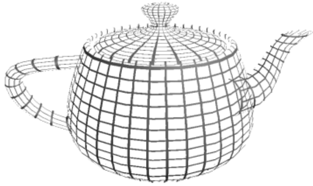
Computação Gráfica

Introdução +
Transformações Geométricas

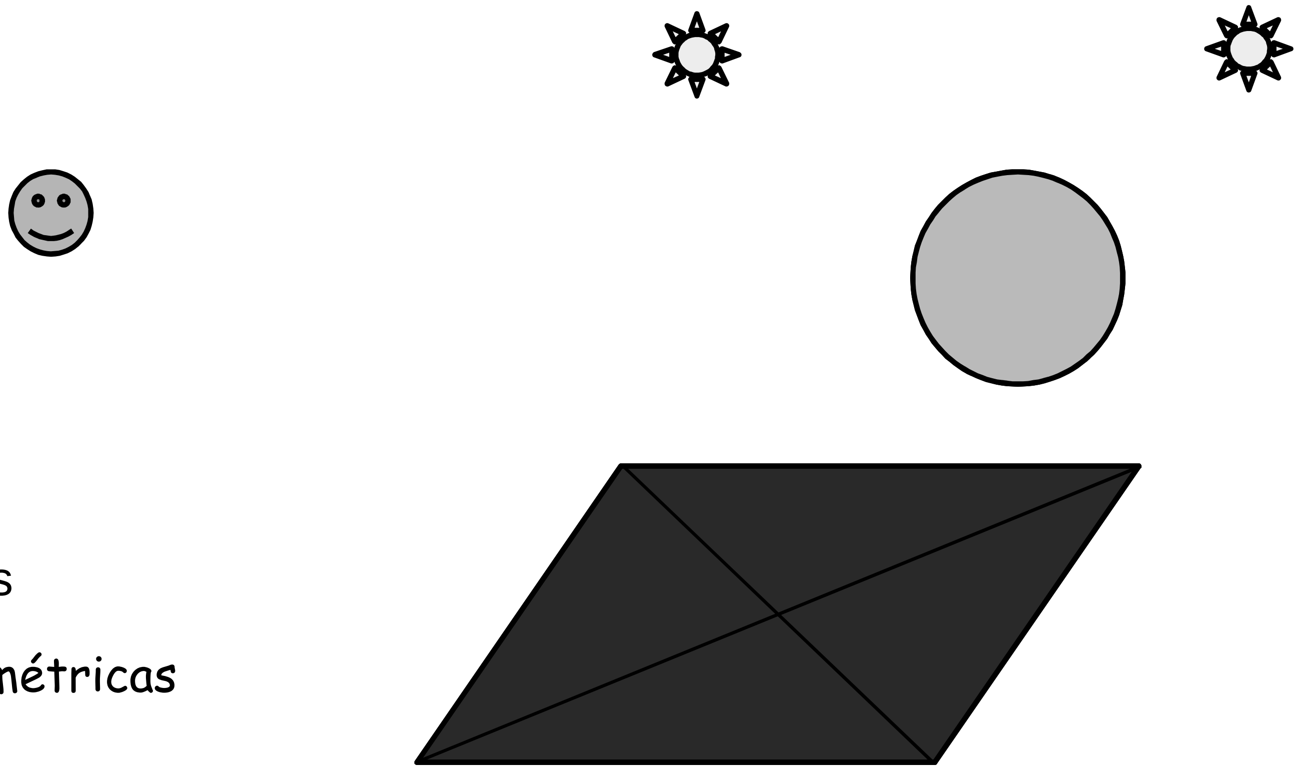


Computação Gráfica

- Processamento de Imagem + Visão por Computador
  Extrair informação a partir de uma ou mais (vídeo) imagens
- Modelação
  Construção do modelo geométrico de um mundo virtual
- Visualização Científica
  Extracção de informação a partir de dados abstractos e apresentação da mesma sob a forma de imagens
- Síntese de Imagem (*Rendering*)
  Construção de uma imagem a partir de uma modelo virtual de uma cena (mundo virtual)

Rendering	Realista	Expressiva
2D		
3D	X	

Rendering : Contexto

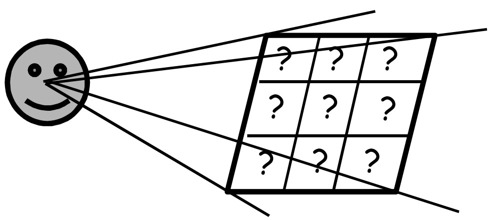


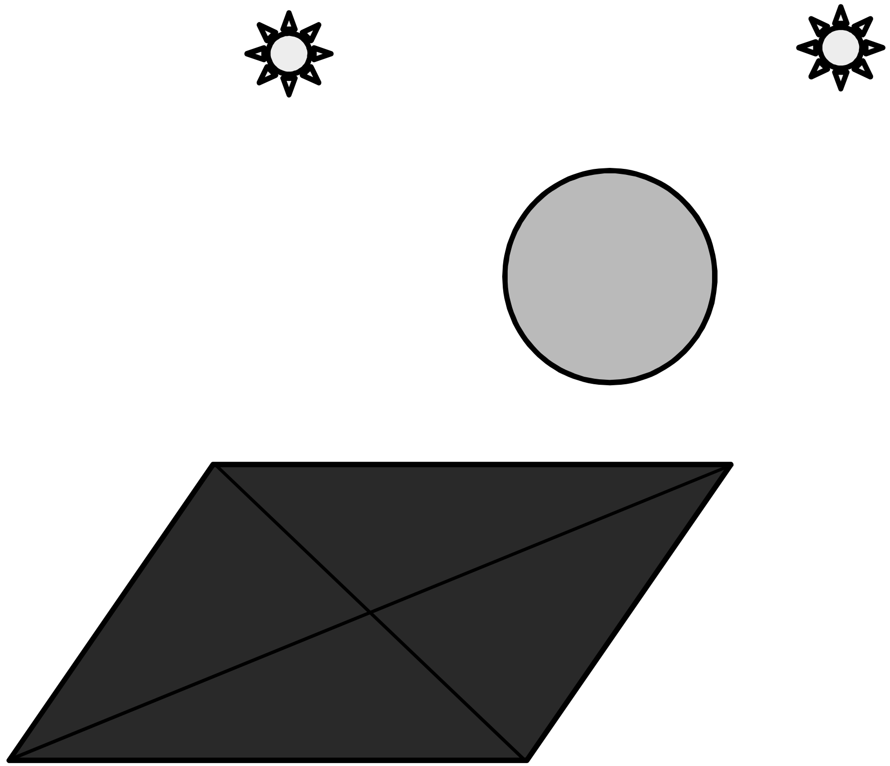
Entidades

- Primitivas geométricas
- Materiais
- Fontes de luz
- Câmara

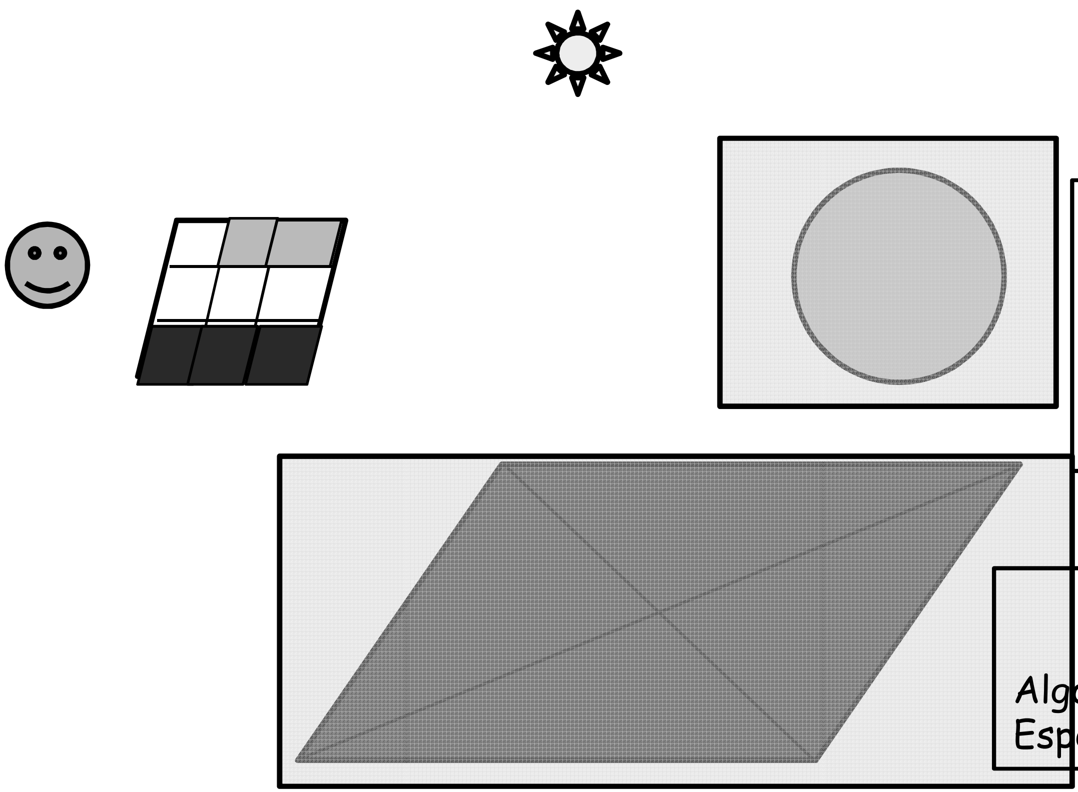
Visibilidade

- A síntese de imagem exige a determinação de qual o objecto que é visível ao longo de cada direcção





Visibilidade: *rasterização*



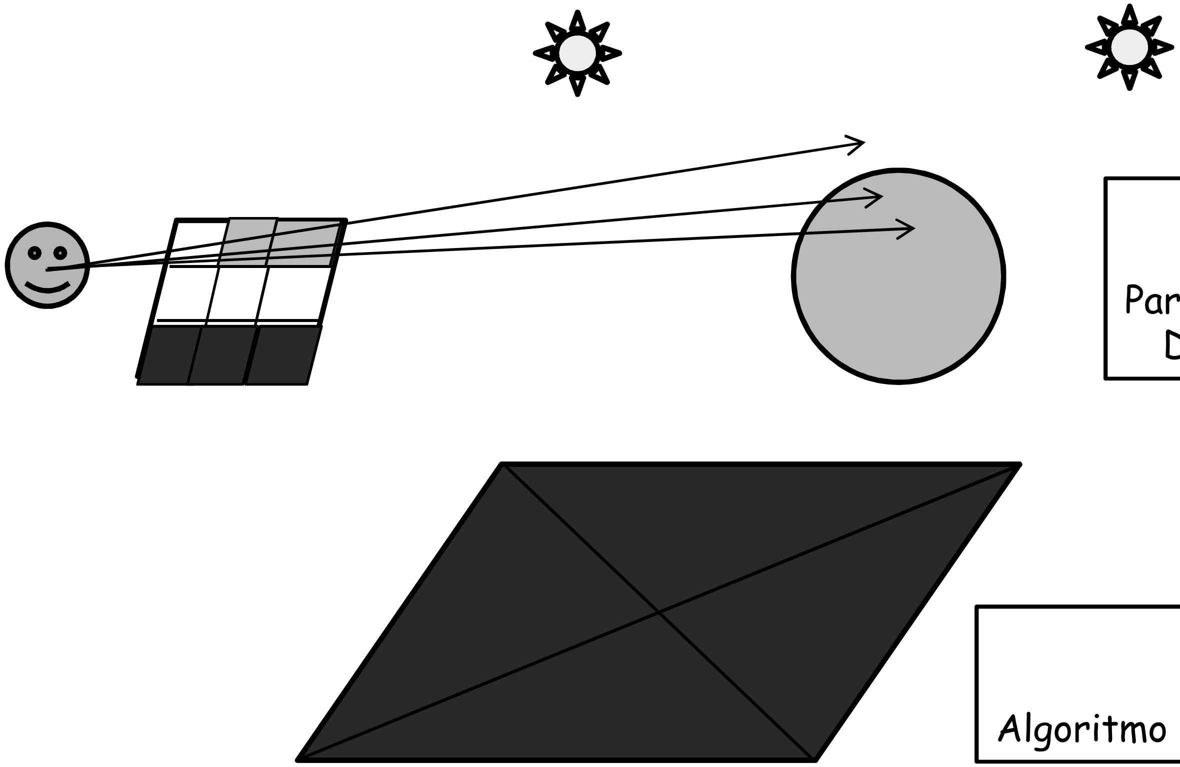
Raster

Para cada objecto
Projectar no plano de imagem
determinando em que pixeis projecta
(atenção à profundidade: *Z-buffer*)

Raster

Algoritmo de visibilidade utilizado pelas GPUs
Especialmente eficazes a realizar projecções

Visibilidade: *ray tracing*



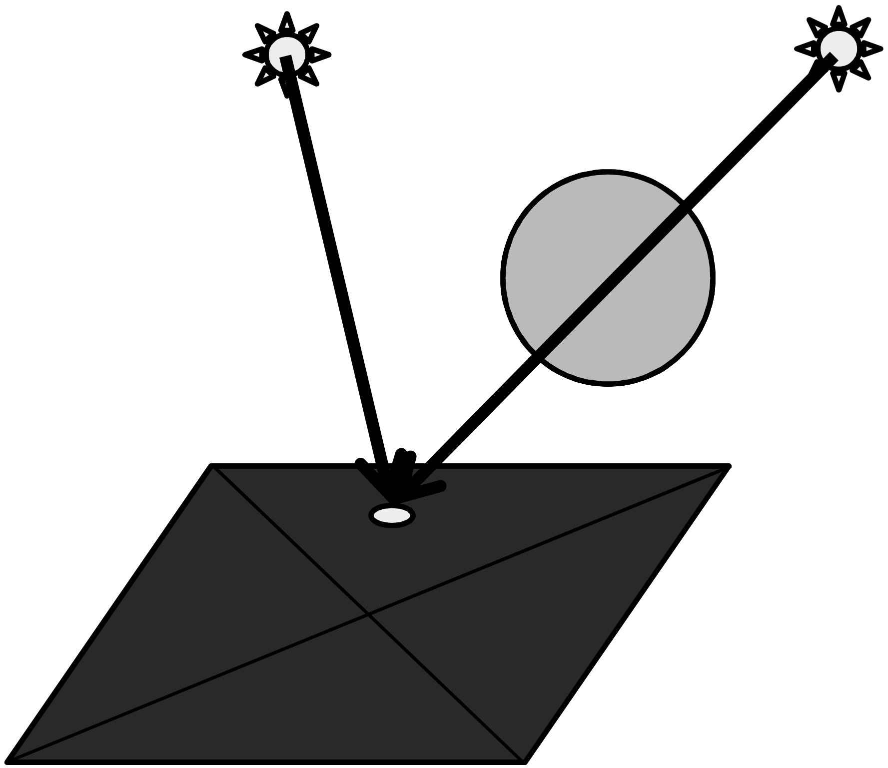
Ray Tracing

Para cada pixel
  Determinar o objecto que aí projecta

Ray tracing

Algoritmo de visibilidade utilizado pelos CPUs

Iluminação Local vs. Global



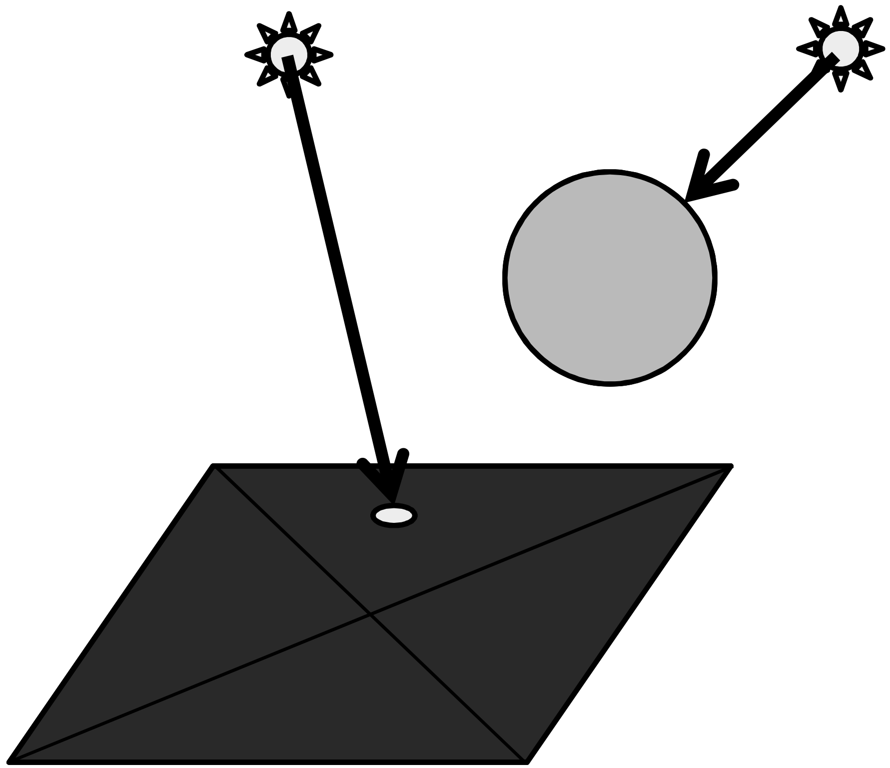
Iluminação Local

A iluminação de um ponto apenas tem em consideração o próprio ponto e a posição das fontes de luz; os restantes objectos são ignorados.

Não simula fenómenos como sombras, reflexões, etc.

Modelo de iluminação usado por defeito pelos GPUs

Iluminação Local vs. Global



Iluminação Global

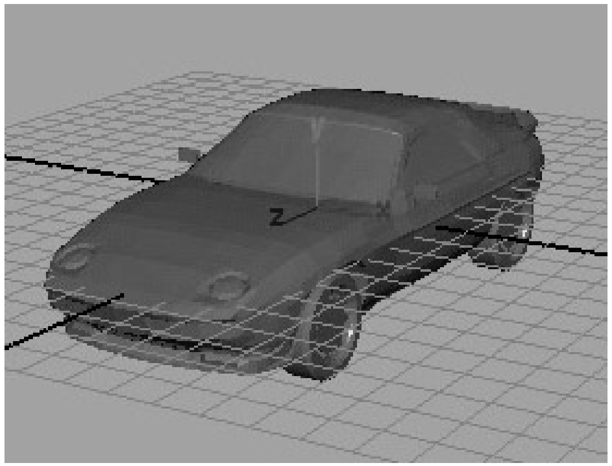
A iluminação de um ponto tem em consideração o próprio ponto, a posição das fontes de luz bem como todos os objectos da cena.

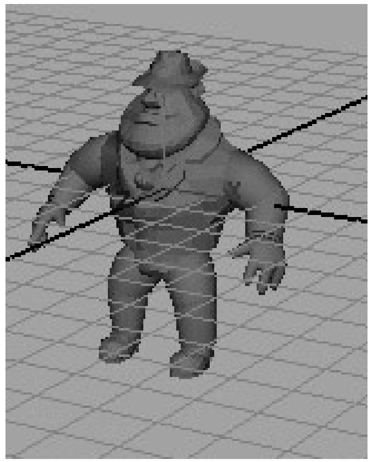
Simula fenómenos como sombras, reflexões, transmissões, etc.

Modelo de iluminação associado ao *ray tracing*

Sistemas de Coordenadas

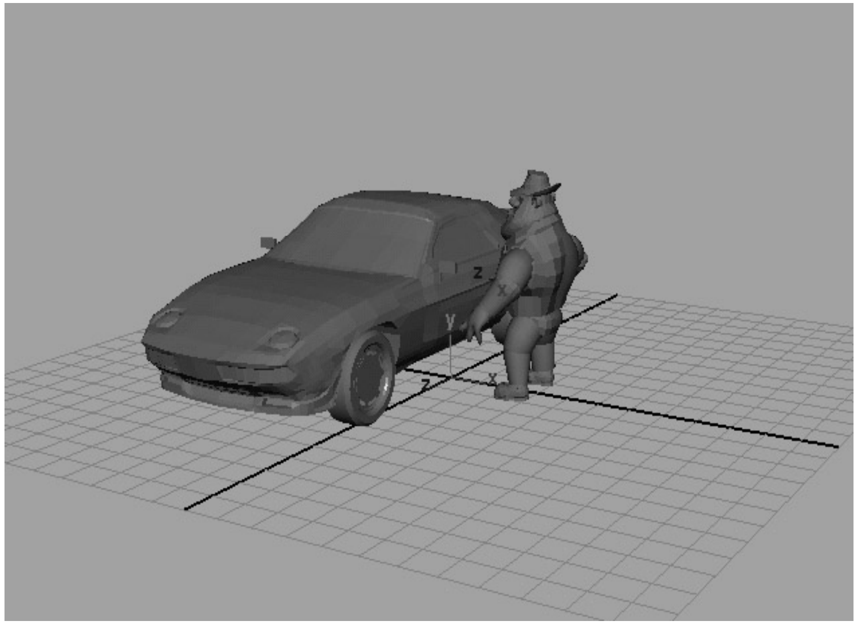
- Object Space ou Modelling Space (Espaço local)
  - Este espaço é o sistema de coordenadas relativas a um objecto (ou grupo de objectos).
  - Permite-nos definir coordenadas relativas.



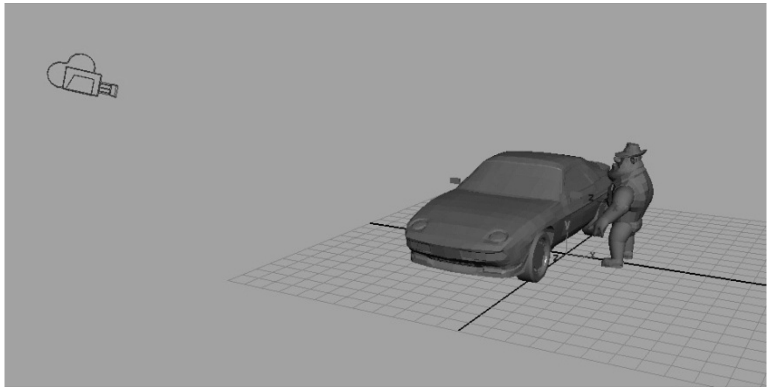


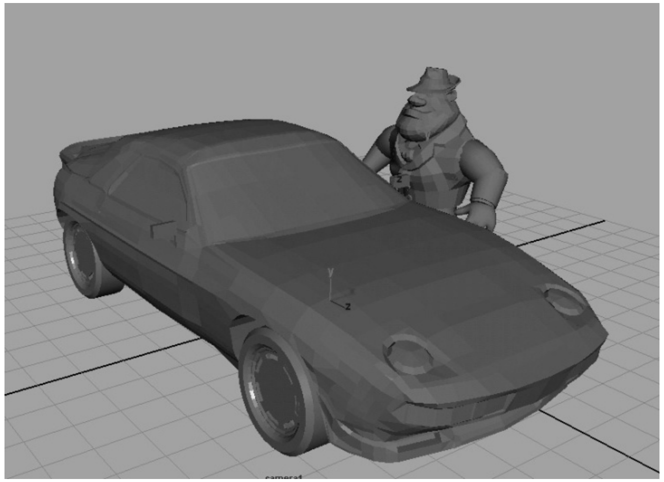
Sistemas de Coordenadas

- 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



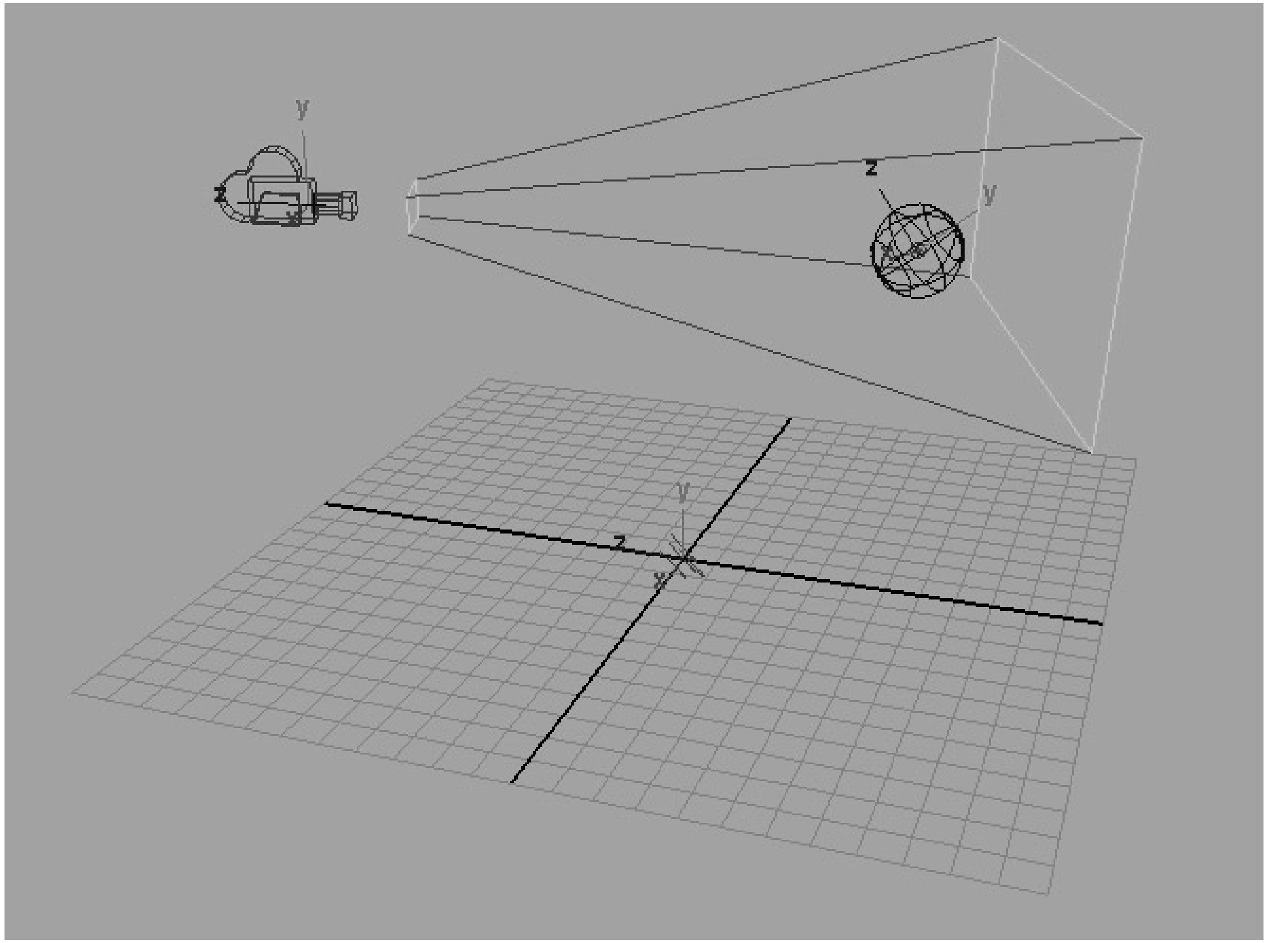
Sistemas de Coordenadas





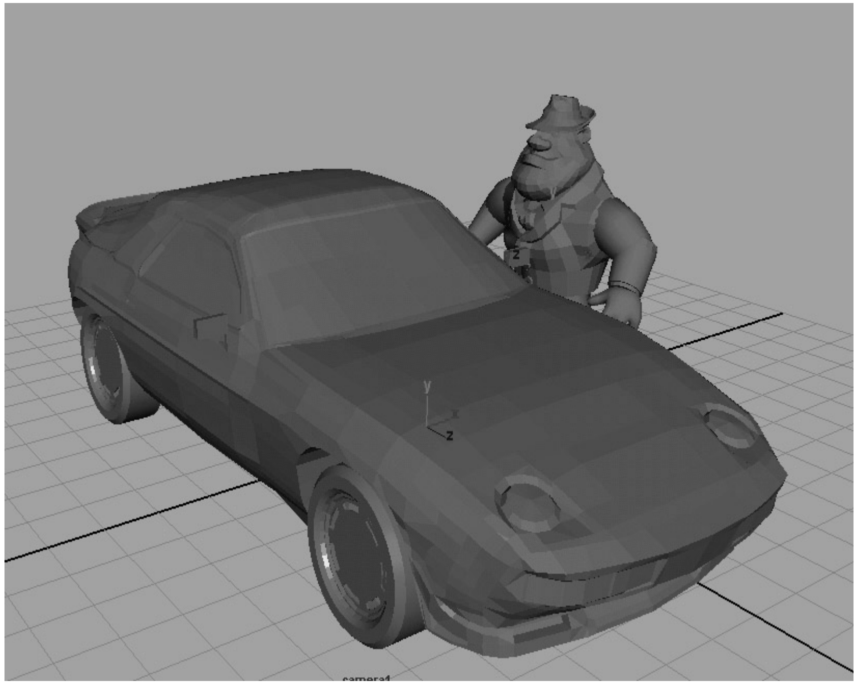
- Camera Space (Espaço da Câmara)
  - Este sistema de coordenadas está associado ao observador, ou câmara.
  - A sua origem é a posição da câmara.
  - O seu sistema de eixos é determinado pela orientação da câmara.

Sistemas de Coordenadas



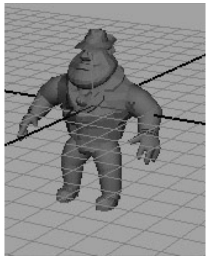
Sistemas de Coordenadas

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

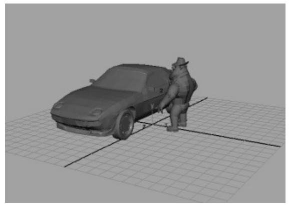


Sistemas de Coordenadas

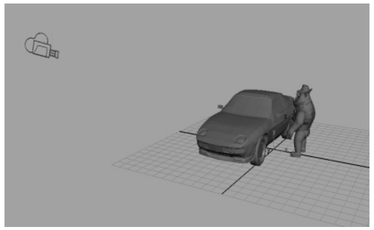
Object Space



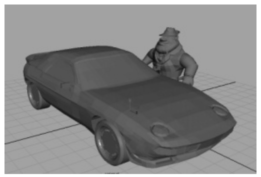
World Space



Camera Space



Screen Space



Vectores

- Magnitude

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

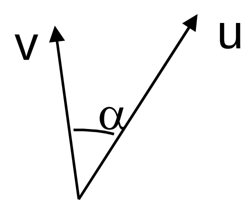
- Vector Normalizado (mag = 1)

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

- Produto Interno

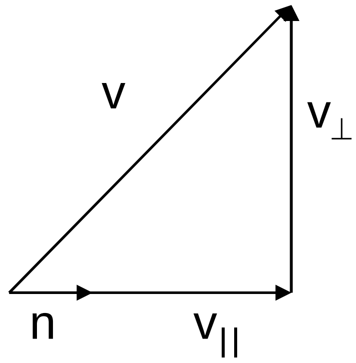
$$v \bullet u = \sum_{i=1}^{3} v_i * u_i$$

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



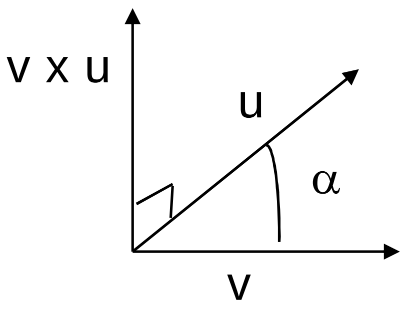
Vectores

- Projecção
  (projecção de v em n:
  qual a magnitude do vector v na
  direcção do vector n)



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

- Produto Externo
  (t = v x u é um vector
  perpendicular a v e u)



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

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

Transformações Geométricas

- Consideremos a matriz identidade e um ponto no sistema de coordenadas global

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + a_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- As coordenadas do ponto *a* podem ser expressas em função das colunas da matriz

- O ponto *a* é uma combinação linear dos vectores coluna da matriz

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1 \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + a_3 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Transformações Geométricas

- Um triplo de vectores (u,v,w) pode definir um sistema de coordenadas 3D desde que sejam linearmente independentes.

- Um conjunto de 3 vectores (u,v,w) é linearmente independente se nenhum dos vectores se puder escrever como uma combinação linear dos restantes,

- ou seja, não existe nenhuma combinação de números α, β, φ, sendo pelo menos um deles diferente de zero, tal que

$$\alpha v + \beta u + \varphi w = 0$$

Transformações Geométricas

- 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.

Transformações Geométricas

- Vejamos o que acontece quando os vectores unitários x,y,z são transformados por uma matriz arbitrária M invertível

- O resultado são as colunas da matriz M

- As colunas da matriz M formam os eixos de um sistema de coordenadas

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} m_{13} \\ m_{23} \\ m_{33} \end{bmatrix}$$

Transformações Geométricas

- Da mesma forma, assumindo que M é invertível, aplicar a inversa de M aos vectores coluna de M dá o seguinte resultado:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}^{-1} \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}^{-1} \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}^{-1} \begin{bmatrix} m_{13} \\ m_{23} \\ m_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Transformações Geométricas

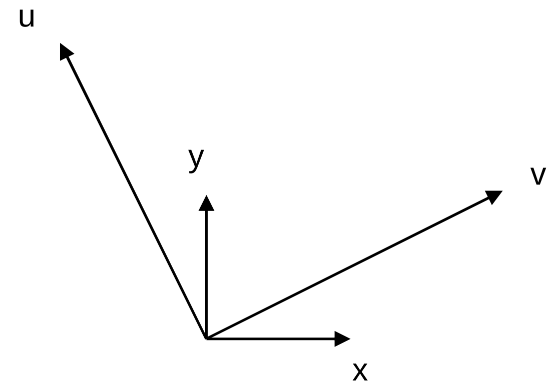
- Visualização de uma matriz 2D

  sistema preto

$$S = [x \quad y] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

  sistema azul

$$M = [v \quad u] = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$$



- O vector x é transformado no vector v, e o vector y é transformado no vector u

Transformações Geométricas

$$v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

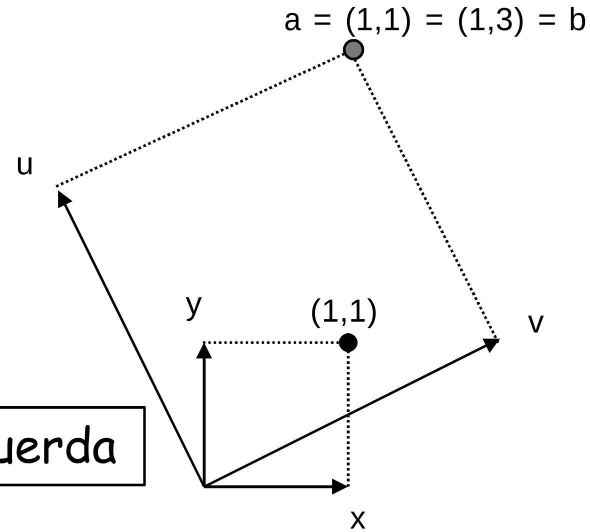
$$u = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$$

$$M = \begin{bmatrix} v & u \end{bmatrix}$$

$$b = Ma$$

ATENÇÃO: multiplicação à esquerda

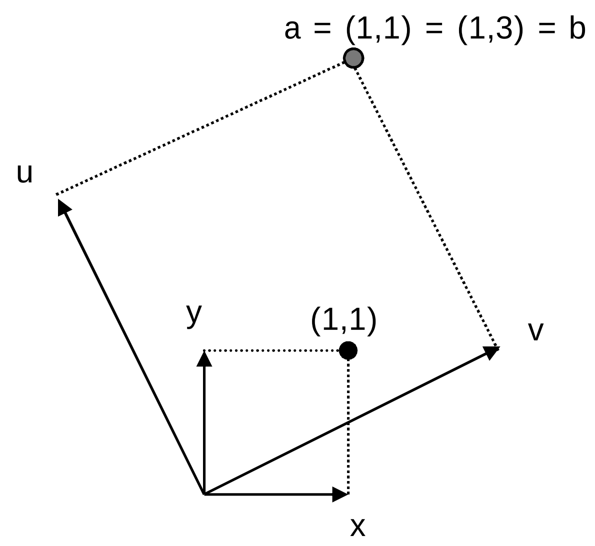
$$b = \begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$



Uma matriz pode ser vista como uma transformação do sistema de coordenadas, ou como uma transformação de pontos (ou objectos).

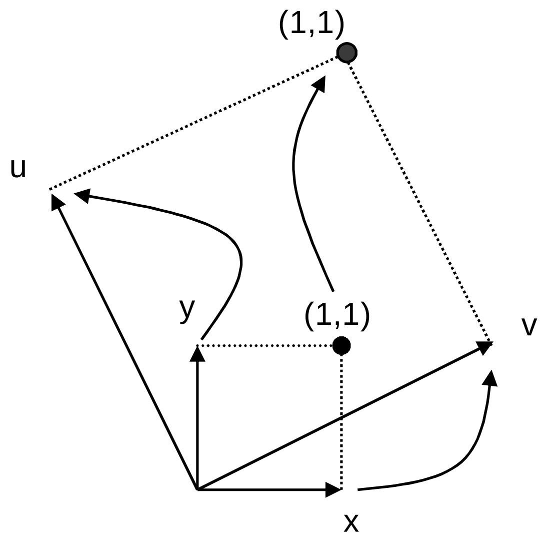
Transformações Geométricas

- Transformação de sistemas de coordenadas
  - Dado um ponto no sistema azul, a matriz M diz-nos quando vale esse ponto no sistema preto
  - Da mesma forma, dado um ponto no sistema preto, a matriz M^{-1} diz-nos quanto vale esse ponto no sistema azul
- Ou transformação de pontos no mesmo sistema de coordenadas
  - Dado um ponto no sistema preto, a matriz M transforma esse ponto num outro ponto preto: (1,1) é transformado em (1,3).



Transformações Geométricas

Vejamos o que acontece ao paralelograma formado pelos eixos de cada sistema

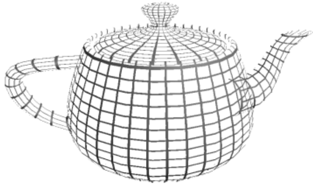


Transformações Geométricas

- Quais as vantagens de representar transformações de pontos através de matrizes?
  - Múltiplas transformações
    - $M_1\ M_2\ P = (M_1\ M_2)\ P$
    - $M_{12}\ P$, sendo $M_{12} = M_1\ M_2$
  - Notação Standard para todas as transformações
  - Transformação Inversa é definida pela matriz inversa

OpenGL: Transformações Geométricas

- Uma transformação geométrica é representada por uma matriz
- Aplicar a transformada a um ponto (ou conjunto de pontos) pode ser visto como aplicar essa transformada aquele ponto, transformando-o noutro ponto no mesmo sistema de eixos
- O OpenGL tem sempre "uma" transformada activa: todas as coordenadas são sempre transformadas (multiplicadas) por esta matriz - Inicialmente a transformada é a matriz Identidade
- Quando é especificada uma transformação ESTA fica ACTIVA e é aplicada a todos os pontos que forem entrados posteriormente
- A especificação de uma nova transformada é composta com a transformada anterior



Transformações Geométricas

- Escala

Para definir uma escala uniforme em todos os eixos definimos a seguinte matriz

$$P' = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{bmatrix} P$$

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

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

Matriz Inversa:

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

Transformações Geométricas

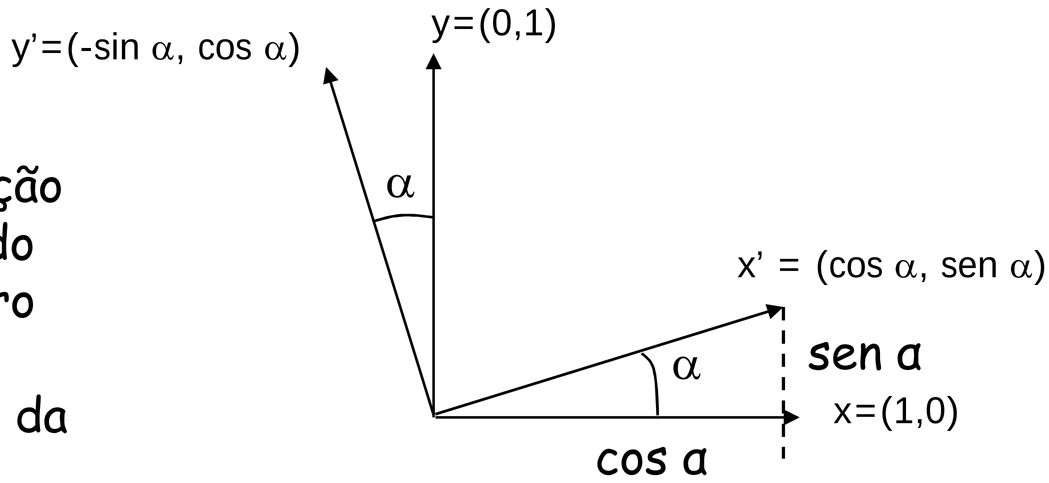
- Escala em OpenGL

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

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

Transformações Geométricas

- Rotação
  - Para exprimir uma rotação de um ângulo α utilizando matrizes, vamos primeiro definir um sistema de coordenadas resultante da rotação dos eixos por α.



$$M = \begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

Transformações Geométricas

- Rotação 3D em torno dos eixos
  - A rotação inversa é obtida pela inversa da matriz
  - a inversa de uma rotação é a transposta da matriz de rotação(*)

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

(*) para os puristas: isto é verdade porque a matriz de rotação é uma matriz ortogonal, isto é, define um sistema ortonormal

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

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

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

Transformações Geométricas

- Rotação em OpenGL

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

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

Transformações Geométricas

- Translação
  - A translação não pode ser expressa por um produto de uma matriz 3x3, mas sim por uma adição!
  - Sendo assim a execução de uma translação seguida de rotações ou escalas é definida da seguinte forma
    - $P' = MP + T$,
    - sendo M uma matriz invertível, e T uma translação.
  - Logo, aplicando novamente a sequência de operações acima definida ficaríamos com
    - $P'' = M'P' + T' = M'(MP + T) + T' = (M'M)P + M'T + T'$

Transformações Geométricas

- Desta forma seria necessário guardar os resultados parciais para operações posteriores

$$P'' = M'P' + T' = M'(\underline{M}P + \underline{T}) + T' =$$
$$=(\underline{M'M})P + \underline{M'T} + \underline{T'}$$

- Resumindo, utilizando matrizes 3x3 não conseguimos compor rotações, translações e escalas numa única matriz M

- A solução está na utilização de matrizes 4x4.

Transformações Geométricas

- Matrizes 4x4

$$F = \begin{bmatrix} M & T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} M_{11} & M_{12} & M_{13} & Tx \\ M_{21} & M_{22} & M_{23} & Ty \\ M_{31} & M_{32} & M_{33} & Tz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P' = MP + T = FP$$

Esta operação corresponde a:
translação seguida de uma rotação e/ou escala

Transformações Geométricas

- Matrizes 4x4 => Pontos com 4 coordenadas
- Pontos com coordenadas distintas podem representar o mesmo ponto 3D
- O ponto 3D é obtido dividindo as três primeiras coordenadas pela última coordenada.
- Os pontos são frequentemente pensados (representados) com w=1
- Para vectores w = 0, porquê? (tip: diferença de pontos)

$$P_4 = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

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

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

Transformações Geométricas

- Exemplo: sem rotação e translação [2, 1, 3]

P= [2, 2, 3]

F = [1 0 0 2]
[0 1 0 1]
[0 0 1 3]
[0 0 0 1]

P' = F [P 1] =
[1 0 0 2] [2] = [4]
[0 1 0 1] [2] = [3]
[0 0 1 3] [3] = [6]
[0 0 0 1] [1] = [1]

Transformações Geométricas

- Transformação inversa
  - rotação seguida de translação
  - duas maneiras diferentes de obter a matriz F^{-1}

$$P' = MP + T$$

$$P = M^{-1}P' - M^{-1}T$$

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

$$F^{-1} = \begin{bmatrix} M^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_3 & -T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} M^{-1} & -M^{-1}T \\ 0 & 1 \end{bmatrix}$$

Transformações Geométricas

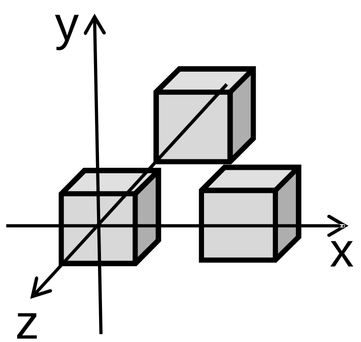
- Translação em OpenGL
  - `glTranslate{d,f}(x,y,z);`

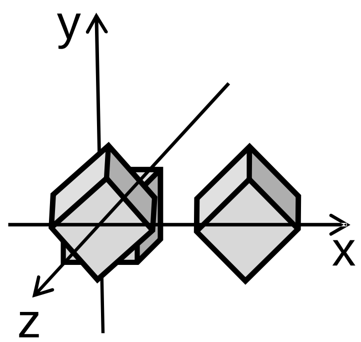
Transformações Geométricas

- Ordem das Transformações:
  pensando nas transformações como transformando as coordenadas dos objectos no sistema de eixos do mundo, estas ocorrem na ordem contrária à que aparecem no código

```
glRotatef (-45, 0., 0., 1.);
glTranslatef (3., 0., 0.);
glWireCube (1);
```

```
glTranslatef (3., 0., 0.);
glRotatef (-45, 0., 0., 1.);
glWireCube (1);
```

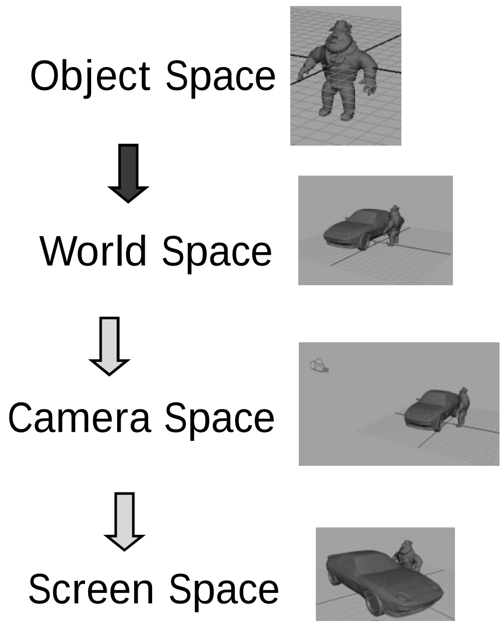




Transformações Geométricas

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

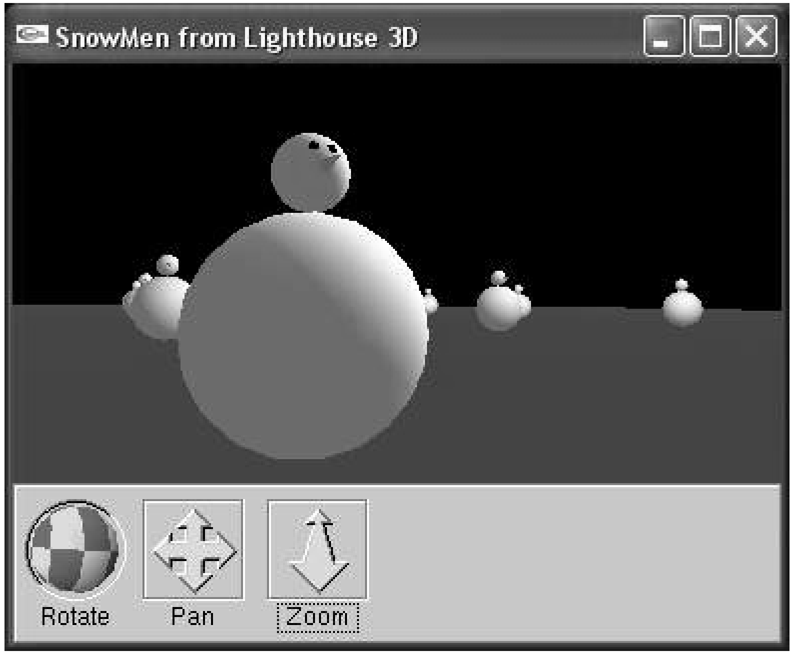
```
glScale{f,d}()
glTranslate{f,d}()
glRotate{f,d}()
```



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);
        glRotatef(90.0f,1.0f, 0.0f, 0.0f);
        glutSolidCone(0.08f,0.5f,10,2);
}
```

Modelar um boneco de neve
com esferas e um cone



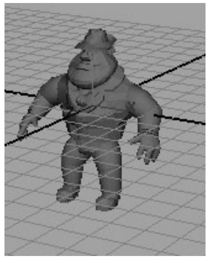
Transformações Geométricas

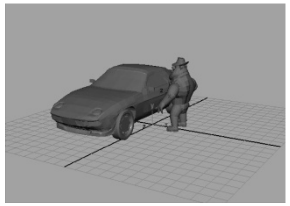
Object Space

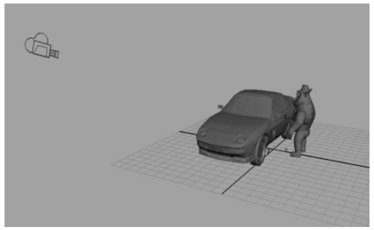
World Space

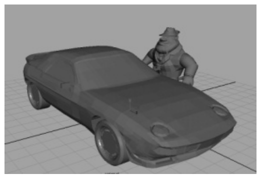
Camera Space

Screen Space









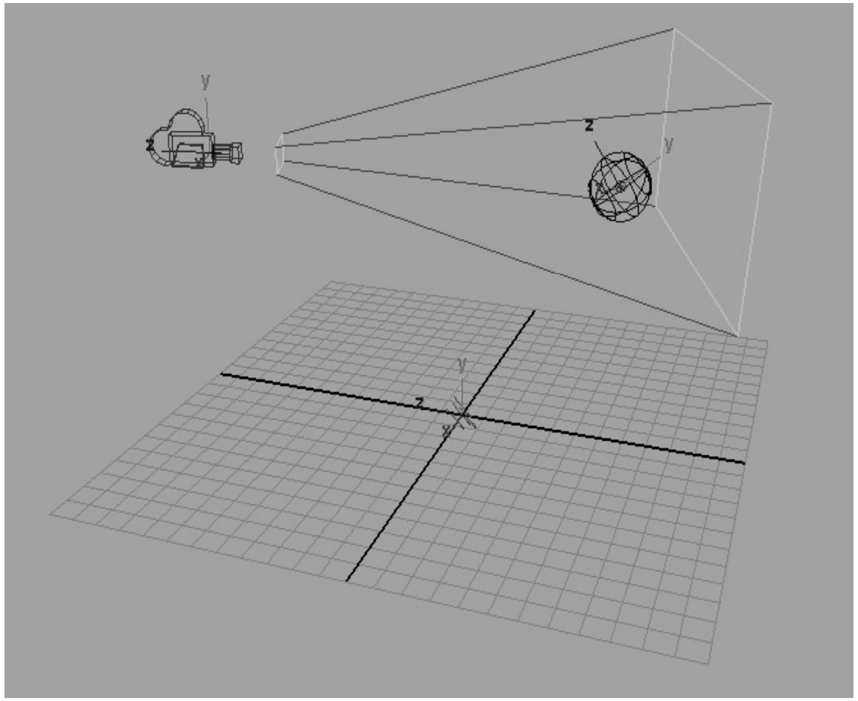
Transformações Geométricas

- 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
    (dir =-Z da câmara)
  - "este lado para cima"
    (up =Y da câmara)



Transformações Geométricas

- Operações sobre a câmara:
  - Translação da câmara para *posição*
  - Orientação da câmara de acordo com os vectores especificados
- Podemos facilmente especificar os eixos do sistema de coordenadas da câmara. Assumindo que os vectores fornecidos se encontram normalizados:

cz = -dir
cy = up
cx = cz x up (normalizar)

- Podemos então definir uma transformação linear que permita posicionar a câmara:

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

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

Transformações Geométricas

- A matriz F permite converter pontos do espaço da câmara para o espaço global.

- O que se pretende é exactamente o contrário, ou seja, pretende-se converter pontos do espaço global para o espaço da câmara.

- Solução: utilizar a transformação inversa!

Transformações Geométricas

- F^{-1} permite passar do espaço global para o espaço da câmara

$$F^{-1} = \begin{bmatrix} M^{-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_3 & -T \\ 0 & 1 \end{bmatrix}$$

$$M^{-1} = M^T = \begin{bmatrix} cx_1 & cx_2 & cx_3 \\ cy_1 & cy_2 & cy_3 \\ cz_1 & cz_2 & cz_3 \end{bmatrix}$$

Transformações Geométricas

- Posicionamento da câmara em OpenGL

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

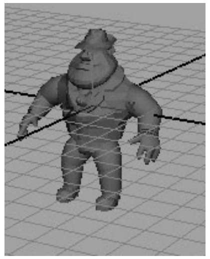
sendo:

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

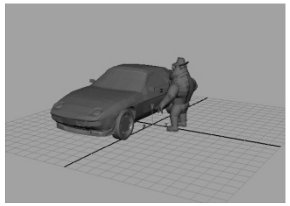
Nota para iniciantes:
a transformação associada à câmara deve ser a primeira no código, isto é, gluLookAt() deve aparecer logo após o glLoadIdentity() e antes de qualquer outra transformação!

Transformações Geométricas

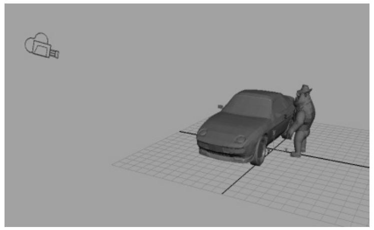
Object Space



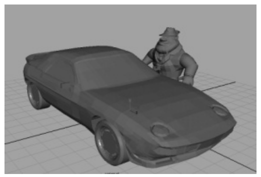
World Space



Camera Space

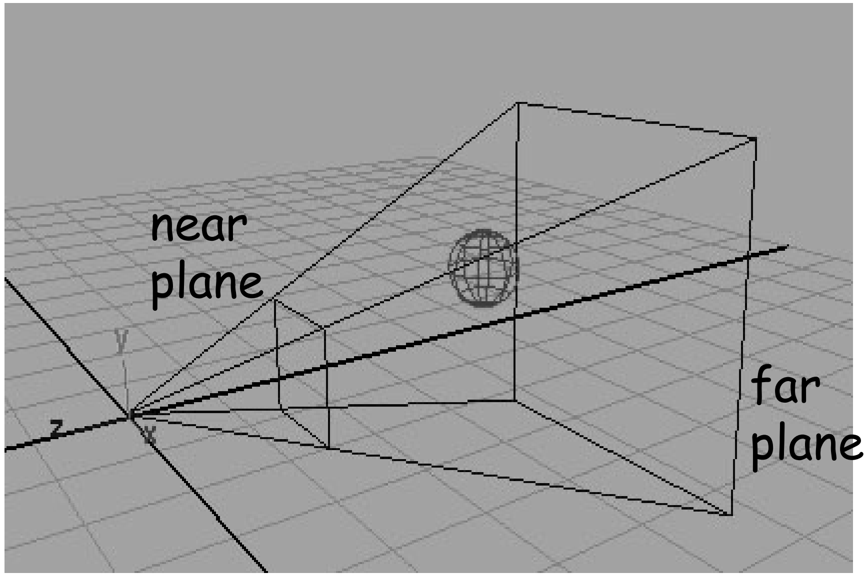


Screen Space



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 (*Px,Py,Pz*) (espaço câmara) no plano de projecção (screen space)

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

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

Transformações Geométricas

- Definição do Frustum em OpenGL

  – `glFrustum(left,right,bottom,top,near,far);`

  (*left*, *bottom*, *-near*) e (*right*, *top*, *-near*) especificam os pontos do *near clipping plane* que mapeiam nos cantos inferior esquerdo e superior direito da janela, assumindo que o observador está localizado em (0, 0, 0).

- Viewport (janela) em OpenGL

  – `glViewport(x,y,width,height);`

Transformações Geométricas

- O GLU fornece uma alternativa mais simpática para o *view frustrum*:

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

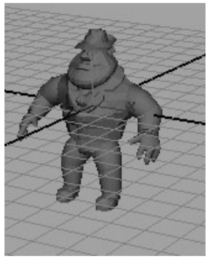
  - sendo
    - fy – ângulo de visão em y.
    - ratio – relação fx/fy

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

Transformações Geométricas

- Matrizes em OpenGL

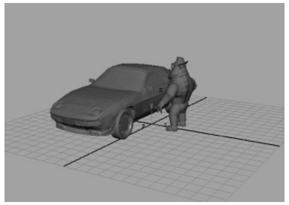
Object Space



⇩

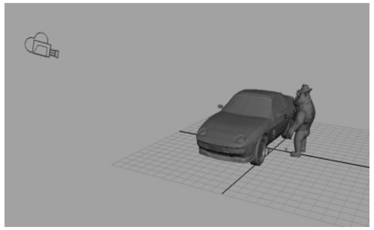
ModelView
`GL_MODELVIEW`

World Space



⇩

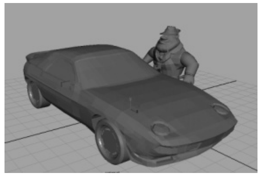
Camera Space



Projection
`GL_PROJECTION`

⇩

Screen Space

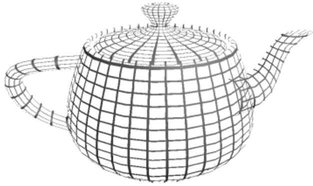


OpenGL

```
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 (ex., evento Reshape)



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 (ou Frame) Buffer
  - O OpenGL permite ter 2 buffers distintos.
  - Em cada instante visualiza-se um buffer e escreve-se no outro.
    (ou seja, a imagem que visualizamos em cada instante corresponde à imagem gerada pela execução anterior do renderScene())
  - No final da frame trocam-se os buffers.

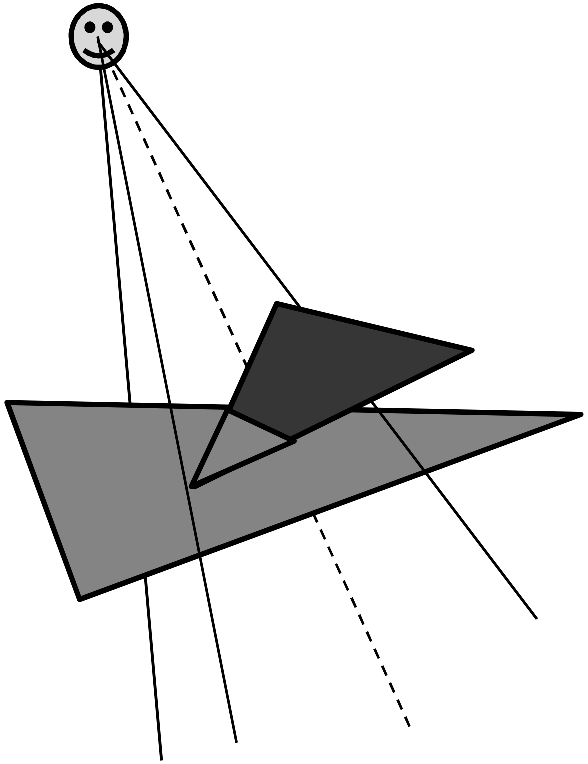
Buffers

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

Buffers

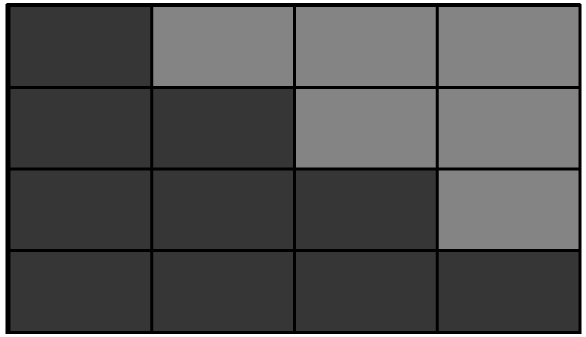
- Depth Buffer ou Z-Buffer
  - Buffer que armazena os valores de Z dos pixels que já foram desenhados
  - Permite assim criar uma imagem correcta sem ser necessário ordenar e dividir polígonos

Buffers



- Processa verde
- Processa vermelho

Frame buffer



Z buffer

24	25	25	25
22	22	23	23
16	16	16	18
12	11	11	10

Buffers

- Depth Buffer em OpenGL

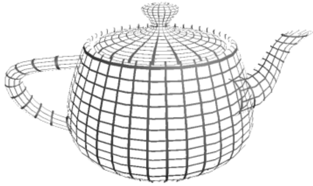
  - Na inicialização

    - `glutInitDisplayMode(GLUT_DEPTH | ... );`
    - `glEnable(GL_DEPTH_TEST);`

  - No início de cada frame
    - `glClear(GL_DEPTH_BUFFER_BIT | ...);`

Buffers

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



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