Computação Gráfica

Transformações Geométricas

Vectores

- Magnitude

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

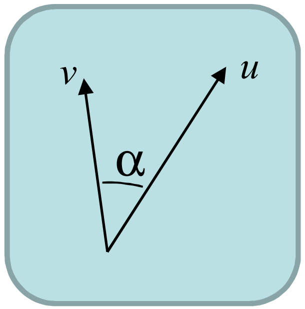
- Vector Normalizado (magnitude = 1)

$$v_{norm} = \frac{v}{|v|}$$

- Produto Interno

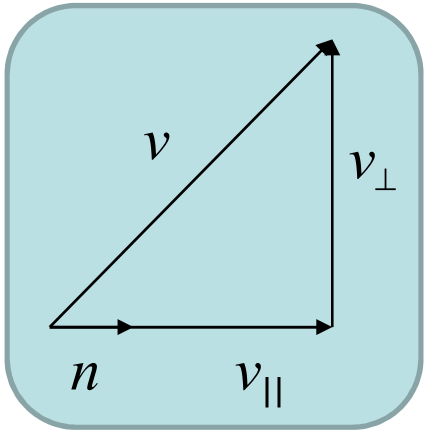
$$v \cdot u = \sum_{i=1}^{3} v_i * u_i$$

$$v \cdot u = \|v\| \times \|u\| \times \cos(\alpha)$$



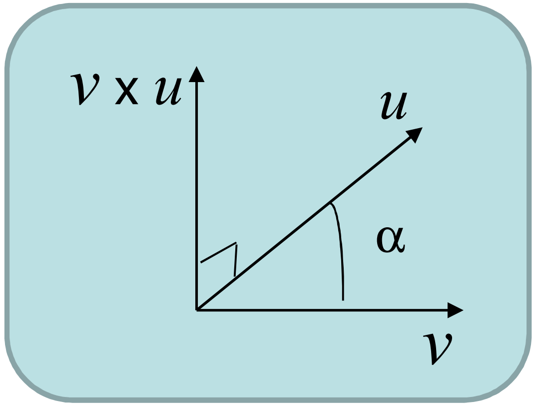
Vectores

- Projecção



$$v_{||} = n\frac{v \cdot n}{|n|^2}$$

- Produto Externo



$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \times \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} v_y u_z - v_z u_y \\ v_x u_z - v_z u_x \\ v_x u_y - v_y u_x \end{bmatrix}$$

$$|v \times u| = |v| \times |u| \times \sin(\alpha)$$

Sistemas de Coordenadas

- Um triplo de vectores *(u,v,w)* pode definir um sistema de coordenadas 2D desde que sejam linearmente independentes.

- Um conjunto de 2 vectores é linearmente independente se nenhum dos vectores se puder escrever como uma combinação linear dos restantes,

- ou seja, não existe nenhuma combinação de números a_1, a_2, a_3,sendo pelo menos um deles diferente de zero, tal que

$$a_1 v + a_2 u + a_3 w = 0$$

Sistemas de Coordenadas

- Uma matriz invertível pode ser vista como uma transformação entre sistemas de coordenadas.

- Uma matriz invertível implica que os seus vectores (linha ou coluna) sejam linearmente independentes.

- Os vectores de uma matriz invertível representam um sistema de eixos, ou seja, um sistema de coordenadas.

Sistemas de Coordenadas

- Um conjunto de vectores $(v_1,\ldots v_n)$ forma uma base <u>ortogonal</u> se

$$\boxed{\forall (i,j), i \neq j, v_i \cdot v_j = 0}$$

- Um conjunto de vectores (v1,...,vn) forma uma base <u>ortonormal</u> se

$$\boxed{\begin{array}{l} \forall (i,j), v_i \cdot v_j = \delta_{ij} \\ \delta_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j \end{cases} \end{array}}$$

Sistemas de Coordenadas

- Uma matriz cujos vectores coluna formem uma *base ortonormal* é uma *matriz ortogonal*
- Se G é ortogonal então

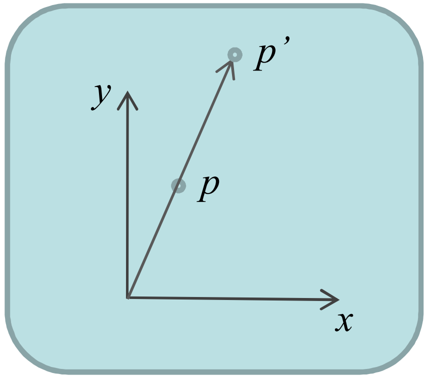
$$G^{-1} = G^T$$

- Um caso particular são as rotações!

Escala

- Escala Uniforme
  - Seja p um ponto e k um escalar,

    $p' = kp$



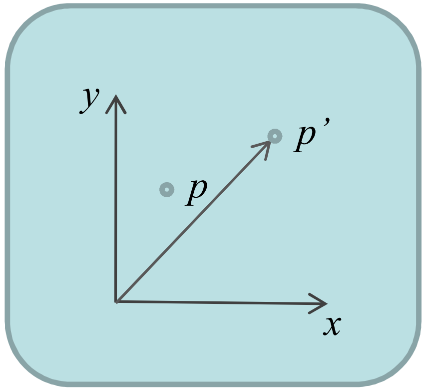
equações $x' = kx$

$y' = ky$

forma matricial $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

- Escala Não Uniforme
  - Seja p um ponto e k_1, k_2 um par de escalares,

    $p' = Kp$



equações $x' = k_1 x$

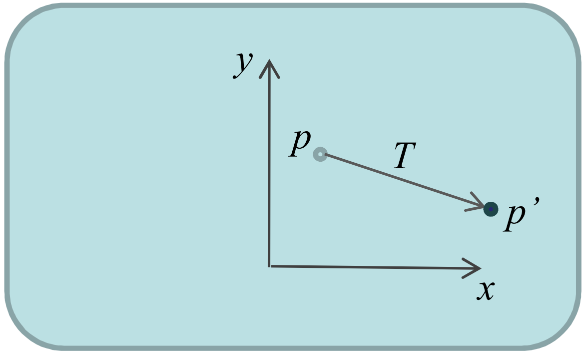
$y' = k_2 y$

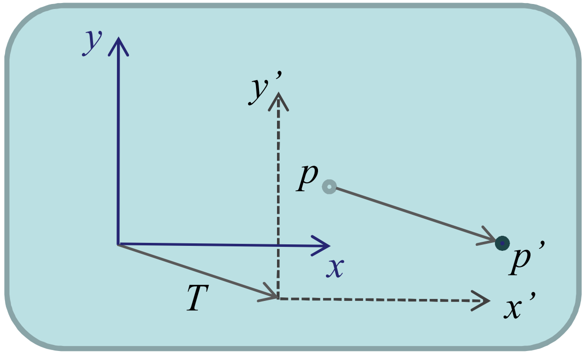
forma matricial $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} k_1 & 0 \\ 0 & k_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$

Translação

- Seja p um ponto e T um vector,

$$p' = p + T$$

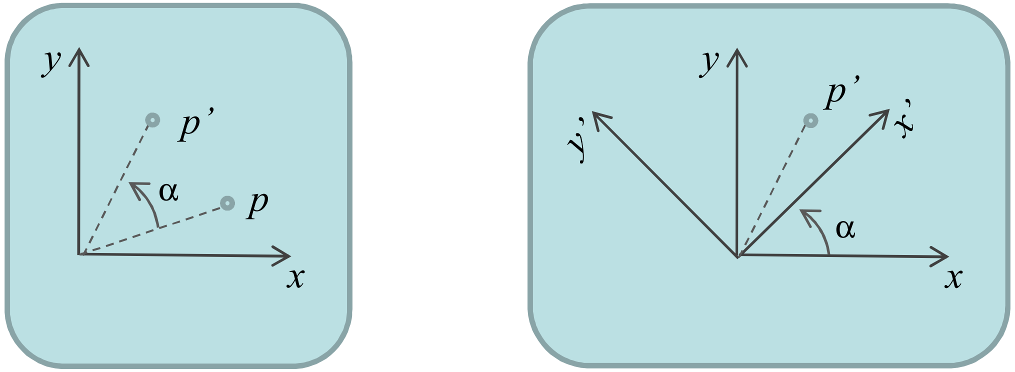




Translação do ponto num sistema fixo
⇔
Translação do sistema de coordenadas

Rotação

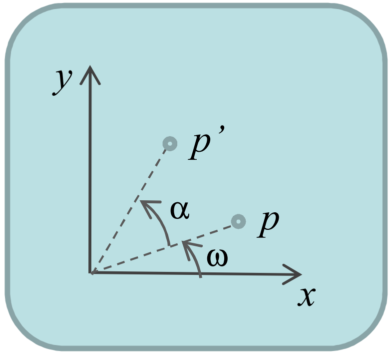
- Rotação em torno da origem



Rotação do ponto num sistema fixo
⇔
Rotação do sistema de coordenadas

Rotação

- Seja $p = (a,b)$ e $p' = (a', b')$



equações

$$a' = a\cos(\alpha) - b\sin(\alpha)$$
$$b' = a\sin(\alpha) + b\cos(\alpha)$$

forma matricial

$$\begin{bmatrix} a' \\ b' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

Trigonometria

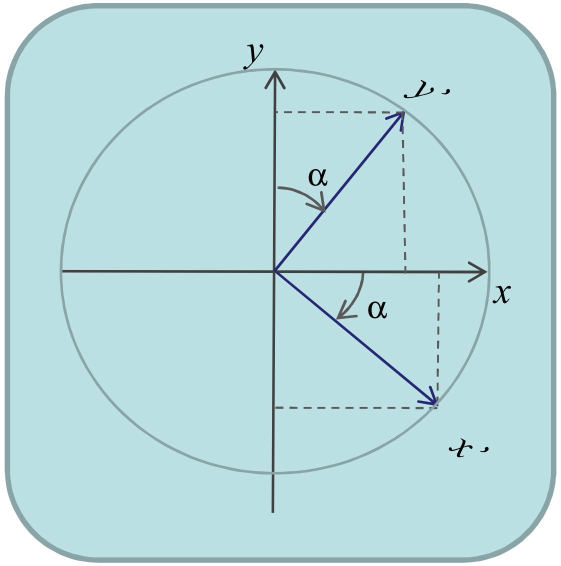
$$a' = |p'|\cos(\omega + \alpha)$$
$$b' = |p'|\sin(\omega + \alpha)$$

$$\cos(\omega) = \frac{a}{|p|}$$
$$\sin(\omega) = \frac{b}{|p|}$$

$$\cos(\omega + \alpha) = \cos(\omega)\cos(\alpha) - \sin(\omega)\sin(\alpha)$$
$$\sin(\omega + \alpha) = \sin(\omega)\cos(\alpha) + \cos(\omega)\sin(\alpha)$$

Rotação

- Vejamos qual o resultado de escrevermos o novo sistema de coordenadas em função dos eixos do sistema original



$$x' = (\cos(\alpha), \sin(\alpha))$$

$$y' = (-\sin(\alpha), \cos(\alpha))$$

A definição dos eixos do novo sistema *(x',y')* corresponde às colunas da matriz R

$$R = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

Rotação

- Rotação Inversa

- Se

$$R_\alpha = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$$

- Então

$$R_{-\alpha} = \begin{bmatrix} \cos(-\alpha) & -\sin(-\alpha) \\ \sin(-\alpha) & \cos(-\alpha) \end{bmatrix}$$

$$R_{-\alpha} R_\alpha = I$$

Rotação

- Sabemos que

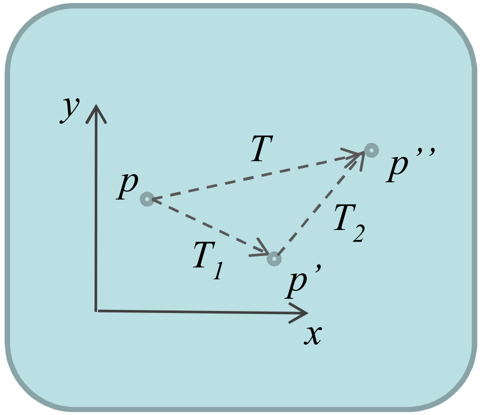
$$\cos(-\alpha) = \cos(\alpha)$$
$$\sin(-\alpha) = -\sin(\alpha)$$

- Logo

$$R_{-\alpha} = \begin{bmatrix} \cos(-\alpha) & -\sin(-\alpha) \\ \sin(-\alpha) & \cos(-\alpha) \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix}$$
$$R_{-\alpha} = R_{\alpha}^{T}$$

Composição de Transformações

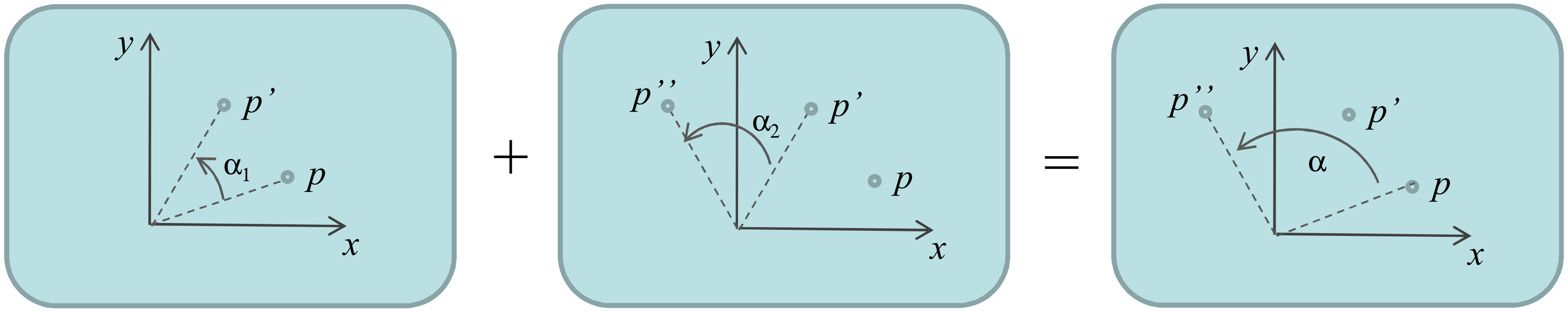
- Translação + Translação



$$p' = p + T_1$$
$$p'' = p' + T_2$$
$$p'' = p + T_1 + T_2 = p + T$$

Composição de Transformações

- Rotação + Rotação



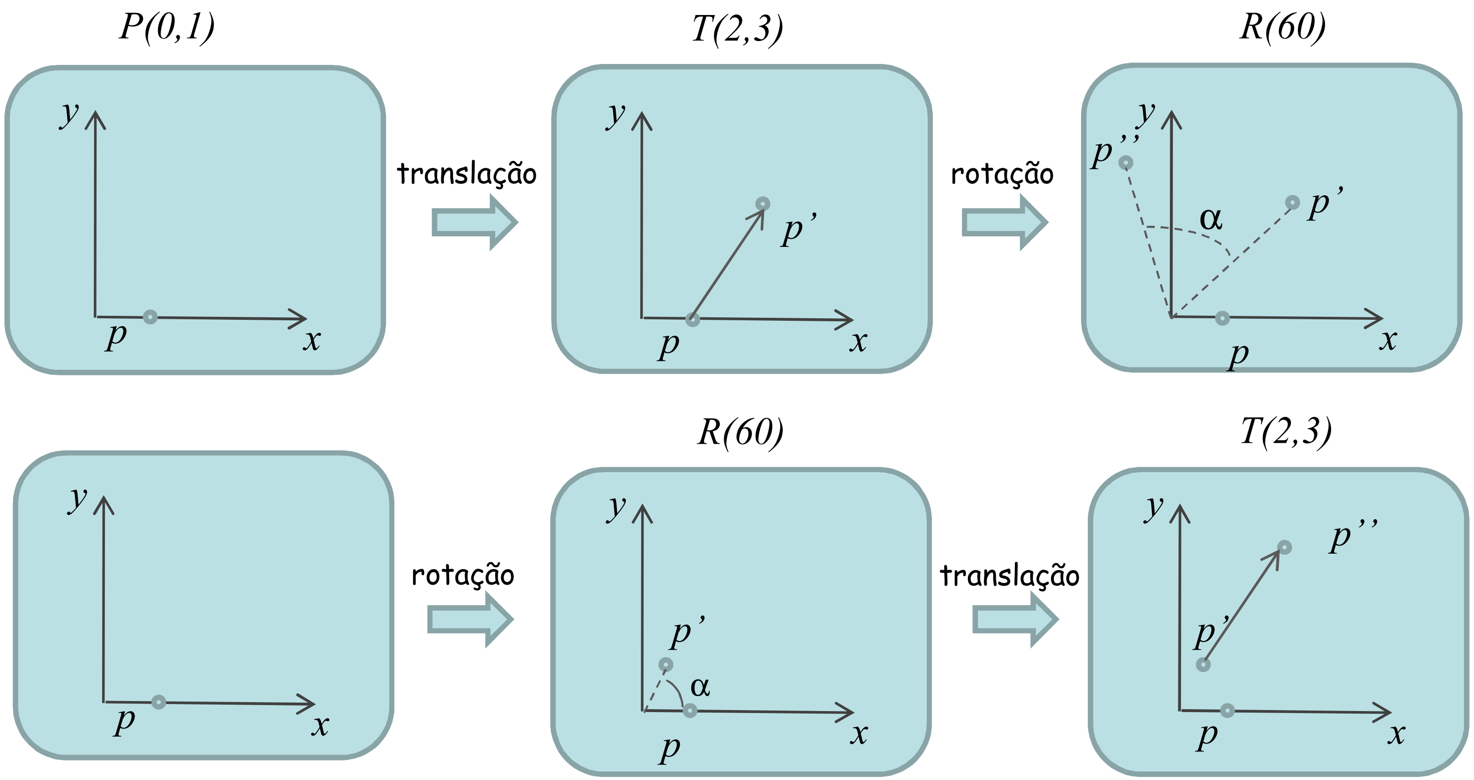
$$p' = R_{\alpha_1} \cdot p$$

$$p'' = R_{\alpha_2} \cdot p'$$

$$p'' = R_{\alpha_2} \cdot R_{\alpha_1} \cdot p = R_{\alpha} \cdot p$$

Composição de Transformações

- Rotação e Translação
  - A ordem das transformações é relevante!

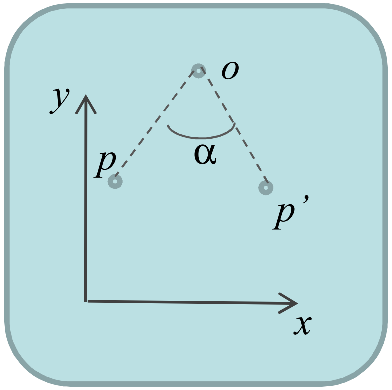


$$p'' = R_{-\alpha}(p+T)$$
$$p'' = (-1.1, 4.1)$$

$$p'' = T + (R_{-\alpha}p)$$
$$p'' = (2.5, 3.86)$$

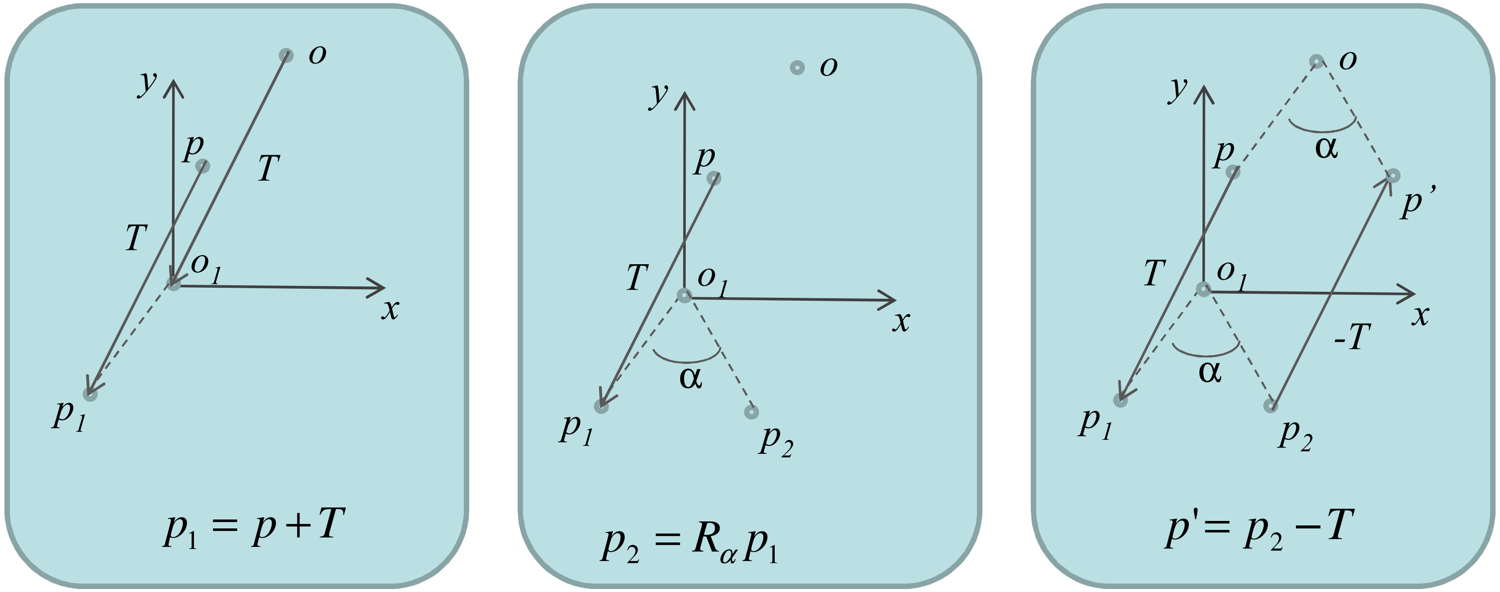
Composição de Transformações

- Rotação de p em torno de um ponto arbitrário o



Composição de Transformações

- Rotação de p em torno de um ponto arbitrário o



$$p' = R_{\alpha}(p + T) - T$$

Composição de Transformações

- E se quisermos em seguida rodar o resultado p' em torno de outro ponto arbitrário?

- Sabemos que:

$$\boxed{\begin{aligned} p' &= R_{\alpha_1}(p + T_1) - T_1 \\ p'' &= R_{\alpha_2}(p' + T_2) - T_2 \end{aligned}}$$

- Logo:

$$\boxed{p'' = R_{\alpha_2}((R_{\alpha_1}(p + T_1) - T_1) + T_2) - T_2}$$

Coordenadas Homogéneas

- Pretende-se uniformizar a forma das transformações geométricas

- Para tal recorre-se a coordenadas homogéneas.

- Em 2D, um ponto $P(X,Y)$ em coordenadas cartesianas representa-se por $p(x,y,w)$ em coordenadas homogéneas,

Sendo $$X = \frac{x}{w}, Y = \frac{y}{w}$$

- Por omissão considera-se $w = 1$

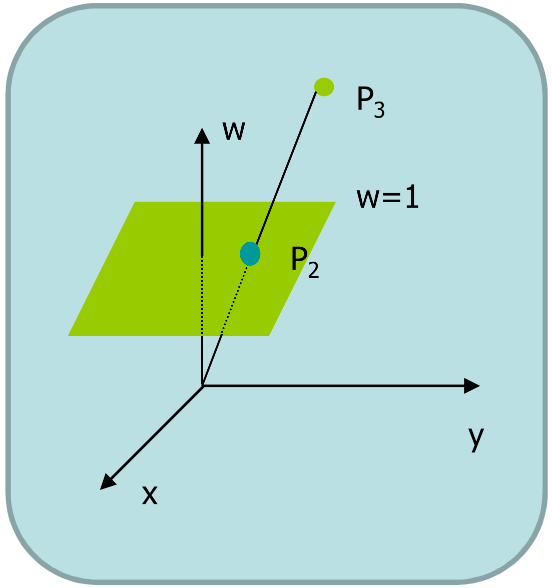
Coordenadas Homogéneas

- Pontos com coordenadas homógeneas distintas representam o mesmo ponto 2D
- O ponto em coordenadas cartesianas é obtido dividindo as duas primeiras coordenadas pela última coordenada.
- Para vectores w = 0, porquê? (tip: diferença de pontos)

$$P_3 = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

$$P_2 = \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

$$\begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \approx \begin{bmatrix} 4 \\ 6 \\ 2 \end{bmatrix}$$



Coordenadas Homogéneas

- Revisitar Transformações Geométricas

  - Pontos têm mais uma coordenada, logo rotações e escalas em 2D são representadas por matrizes 3x3. Seja $p = (x,y,1)$

  - Escalas

$$p' = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = (k_1 x, k_2 y, 1)$$

escala $(k1,k2)$

$$p = \begin{bmatrix} \frac{1}{k_1} & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} k_1 x \\ k_2 y \\ 1 \end{bmatrix} = (x, y, 1)$$

escala inversa

Coordenadas Homogéneas

- Revisitar Transformações Geométricas

  - Rotações

$$p' = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} R_\alpha & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_2 \\ 1 \end{bmatrix} = (R_\alpha p_2, 1)$$

rotação α

$$p = \begin{bmatrix} R_{-\alpha} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_\alpha p \\ 1 \end{bmatrix} = (R_{-\alpha} R_\alpha p_2, 1) = (p_2, 1)$$

rotação inversa

Coordenadas Homogéneas

- Revisitar Transformações Geométricas

  - Translações 2D também podem ser representadas por uma matriz 3x3

$$p = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} I_2 & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ 1 \end{bmatrix} = (p + T, 1)$$

Com coordenadas homogéneas <u>todas</u> as transformações geométricas em 2D podem ser representadas por matrizes 3x3

Composição de Transformações

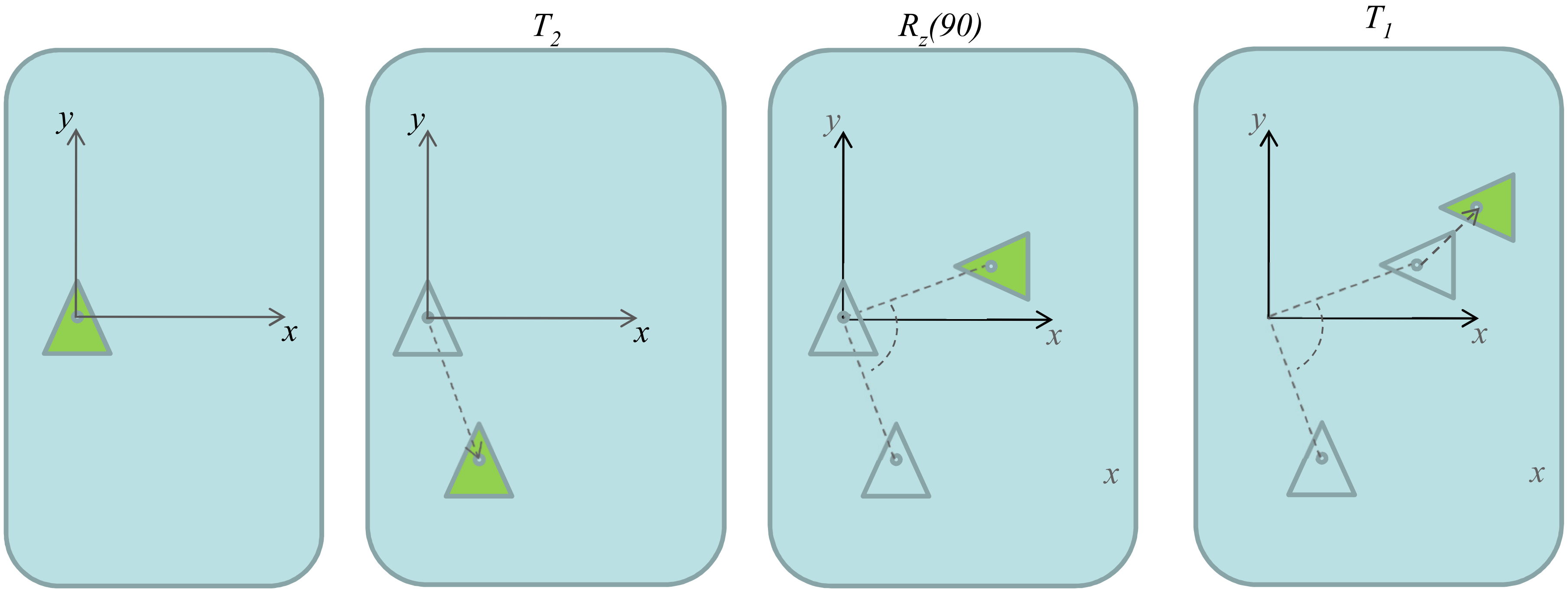
- Todas as transformações 2D podem ser definidas através de matrizes 3x3
- A composição de múltiplas transformações geométricas 2D resulta na multiplicação de matrizes 3x3
- Ou seja, qualquer transformação geométrica 2D, por mais complexa que seja, pode ser representada por uma única matriz 3x3

Composição de Transformações

- Como interpretar uma sequência de transformações?

(leitura da <u>direita para a esquerda</u>, <u>transformando o objecto</u>)

$$p' = T_1 R T_2 p$$
$$T_1 = (1,1,0), R = R_z(90), T_2 = (1,-2,0)$$

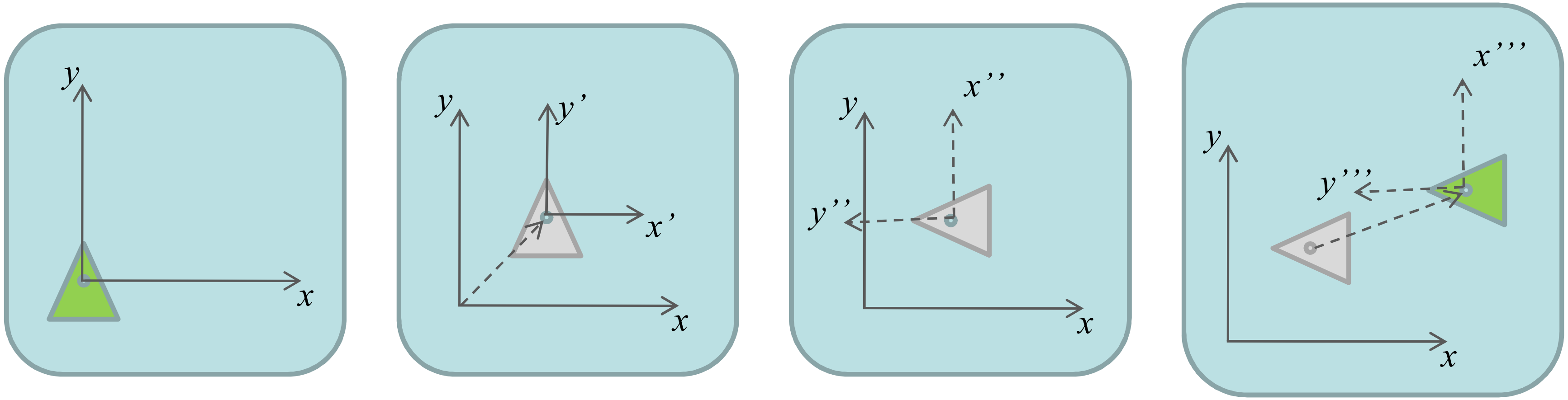


Composição de Transformações

- Lendo da esquerda para a direita podemos determinar o que acontece ao sistema de coordenadas em cada passo. O objecto é desenhado no sistema de coordenadas final.

$$p' = T_1 R T_2 p$$

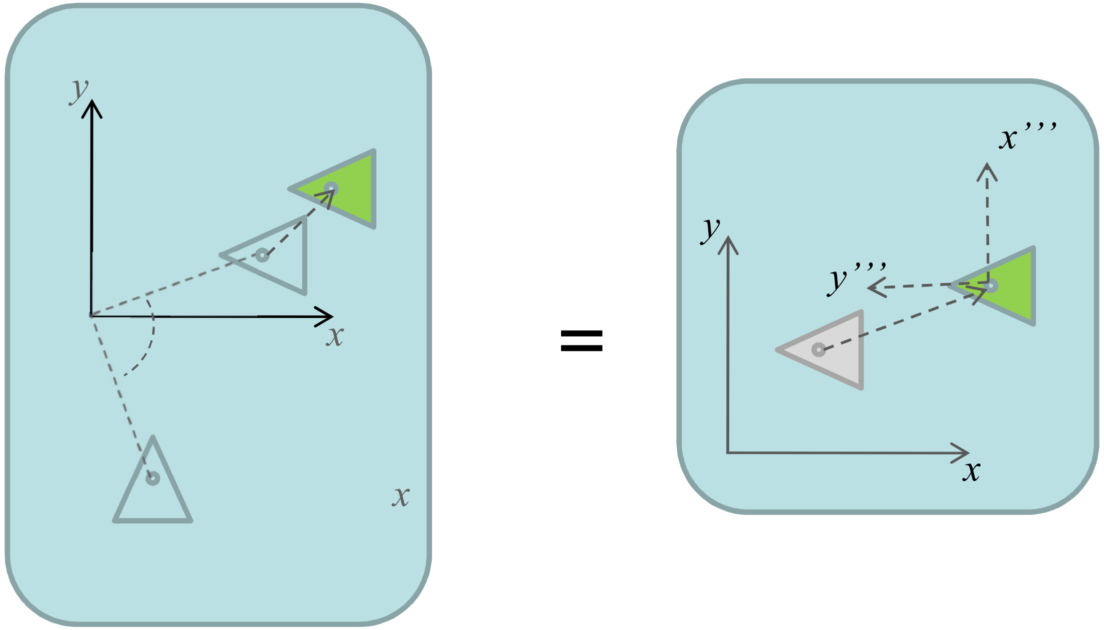
$$T_1 = (1,1,0), R = R_z(90), T_2 = (1,-2,0)$$



T_1 $R_z(90)$ T_2

Composição de Transformações

- Duas formas de ver o problema



Composição de Transformações

- Translação e Rotação

$$M = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} I \times R + T \times 0 & I \times 0 + T \times 1 \\ 0 \times R + 0 \times 1 & 0 \times 0 + 1 \times 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}$$

- Inversa

$$M^{-1} = \left\{ \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix} \right\}^{-1} =$$

$$= \begin{bmatrix} R & 0 \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & -T \\ 0 & 1 \end{bmatrix} =$$

$$= \begin{bmatrix} R^{-1} & -R^{-1}T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R^{T} & -R^{T}T \\ 0 & 1 \end{bmatrix}$$

Transformações Geométricas 3D

- Escala

Para definir uma escala uniforme em todos os eixos temos a = b = c definimos a seguinte matriz

$$p' = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p$$

Para definir uma escala não-uniforme atribuímos diferentes coeficientes na diagonal

$$p = \begin{bmatrix} \frac{1}{a} & 0 & 0 & 0 \\ 0 & \frac{1}{b} & 0 & 0 \\ 0 & 0 & \frac{1}{c} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} p'$$

Transformações Geométricas 3D

- Escala em OpenGL

  - `glScaled(GLdouble x, GLdouble y, GLdouble z)`

  - `glScalef(GLfloat x, GLfloat y, GLfloat z);`

Transformações Geométricas 3D

- Translação

$$p' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} p$$

$$p = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} p'$$

Transformações Geométricas 3D

- Translação em OpenGL

  - `glTranslate{d,f}(x,y,z);`

Transformações Geométricas 3D

- Rotação 3D em torno dos eixos
  - A rotação inversa é obtida pela inversa da matriz, ou seja, pela transposta

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

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

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

Transformações Geométricas 3D

- Rotação em torno de uma direcção arbitrária (solução algébrica)
  - Pretende-se aplicar uma rotação de α graus em torno de n

$p' = u\cos(\alpha) + v\sin(\alpha)$

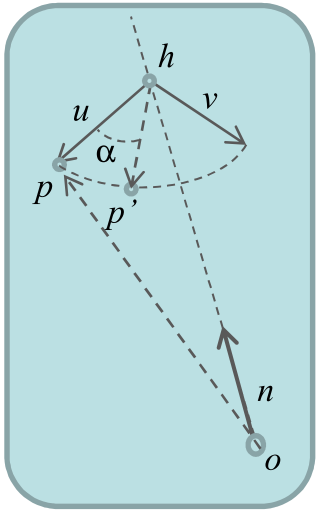
Para determinar u precisamos de calcular h

Cálculo da projecção do vector op em n resulta no vector oh

$h = o + oh$

$u = p - h$

$v = n \times u$



Sendo n = (x,y,z) obtem-se a seguinte form matricial

$$R = \begin{bmatrix} tx^2 + c & txy + sz & txz - sy & 0 \\ txy - sz & ty^2 + c & tyz + sx & 0 \\ txz + sy & tyz - sx & tz^2 + c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$c = \cos(\alpha), \quad s = \sin(\alpha), \quad t = (1-c)$

Transformações Geométricas 3D

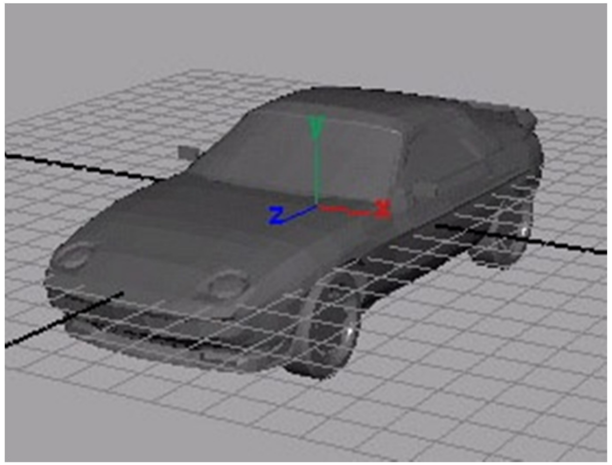
- Rotação em OpenGL

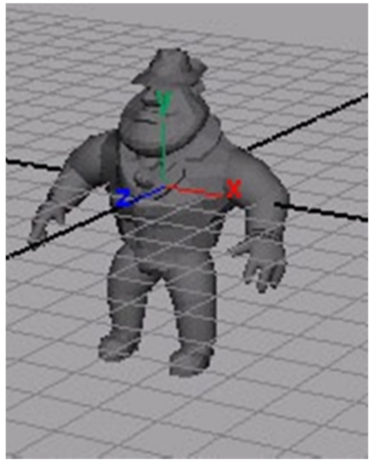
  - `glRotate{d,f}(ang,x,y,z);`

    - sendo ang o ângulo de rotação em graus;
    - e (x,y,z) o vector que define o eixo de rotação;

Sistemas de Coordenadas

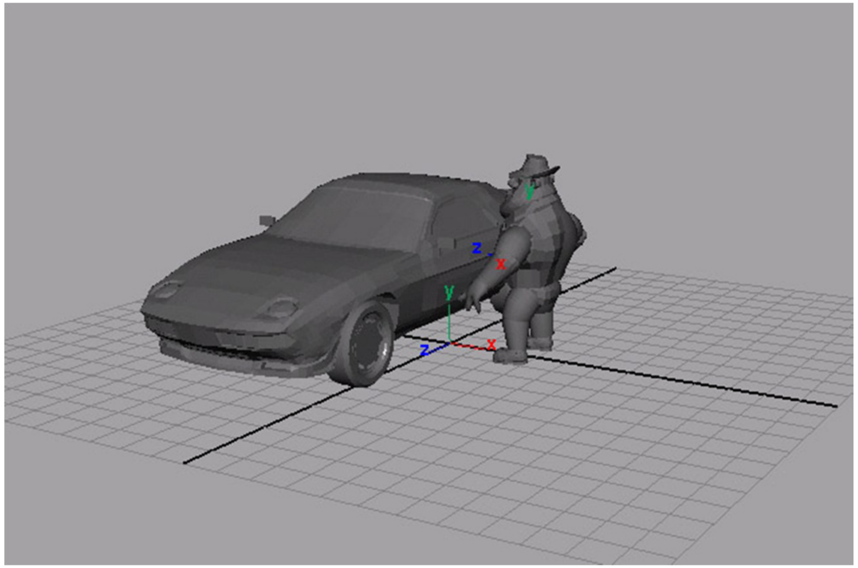
- Object Space ou Modelling Space (Espaço local)
  - Este espaço é o sistema de coordenadas relativas a um objecto (ou grupo de objectos).
  - Permite-nos definir coordenadas relativas.





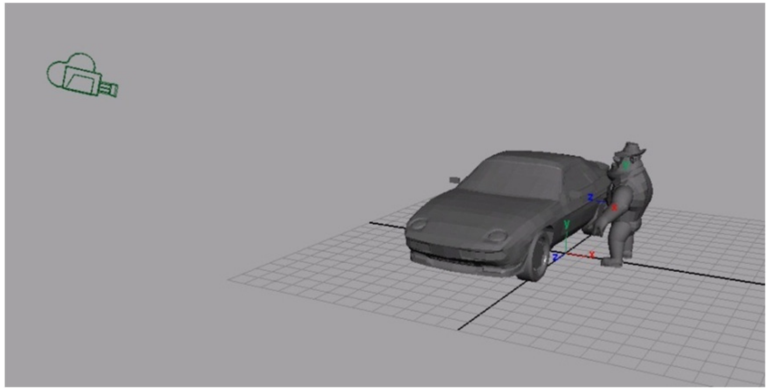
Sistemas de Coordenadas

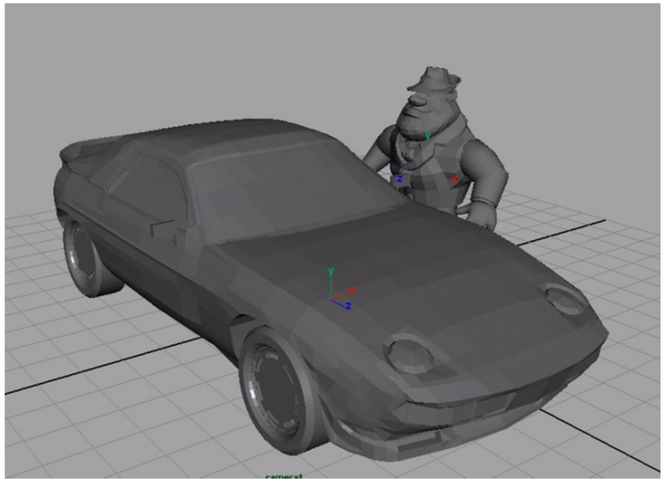
- World Space (Espaço Global)
  - Este espaço engloba todo o *universo* e permite-nos exprimir as coordenadas de forma *absoluta*.
  - É neste espaço que os modelos são compostos para criar o mundo virtual



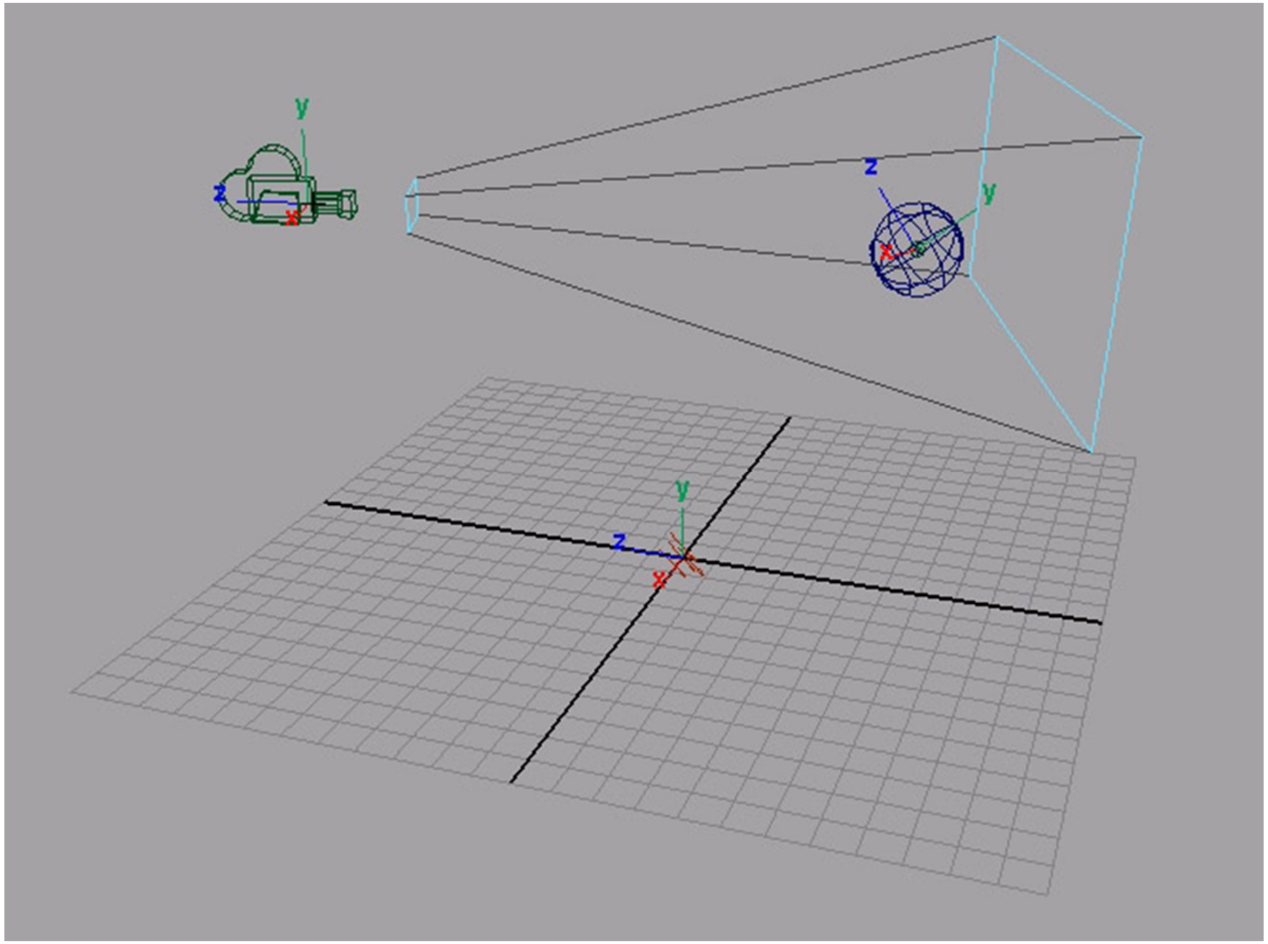
Sistemas de Coordenadas

- Camera Space (Espaço da Câmara)
  - Este sistema de coordenadas esta associado ao observador, ou câmara.
  - A sua origem é a posição da câmara.
  - O seu sistema de eixos é determinado pela orientação da câmara.



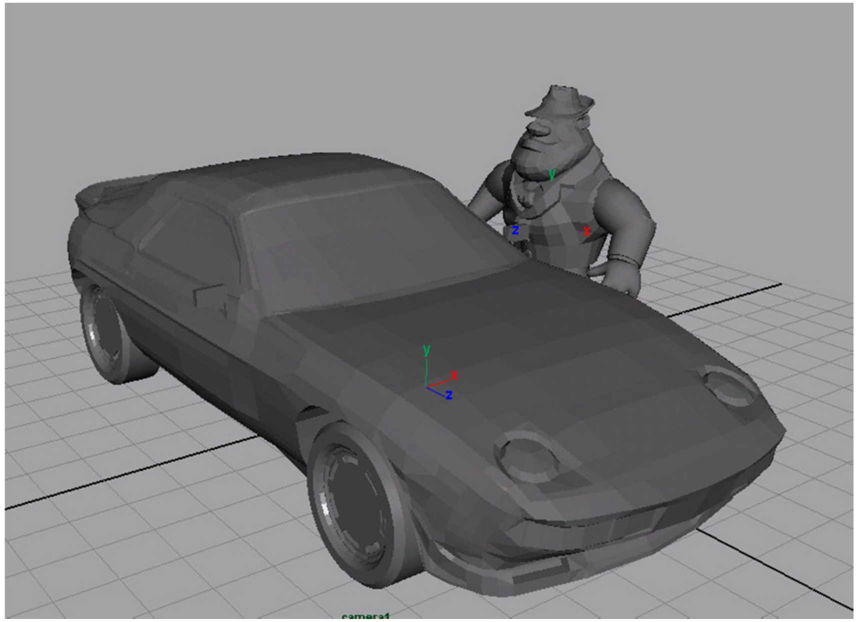


Sistemas de Coordenadas



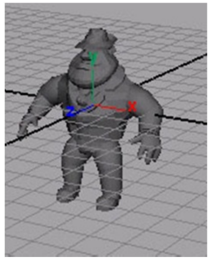
Sistemas de Coordenadas

- Screen Space (Espaço do ecrã)
  - Espaço 2D onde é visualizado o mundo virtual
  - Resultado de uma projecção

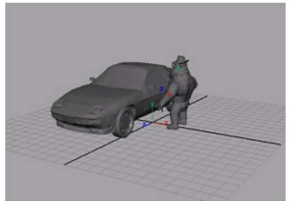


Sistemas de Coordenadas

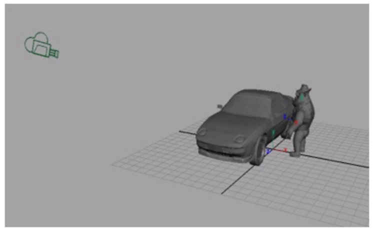
Object Space



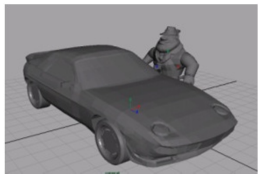
World Space



Camera Space



Screen Space



Transformações Geométricas

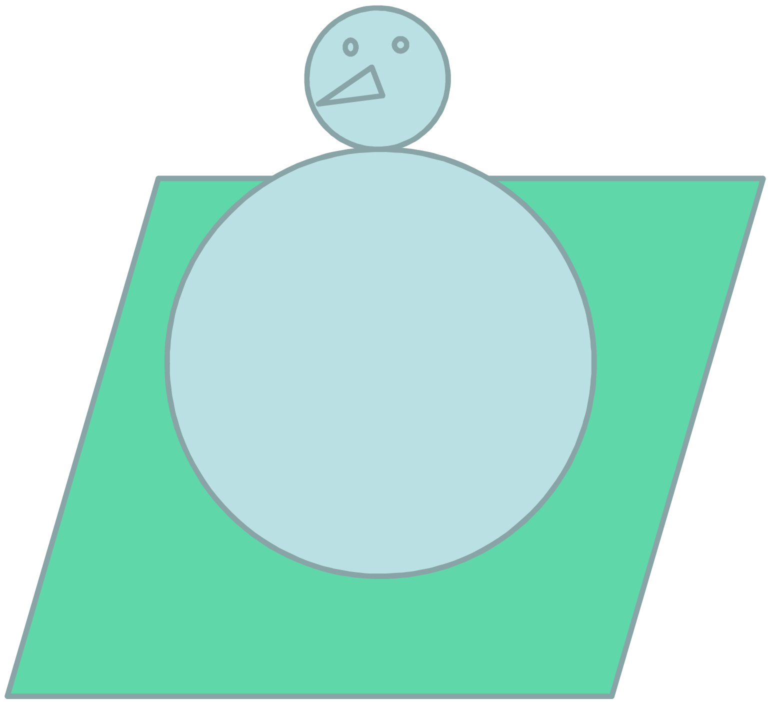
- As transformações mencionadas até agora permitem-nos posicionar os objectos no espaço global.

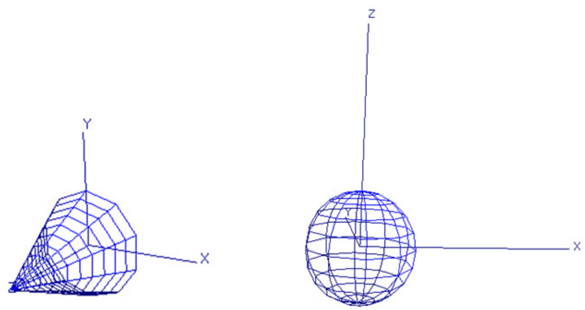
Demo!!!

(transformações geométricas)

Transformações Geométricas

- Desenhar um boneco de neve!

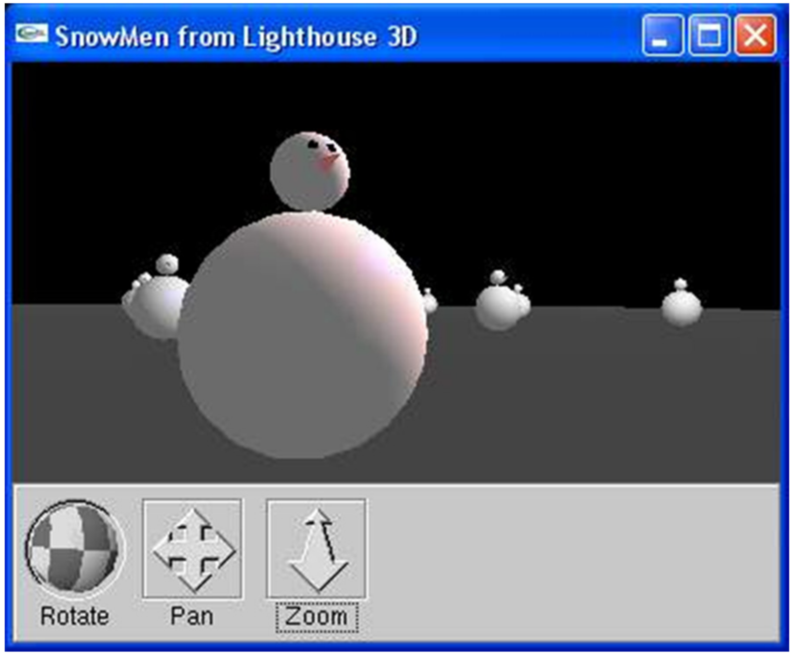




Transformações Geométricas

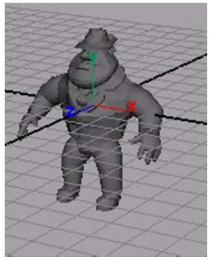
```
void drawSnowMan() {
          glColor3f(1.0f, 1.0f, 1.0f);

// Draw Body
          glTranslatef(0.0f ,0.75f, 0.0f);
          glutSolidSphere(0.75f,20,20);

// Draw Head
          glTranslatef(0.0f, 1.0f, 0.0f);
          glutSolidSphere(0.25f,20,20);

// Draw Eyes
          glPushMatrix();
          glColor3f(0.0f,0.0f,0.0f);
          glTranslatef(0.05f, 0.10f, 0.18f);
          glutSolidSphere(0.05f,10,10);
          glTranslatef(-0.1f, 0.0f, 0.0f);
          glutSolidSphere(0.05f,10,10);
          glPopMatrix();

// Draw Nose
          glColor3f(1.0f, 0.5f , 0.5f);
          glutSolidCone(0.08f,0.5f,10,2);
}
```

Modelar um boneco de neve com esferas e um cone

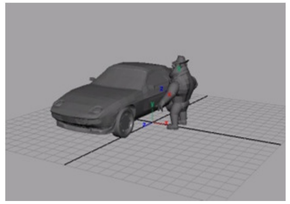


Transformações Geométricas

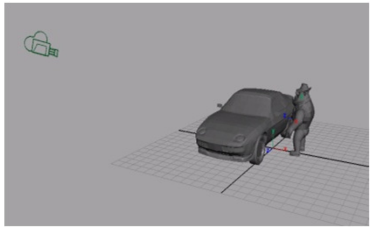
Object Space



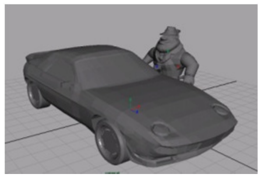
World Space



Camera Space



Screen Space

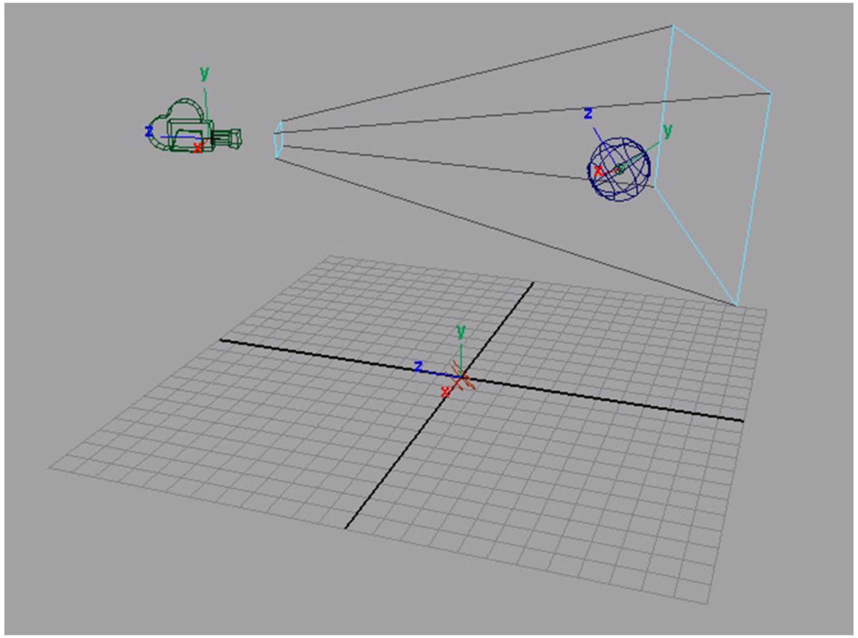


Transformações Geométricas

- Por omissão (em OpenGL) considera-se que a câmara se encontra na origem, a apontar na direcção do Z negativo.
- Como definir uma câmara com posição e orientação arbitrárias?
- Que dados são necessários para definir uma câmara?

Transformações Geométricas

- Dados para definir uma câmara:
  - posição
  - direcção
  - "este lado para cima"

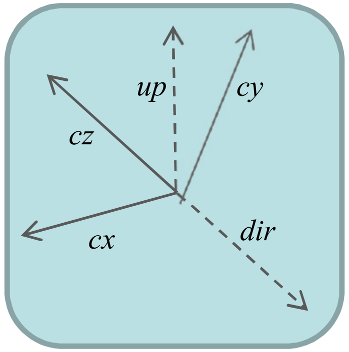


Transformações Geométricas

- Operações sobre a câmara:
  - *Definição do sistema de coordenadas*
  - Translação da posição da câmara

- Podemos facilmente especificar os eixos do sistema de coordenadas da câmara. Assumindo que os vectores fornecidos se encontram normalizados

- É necessário normalizar todos os vectores (cx, cy, cz)



$$cz = -dir$$
$$cx = dir \times up$$
$$cy = cx \times dir$$

Transformações Geométricas

- Podemos então definir uma matriz de rotação baseada nos vectores encontrados para a câmara:

$$M = \begin{bmatrix} cx_1 & cy_1 & cz_1 \\ cx_2 & cy_2 & cz_2 \\ cx_3 & cy_3 & cz_3 \end{bmatrix}$$

- Se pretendessemos colocar um objecto na posição da câmara poderiamos utilizar a seguinte transformação:

$$F = TM$$

- sendo T a translação de acordo com a posição da câmara

Transformações Geométricas

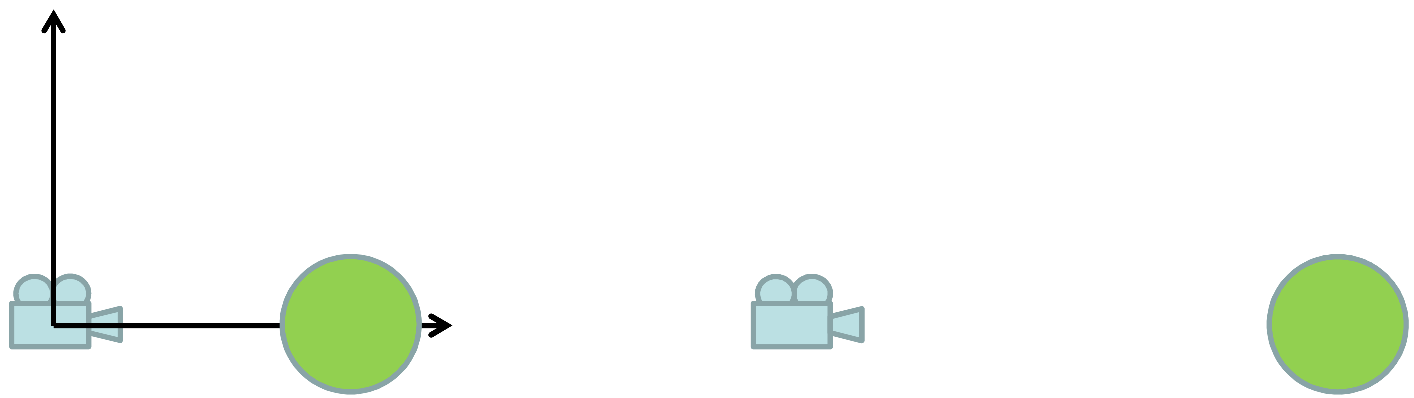
- Podemos então definir uma transformação que permita posicionar um objecto na posição da câmara:

$$F = \begin{bmatrix} M & T \\ o & 1 \end{bmatrix}$$

- Mas não é isto que pretendemos!

Transformações Geométricas

- A operação de projecção (introduzida mais tarde) assume que a câmara se encontra na origem.
- Como simular a colocação da câmara num ponto arbitrário e com uma orientação também arbitrária?



Câmara e Objecto nas posições originais   Câmara e Objecto nas posições desejadas

Transformações Geométricas

- Duas opções:
  - Move-se a câmara para a posição desejada
  - Move-se o objecto no sentido inverso

Aplicar uma transformação à câmara é equivalente a aplicar a transformação inversa ao objecto

- Se a transformação a aplicar para posicionar a câmara é

$$F = \begin{bmatrix} M & T \\ o & 1 \end{bmatrix}$$

- Então, aplicamos a transformação inversa aos objectos

$$F^{-1} = \begin{bmatrix} M^{-1} & M^{-1}(-T) \\ o & 1 \end{bmatrix}$$

$$= \begin{bmatrix} M^{T} & M^{T}(-T) \\ 0 & 1 \end{bmatrix}$$

Transformações Geométricas

- Posicionamento da câmara em OpenGL

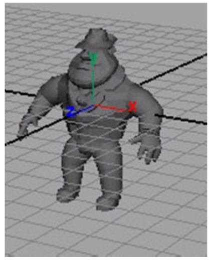
```
– gluLookAt(     posx, posy, posz,
                 atx, aty, atz,
                 upx, upy, upz)
```

sendo:

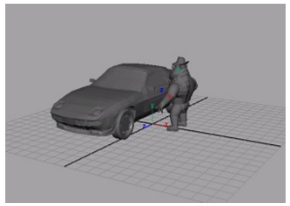
pos – a posição da câmara
at – um ponto para onde a câmara aponta
up – a direcção do vector vertical

Transformações Geométricas

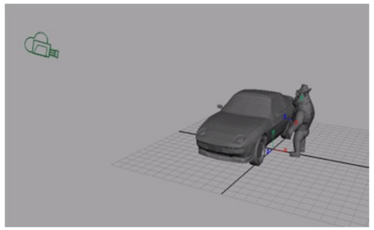
Object Space



World Space



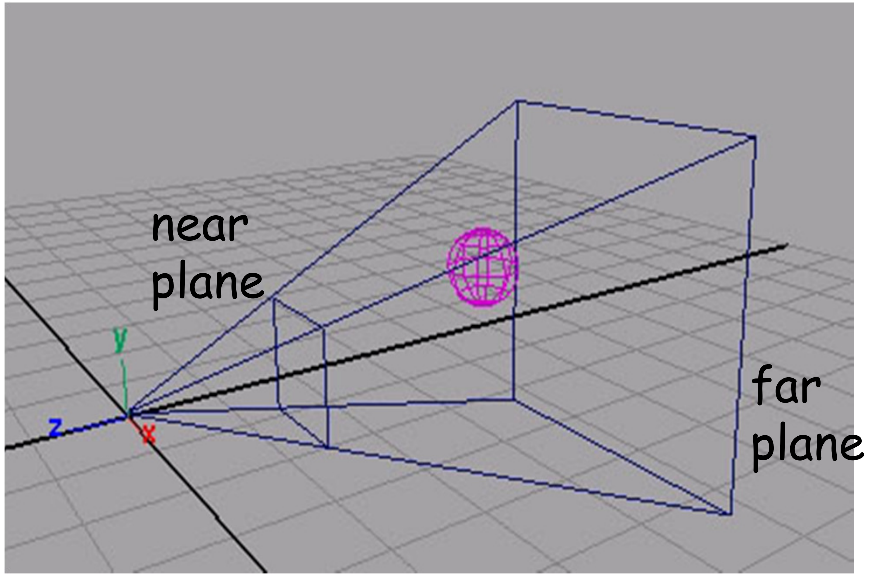
Camera Space



Screen Space

Transformações Geométricas

- Perspectiva - View Frustum
  - Pirâmide truncada que define a região visível



Em OpenGL o plano de projecção é o near plane

Transformações Geométricas

- O plano de projecção é um plano perpendicular ao eixo do Z, a uma distância n da origem

- A câmara encontra-se situada na origem, a apontar na direcção do eixo do Z negativo

- Calculo das projecções de um ponto 3D (p_x, p_y, p_z)(no espaço câmara) no plano de projecção

$$x = -\frac{n}{Pz} Px$$

$$y = -\frac{n}{Pz} Py$$

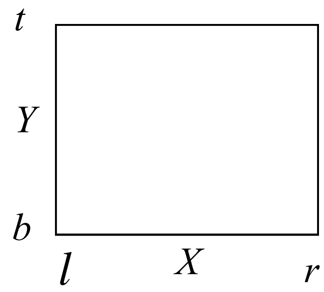
Transformações Geométricas

- Clip Space
  - O clip space é um espaço intermédio entre o espaço câmara e o espaço ecrã.
  - O view frustum é convertido para um cubo cuja gama de valores nas três coordenadas é [-1,1].
  - Desta forma, é extremamente simples determinar qual a geometria que se encontra dentro do view frustum.

Transformações Geométricas

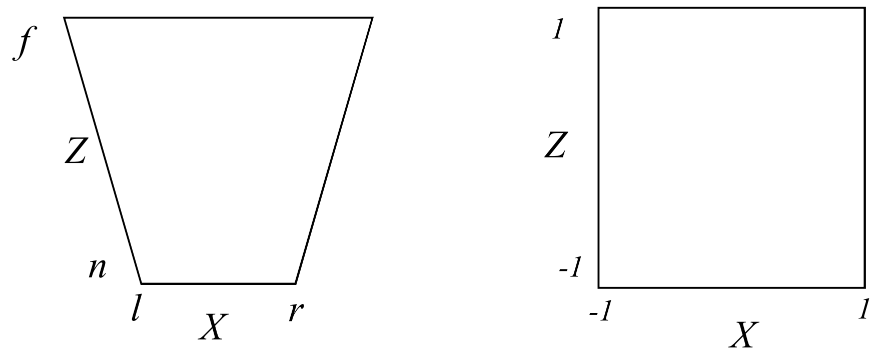
- O plano de projecção é definido pelos seus limites de variação
- $x = [l,r]$ e $y = [t,b]$

plano de projecção



ViewFrustum => Clip space

limites de variação de $z = [n,f]$



Transformações Geométricas

- Definição do Frustum em OpenGL
  - `glFrustum(left,right,bottom,top,near,far);`
- Os parâmetros definem o frustum de visualização.

Transformações Geométricas

- O GLU fornece uma alternativa simpática:

  - `gluPerspective(fy, ratio, near,far);`

  - sendo
    - fy – ângulo de visão em y.
    - ratio – relação fovx/fovy

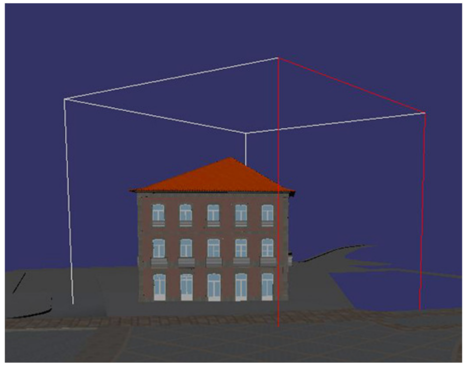
$$f_y = \frac{\arctan((top - bottom)}{2 * near}$$

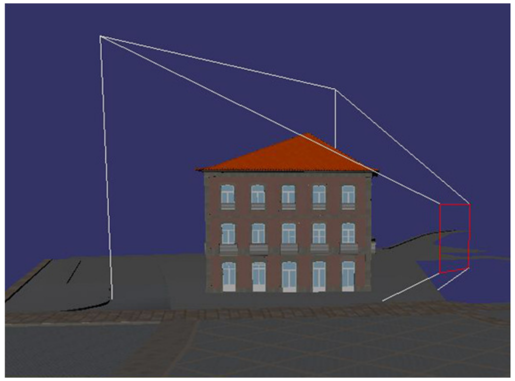
Transformações Geométricas

- Projecção Ortográfica em OpenGL
  - `glOrtho(left,right,bottom,top,near,far);`
- Neste caso temos um paralelepípedo de visualização.

Transformações Geométricas

- Projecções





Transformações Geométricas

- Screen Space

  - Sejam c_x e c_y as coordenadas normalizadas em clip space de um ponto

  - As coordenadas do viewport, ou janela, (w_x, w_y), com uma determinada largura l e altura a são definidas da seguinte forma:

$$w_x = (c_x + 1)\frac{l}{2}$$

$$w_y = (c_y + 1)\frac{a}{2}$$

Transformações Geométricas

- Viewport em OpenGL
  - `glViewport(x,y,width,height);`

Transformações Geométricas

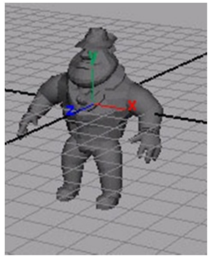
Demo

(projecções - Nate Robbins)

Transformações Geométricas

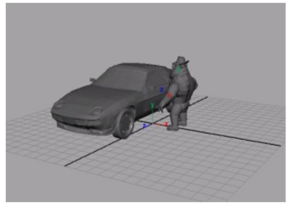
- Matrizes em OpenGL

Object Space

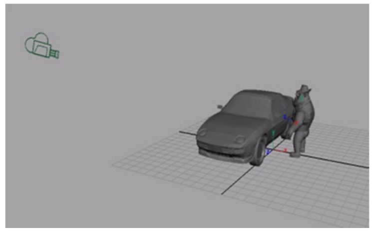


ModelView
GL_MODELVIEW

World Space

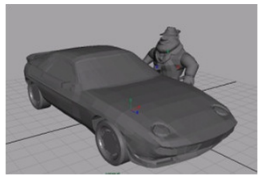


Camera Space



Projection
GL_PROJECTION

Screen Space



OpenGL

```
void changeSize(int w, int h) {

    // Prevent a divide by zero, when window is too short
    // (you cant make a window of zero width).
    if(h == 0)
        h = 1;
    float ratio = 1.0* w / h;

    // Set the viewport to be the entire window
     glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    // Reset the coordinate system before modifying
    glLoadIdentity();

    // Set the correct perspective.
    gluPerspective(45,ratio,1,1000);

    glMatrixMode(GL_MODELVIEW);
}
```

Setup da projecção

Necessário quando a janela sofre modificações, ou ao iniciar a aplicação

OpenGL

```
void renderScene(void) {

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0,
              0.0,0.0,0.0,
              0.0f,1.0f,0.0f);

    glRotatef(a,0.1,0);
    glutSolidTeapot(1);

    a++;
    glutSwapBuffers();
}
```

Buffers

- Color Buffer

  - O OpenGL permite ter 2 buffers distintos.

  - Em cada instante visualiza-se um buffer e escreve-se no outro.

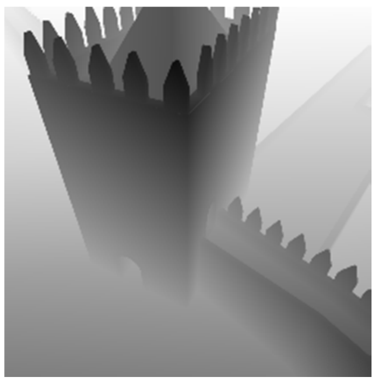
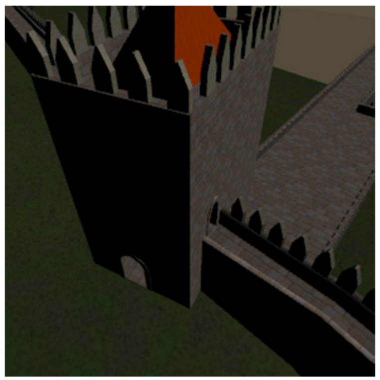
  - No final da frame trocam-se os buffers.

Buffers

- Color Buffer em OpenGL
  - Na inicialização
    - `glutDisplayMode(GLUT_DOUBLE | ...);`
  - No final de cada frame
    - `glutSwapBuffers();`

Buffers

- Depth Buffer ou Z-Buffer
  - Buffer que armazena os valores de Z dos pixels que já foram desenhados
  - Permite assim criar uma imagem correcta sem ser necessário ordenar e dividir polígonos


Buffers

- Depth Buffer em OpenGL
  - Na inicialização
    - `glutInitDisplayMode(GLUT_DEPTH | ... );`
    - `glEnable(GL_DEPTH_TEST);`
  - No início de cada frame
    - `glClear(GL_DEPTH_BUFFER_BIT | ...);`

Buffers

- limitações do Z-Buffer - precisão

near = 0.01
far=100.0

near = 0.0001
far=100.0

near = 0.000001
far=100.0

Buffers

- limitações do Z-Buffer
  - número de bits determina precisão
  - Z-Buffer não é linear: mais detalhe perto do *near plane*
  - Muitos bits são usados para distâncias curtas

Buffers

- A precisão do Z-Buffer é definida por intervalos crescentes desde o near plane até ao far plane

- Exemplo (16 bits):
  - zNear = 1; zFar = 1000
  - z = 10 : intervalo 0.00152
  - z = 900 : intervalo 12.51

Buffers

- A precisão do Z-Buffer é dependente da relação entre o near plane e o far plane

- Exemplo (16 bits):

  - zFar = 1000; z = 900
  - zNear = 1   : intervalo 12.51
  - zNear = 0.01: intervalo 143.25

Buffers

- Z-Buffer: mais bits => mais precisão

- Exemplo :

  - zFar = 1000; z = 900; zNear = 0.01
  - 24 bits: intervalo 0.483
  - 16 bits: intervalo 143.25

Referências

- Mathematics for 3D Game Programming & Computer Graphics, Eric Lengyel
- 3D Math Primer for Graphics and Game Development, Fletcher Dunn e Ian Parberry
- Interactive Computer Graphics: A Top Down Approach with OpenGL, Edward Angel
- OpenGL Reference Manual, OpenGL Architecture Review Board
- "Learning to love your z-buffer, http://sjbaker.org/steve/omniv/love_your_z_buffer.html