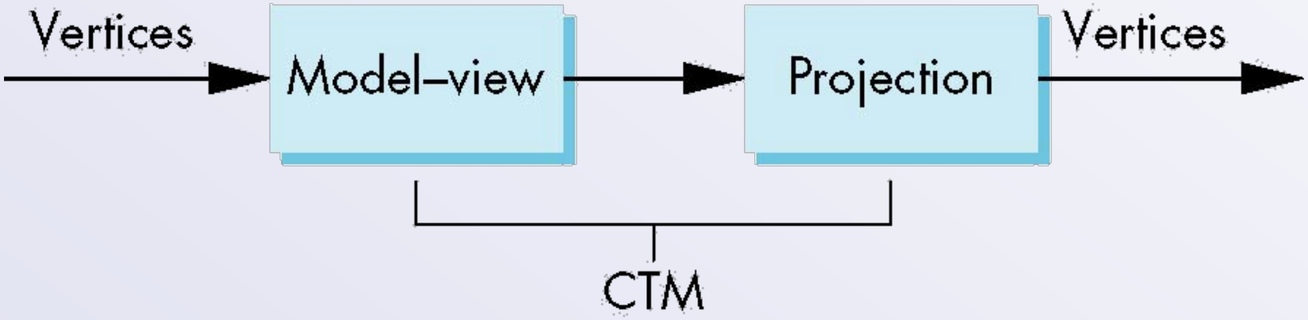
$$\mathbf{p}'^{T} = \mathbf{p}^{T}\mathbf{C}^{T}\mathbf{B}^{T}\mathbf{A}^{T}$$

Matrizes em OpenGL

- Matrizes são parte do estado
- Tipos:
  - Matriz do modelo (**`GL_MODELVIEW`**)
  - Matriz de projeção (**`GL_PROJECTION`**)
  - Matriz de textura (**`GL_TEXTURE`**)
- Dividem o mesmo conjunto de operações
- Seleção:
  - **`glMatrixMode(GL_MODELVIEW);`**
  - **`glMatrixMode(GL_PROJECTION);`**

Matriz corrente (MCT)

- A matriz de modelo e projeção são concatenadas para formar a **matriz corrente de transformação (MCT)**.
- Cada uma pode ser manipulada através da correta seleção.



Rotação, Translação e Escala

Carregando a matriz identidade:

```
glLoadIdentity()
```

Multiplicando à direita:

```
glRotatef(theta, vx, vy, vz)
```

theta em graus, (**vx, vy, vz**) eixo de rotação

```
glTranslatef(dx, dy, dz)
glScalef(sx, sy, sz)
```

Exemplo

- Rotação ao redor do eixo z por 30 graus sob um ponto fixo (1.0, 2.0, 3.0) e não sob a origem:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);
```

- A **última** matriz especificada é a **primeira** a ser aplicada!

Matrizes arbitrárias

- Podemos carregar e multiplicar por matrizes definidas em nossa aplicação

```
glLoadMatrixf(M)
glMultMatrixf(M)
```

- A matrix **M** tem uma dimensão, contendo 16 elementos, que são os componentes da matrix 4x4 organizada em <u>colunas</u>.
- **`glMultMatrixf(M)`** pós-multiplica M pela matriz corrente
  - i.e. C = CxM

Pilhas de matrizes

- Em algumas situações gostaríamos de salvar as transformações para uso posterior
- OpenGL mantém pilhas para cada tipo de matriz (determinado por **`glMatrixMode`**):

```
glPushMatrix()
glPopMatrix()
```

Utilizando transformações

- Exemplo: uso da função idle para rotacionar um cubo e uso da função do mouse para mudar de direção de rotação.
- Desenhar um cubo (`colorcube.c`) de forma padrão
  - Centrado na origem
  - Seus lados coincidem com os eixos x, y e z

main.c

```
void main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB |
       GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(display);
    glutIdleFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

Callbacks Idle e Mouse

```
void spinCube()
{
   theta[axis] += 2.0;
   if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
   glutPostRedisplay();
}
  void mouse(int btn, int state, int x, int y)
  {
     if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
             axis = 0;
     if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
             axis = 1;
     if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
             axis = 2;
  }
```

Callback de display

```
void display()
{
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glLoadIdentity();
   glRotatef(theta[0], 1.0, 0.0, 0.0);
   glRotatef(theta[1], 0.0, 1.0, 0.0);
   glRotatef(theta[2], 0.0, 0.0, 1.0);
   colorcube();
   glutSwapBuffers();
}
```

Tarefa #4

- Como a área/volume de uma figura é afetada por uma transformação linear afim ?
  - Translações e rotações ?
  - Escalas e cisalhamentos ?
- Respostas no fórum da disciplina.