

---

---

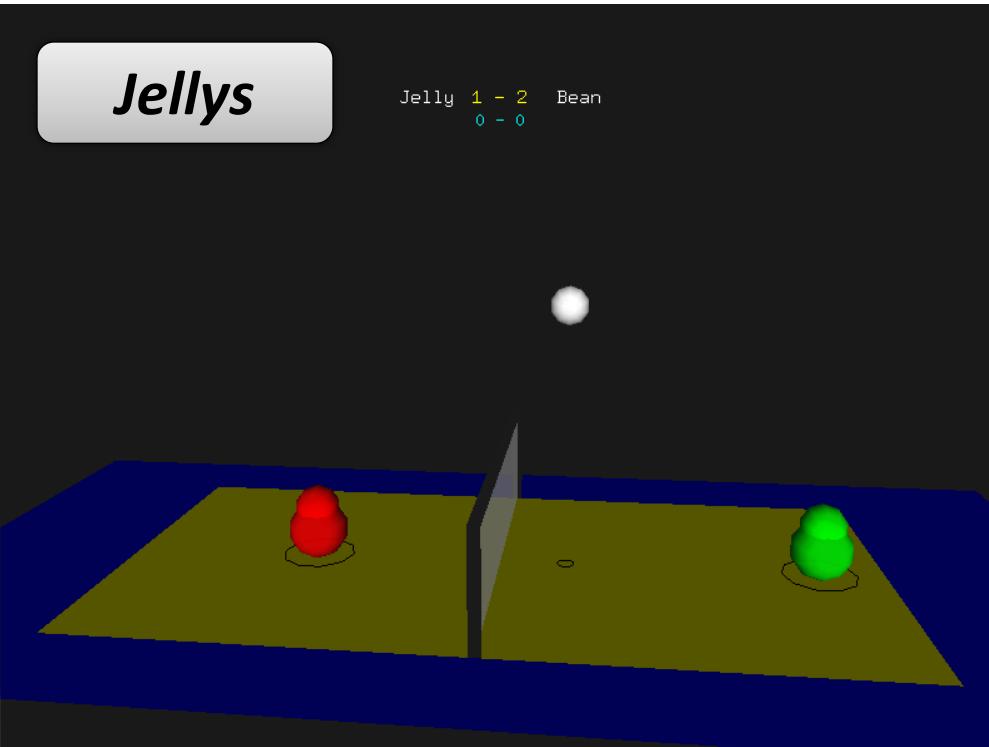
# *OpenGL*

## Computação Gráfica

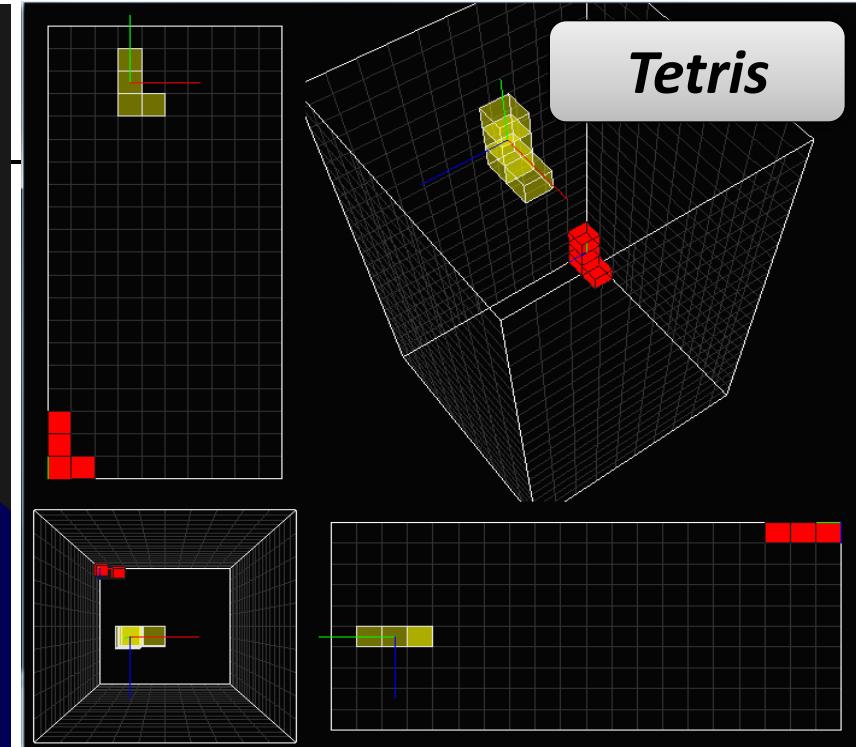


# Jellys

Jelly 1 - 2 Bean  
0 - 0



# Tetris



Gorilla A  
H: 101 V:12      <-----  
Power: 30

Pontos

0 - 0

Gorilla B  
H: 332 V:60  
Power: 26

# Gorillas

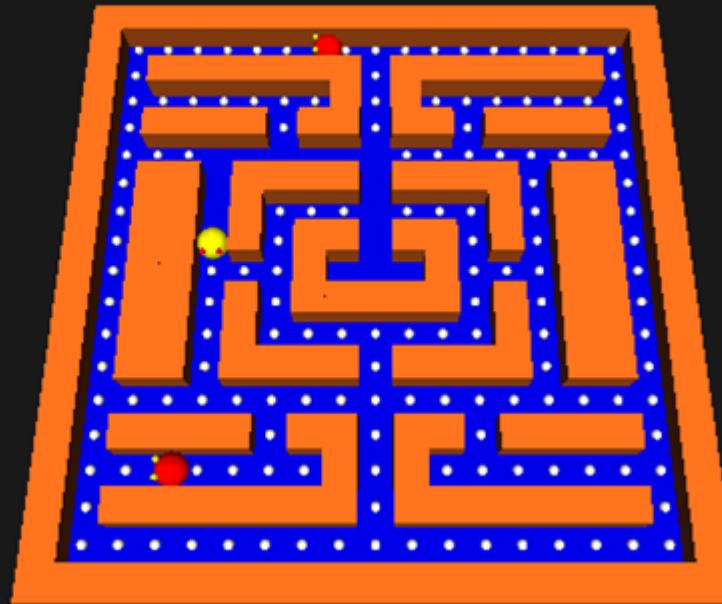
Camera: Gorilla View



Camera: Gorilla View

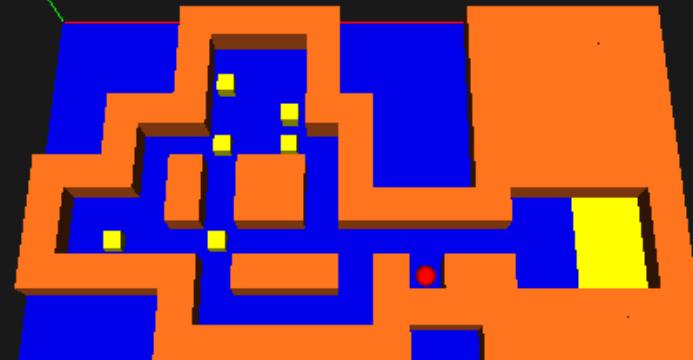
Eggs: 155

PacMan



Moves: 0  
Boxes: 0/6  
Current Level: 1 de 20

FPS:  
62 instant  
64 average



HELP:  
R -> Restart  
H -> Help  
C -> Toggle Camera

Sokoban

# Sumário

---

- Introdução ao OpenGL (capítulos 1 e 2 do *red book*)
  - O que é?
  - Sintaxe
  - Máquina de estados do OpenGL
  - *Rendering pipeline*
  - GLUT
  - Primitivas OpenGL
- Projeção, modelação e visualização
- Cor

# OpenGL – O que é?

---

- É uma API para acesso à placa gráfica
  - API C
  - Contém cerca de 200 comandos (métodos) distintos
  - Independente do *hardware* (placa gráfica)
  - Não dependente de um sistema de janelas específico, nem SO

# OpenGL – Desenvolvimento...

---

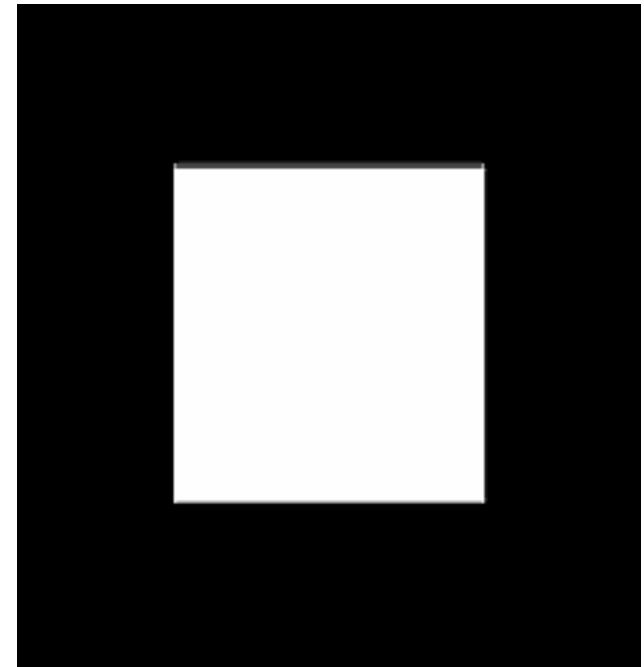
- “Because you can do so many things with OpenGL graphics system, an OpenGL program can be complicated. However, the basic structure of a useful program can be simple...
  - OpenGL Programming Guide (livro de referência desta unidade curricular)

# Um exemplo muitoooo simples...\*

---

```
#include <whateverYouNeed.h>

void main() {
    InitializeAWindowPlease();
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_POLYGON);
        glVertex3f(0.25, 0.25, 0.0);
        glVertex3f(0.75, 0.25, 0.0);
        glVertex3f(0.75, 0.75, 0.0);
        glVertex3f(0.25, 0.75, 0.0);
    glEnd();
    glFlush();
    UpdateTheWindowAndCheckForEvents();
}
```



**Atenção:** Este programa é demasiado simples... \*

- E se redimensionarmos a janela?
- Qual é o sistema de coordenadas?

# OpenGL: Sintaxe

---

- Todos os métodos da API OpenGL são precedidos por **gl**
  - **glClearColor**
  - **glVertex3f**
  - **glOrtho**
- As constantes também são precedidas por **GL**
  - **GL\_COLOR\_BUFFER\_BIT**
  - **GL\_POLYGON**
  - **GL\_SMOOTH**
  - **GL\_LIGHT0**
- Alguns métodos têm sufixos que definem o tipo de dados dos parâmetros
  - **3f**, usado em **glColor3f** ou **glVertex3f**
  - **3f** indica que o método tem três parâmetros do tipo *float*
  - **Nota:** Não esquecer que a API do OpenGL é C (*não existe overload de métodos*)

# Relação entre sufixos e tipos

sufixo	Tipo de dados	Tipo 'C'	Tipo OpenGL
<b>b</b>	inteiro (8-bit)	signed char	GLbyte
<b>s</b>	inteiro (16-bit)	short	GLshort
<b>i</b>	inteiro (32-bit)	int ou long	GLint, GLsizei
<b>f</b>	vírgula flutuante (32-bit)	float	GLfloat, GLclamp
<b>d</b>	vírgula flutuante (64-bit)	double	GLdouble, GLclampd
<b>ub</b>	inteiro sem sinal (8-bit)	unsigned char	GLubyte, GLboolean
<b>us</b>	inteiro sem sinal (16-bit)	unsigned short	GLushort
<b>ui</b>	inteiro sem sinal (32-bit)	unsigned int ou unsigned long	GLuint, GLenum, GLbitfield

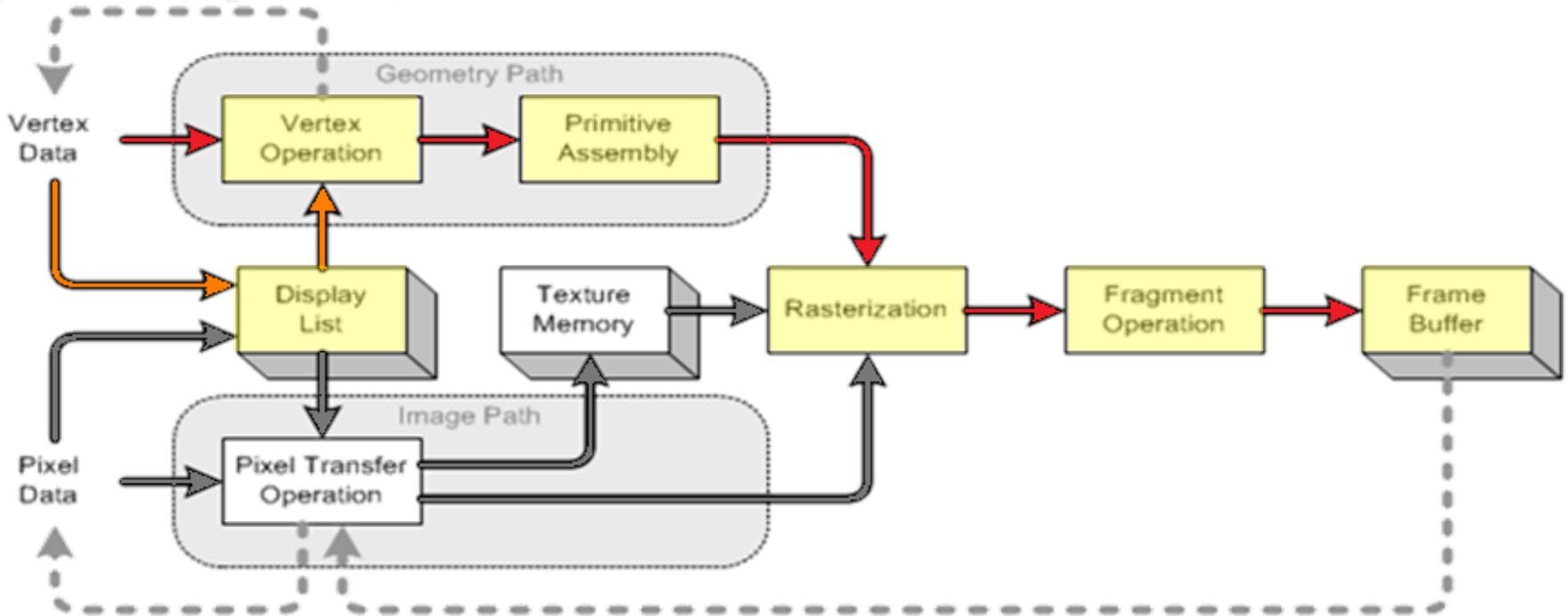
- Alguns métodos terminam ainda com um **v**
  - Indica que recebe um apontador para o array com os valores
  - Ex:
    - `GLfloat colorArray[] = { 1.0f, 1.0f, 0.0f };`
    - `glColor3fv(colorArray);`

# OpenGL: Máquina de estados

---

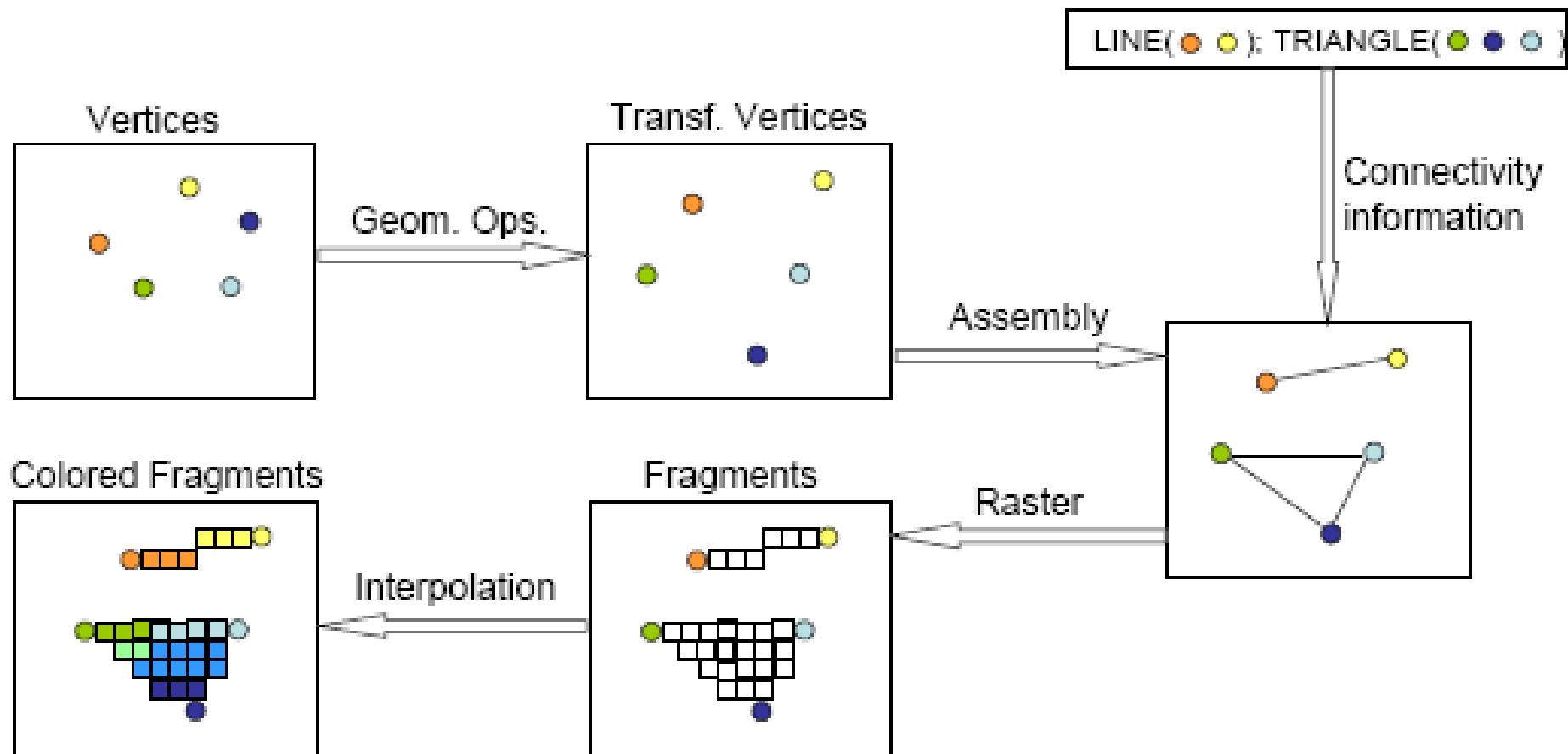
- O OpenGL é uma máquina de estados
    - A cor utilizada para desenhar uma linha é a **cor actual**
    - A textura usada no mapeamento de um triângulo é a **textura actual**
    - A espessura da linha é aquela que está definida como **espessura actual**
    - Etc...
  - O estado da máquina de estados é definido pelo valor das suas variáveis
  - Cada variável tem um valor definido por omissão
  - É possível, em qualquer momento, obter o valor das variáveis
    - Usam-se os métodos `glGet*`
      - Ex: `glGetIntegerv(...)`   `glGetBooleanv(...)`
  - As variáveis de estado activáveis são activadas/inactivadas com os métodos `glEnable(...)` e `glDisable(...)`
    - Ex: `glEnable(GL_LIGHT0)` ou `glEnable(GL_TEXTURES)`
-

# OpenGL: rendering pipeline



- Vertex Operation:** Transformação de visualização dos vértices. Utiliza-se a matriz GL\_MODELVIEW.
- Display List:** Grupo de comandos OpenGL compilados. Aumenta o desempenho da aplicação.
- Primitive Assembly:** Distorção de perspectiva. Clipping. Transformação *window to viewport*.
- Rasterization:** Conversão da informação geométrica em fragmentos (cor, profundidade, ...)
- Fragment Operation:** Operações sobre fragmentos, tais como: textura ou nevoeiro.
- Frame Buffer:** Representa a matriz de *pixels* (fragmentos tornam-se *pixels*).

# OpenGL: rendering pipeline (outra vista)



- **Fonte:** DI-UM - António Ramires Fernandes - Multimédia  
<http://sim.di.uminho.pt/disciplinas/mm/0607/t10/t10.pdf>

# GLUT: OpenGL Utility Toolkit

---

- Sistema de janelas independente do sistema operativo
  - Desenvolvido por Mark Kilgard
- Disponibiliza métodos para desenho de primitivas “complexas” (vs OpenGL)
  - Cubo, esfera, *donut (torus)*, bule de chá (*teapot*)
- Desenvolvimento
  - Os métodos desta API têm o prefixo **glut**
  - É necessário incluir o *header file <GL/glut.h>* e a biblioteca **glut32.lib**
  - Simplifica a criação de janelas, acesso ao teclado, etc...
  - Notificação de eventos feita à custa de registo de funções de *callback*
    - Teclado, Rato
    - Redimensionamento da janela
    - Temporizador (*timer*)
    - Redesenho da cena (*render*)

# GLUT: Gestão de janelas

---

- **glutInit(int \*argc, char \*\*argv)**
  - Inicia o GLUT e processa os valores da linha de comandos (*X Windows System*)
  - Tem que ser o primeiro método a ser chamado
- **glutInitDisplayMode(unsigned int mode)**
  - Define o modelo de cor (indexado ou RGB)
  - Define quais os *buffers* a usar (profundidade, *double-buffering*, *stencil*, etc...)
- **glutInitWindowPosition(int x, int y)**
  - Define a posição inicial da janela relativamente ao canto superior esquerdo do ecrã
- **glutInitWindowSize(int width, int height)**
  - Define o tamanho inicial da janela, em *pixels*
- **int glutCreateWindow(char \*string)**
  - Cria a janela com as características definidas e com o nome indicado pela *string*
  - Não mostra a janela. Ver método *glutMainLoop*.

# GLUT: *Callback* de desenho

---

- **glutDisplayFunc( void (\*func)(void) )**
  - A função de *callback* mais importante do GLUT (é obrigatória)
  - Chamada quando for necessário redesenhar o conteúdo da janela
  - A função a registar, **func**, não recebe parâmetros (*void*) nem retorna valor (*void*)
- **Atenção:**
  - Se o programa alterar o conteúdo da cena, deve chamar a função **glutPostRedisplay()** para que a função de desenho seja chamada

# Exemplo: Hello GLUT\*\*

```
#include <GL/glut.h> /* GLUT definitions (includes OpenGL definitions) */
void display(void) {
    // Clear all pixels with background color
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    // Draw white polygon (rectangle) with
    // corners at (0.25, 0.25, 0.0)
    // and (0.75, 0.75, 0.0)
    glBegin(GL_POLYGON);
        glVertex3f (0.25, 0.25, 0.0);
        glVertex3f (0.75, 0.25, 0.0);
        glVertex3f (0.75, 0.75, 0.0);
        glVertex3f (0.25, 0.75, 0.0);
    glEnd();
    // Don't wait!
    // Start processing buffered routines
    glFlush ();
}

void init(void) {
    /* set background color to black */
    glClearColor (0.0, 0.0, 0.0, 0.0);
}
```

```
/* Declare initial window size, position, and display
mode (single buffer and RGBA). Open window with
"Hello GLUT" in its title bar. Call initialization
routines. Register callback function to display
graphics. Enter main loop and process events. */
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (250, 250);
    glutInitWindowPosition (100, 100);
    glutCreateWindow ("Hello GLUT");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0; /* ISO C requires main to return int. */
}
```

# GLUT: Eventos de entrada\*

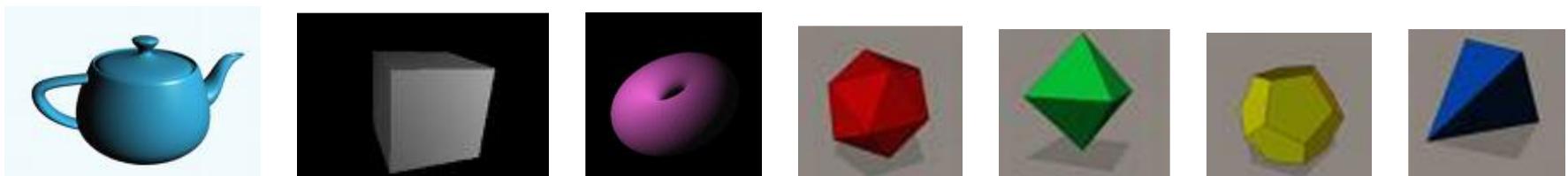
---

- O GLUT gera eventos para os seguintes fenómenos
  - Redimensionamento da janela
  - Teclado, Rato
- **glutReshapeFunc( void (\*func)(int w, int h) )**
  - A janela foi redimensionada e passa a ter w *pixels* de largura e h de altura.
- **glutKeyboardFunc( void (\*func)(unsigned char key, int x, int y) )**
- **glutKeyboardUpFunc ( void (\*func)(unsigned char key, int x, int y) )**
  - Indicam que foi premida ou solta, respectivamente, uma tecla (ASCII) do teclado
  - x e y indicam a posição do rato relativa à janela (em pixels)
- **glutMouseFunc( void (\*func)(int button, int state, int x, int y) )**
  - Indica que foi permitido ou solto (state) um botão (button) do rato
- **glutMotionFunc( void (\*func)(int x, int y) )**
  - Indica que o rato se deslocou para a posição (x, y) da janela, em pixels.
  - Apenas é chamada quando se está a premir um botão do rato

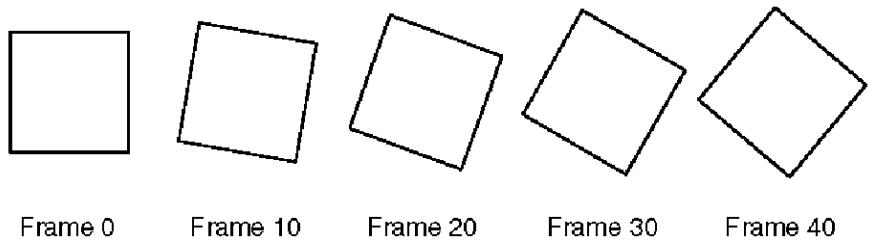
# GLUT: Idle e objectos tridimensionais

---

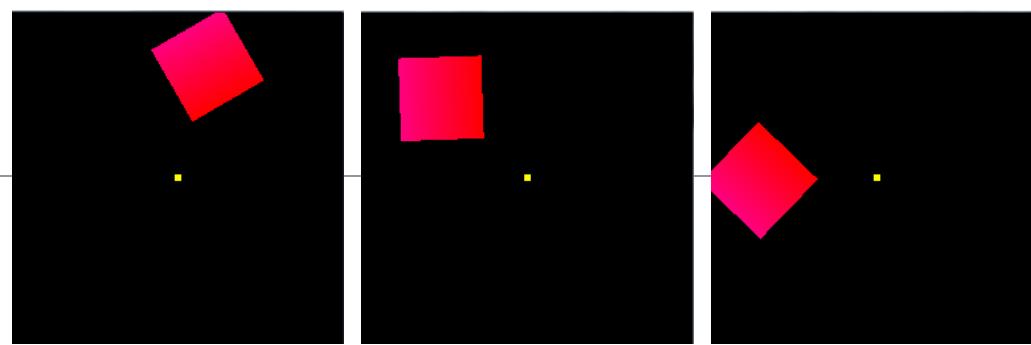
- É possível registar um método a ser chamado quando nenhum evento estiver a ocorrer (*idle time*)
  - `glutIdleFunc( void (*func)(void) )`
- Objectos tridimensionais
  - O OpenGL apenas fornece suporte para desenho de linhas e polígonos
  - O GLUT simplifica o desenho de alguns objectos
    - *Cone, icosahedron, teapot, cube, octahedron, tetrahedron, dodecahedron, sphere e torus*
  - É possível desenhar estes objectos em modelo de arames (*wireframe*) ou sólidos
    - Ex: `glutWireTeapot(GLdouble size)` ou `glutSolidTeapot(GLdouble size)`
  - Todos estes objectos são desenhados no centro do mundo (WCS)



# GLUT: Animação



- É necessário gerar ~25 imagens por segundo para ter noção de movimento
- Para definir cenas com movimento, é necessário um pouco mais...
- Para evitar o efeito de *flickering* deve-se usar uma técnica de *double-buffering*
  - Em vez de se desenhar sempre para a mesma matriz de pixéis, usam-se duas, uma na qual se desenha a cena e outra que é usada pela placa gráfica. No final de cada fase de desenho, trocam-se os papéis das matrizes
  - Usa-se a flag `GL_DOUBLE` no método `glutInitDisplayMode(...)`
  - Usa-se o método `glutSwapBuffers()` no final da fase de desenho para trocar os papéis das matrizes
- Para alterar o estado da aplicação (animação) deve-se usar
  - O registo da função *idle* (não garante chamada à função!)
  - O registo de um *timer* (*callback* de `glutTimerFunc`)



# Exemplo

---

Animação com quadrado a rodar (*idle func*)

# GLUT: Animação de quadrado (1/2)

```
#include <GL/glut.h>
#include <stdlib.h>

// Square angle
static GLfloat spin = 0.0;

/* Init OpenGL state
   (background color) */
void initGL() {
    glClearColor(0.0, 0.0, 0.0, 0.0);
}
```

Este exemplo implementa a animação através do registo da função `spinUpdate` no evento de *idle*.

```
/* Request double buffer display mode.
 * Register mouse callback functions */
int main(int argc, char** argv) {
    /* initialize windows */
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(250, 250);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Spinning shape - V1");
    /* initialize openGL */
    initGL();
    /* callbacks */
    glutDisplayFunc(display); /* desenho */
    glutMouseFunc(mouse); /* Eventos: rato */
    glutIdleFunc(spinUpdate); /* idle */
    /* app main loop */
    glutMainLoop();
    return 0;
}
```

# GLUT: Animação de quadrado (2/2)

```
/* redisplay whole scene */
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(spin, 0.0, 0.0, 1.0);

    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex3f (0.25, 0.25, 0.0);
    glVertex3f (0.75, 0.25, 0.0);

    glColor3f(1.0, 0.3, 0.0);
    glVertex3f (0.75, 0.75, 0.0);
    glVertex3f (0.25, 0.75, 0.0);
    glEnd();

    glutSwapBuffers();
}
```

```
/* update model and redisplay scene */
void spinUpdate(void) {
    spin = spin + 2.0;
    if (spin > 360.0)
        spin = spin - 360.0;
    glutPostRedisplay();
}
```

---

# Exercício

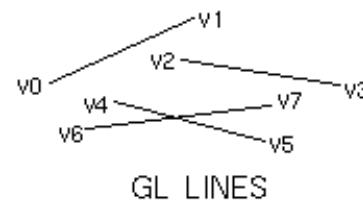
---

Animação com quadrado a rodar (*timer func*)

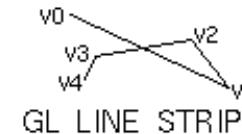


# Primitivas geométricas em OpenGL

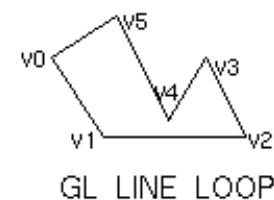
```
glBegin(<primitive>);  
    glVertex3f(...);  
    glColor3f(...);  
    glVertex3f(...);  
    ...  
    glColor3f(...);  
    glVertex3f(...);  
glEnd();
```



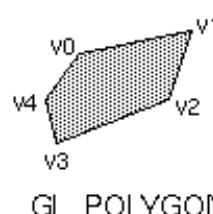
GL\_LINES



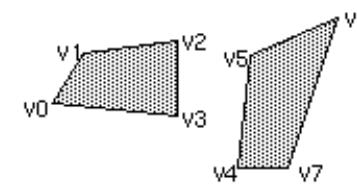
GL\_LINE\_STRIP



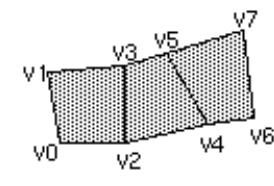
GL\_LINE\_LOOP



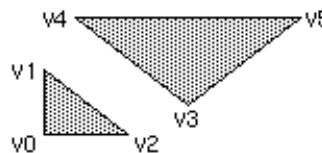
GL\_POLYGON



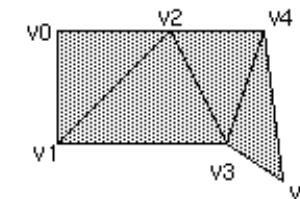
GL\_QUADS



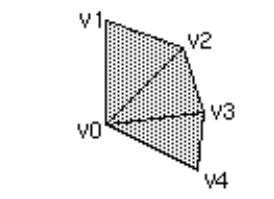
GL\_QUAD\_STRIP



GL\_TRIANGLES



GL\_TRIANGLE\_STRIP



GL\_TRIANGLE\_FAN

`<primitive>` :=

<code>GL_POINTS</code>	<code> </code>	<code>GL_POLYGON</code>
<code>GL_LINES</code>	<code> </code>	<code>GL_LINE_STRIP</code>
<code>GL_TRIANGLES</code>	<code> </code>	<code>GL_TRIANGLE_STRIP</code>
<code>GL_QUADS</code>	<code> </code>	<code>GL_QUAD_STRIP</code>

<code>GL_LINES</code>	<code> </code>	<code>GL_LINE_STRIP</code>
<code>GL_TRIANGLES</code>	<code> </code>	<code>GL_TRIANGLE_STRIP</code>
<code>GL_QUADS</code>	<code> </code>	<code>GL_QUAD_STRIP</code>

|

<code>GL_TRIANGLES</code>	<code> </code>	<code>GL_TRIANGLE_FAN</code>
---------------------------	----------------	------------------------------

• `v4`  
• `v0` • `v3`  
• `v1` • `v2`  
• `v0` • `v3`  
• `v1` • `v2`  
• `v0` • `v3`

GL\_POINTS

---

# Demo

---

## Primitivas OpenGL



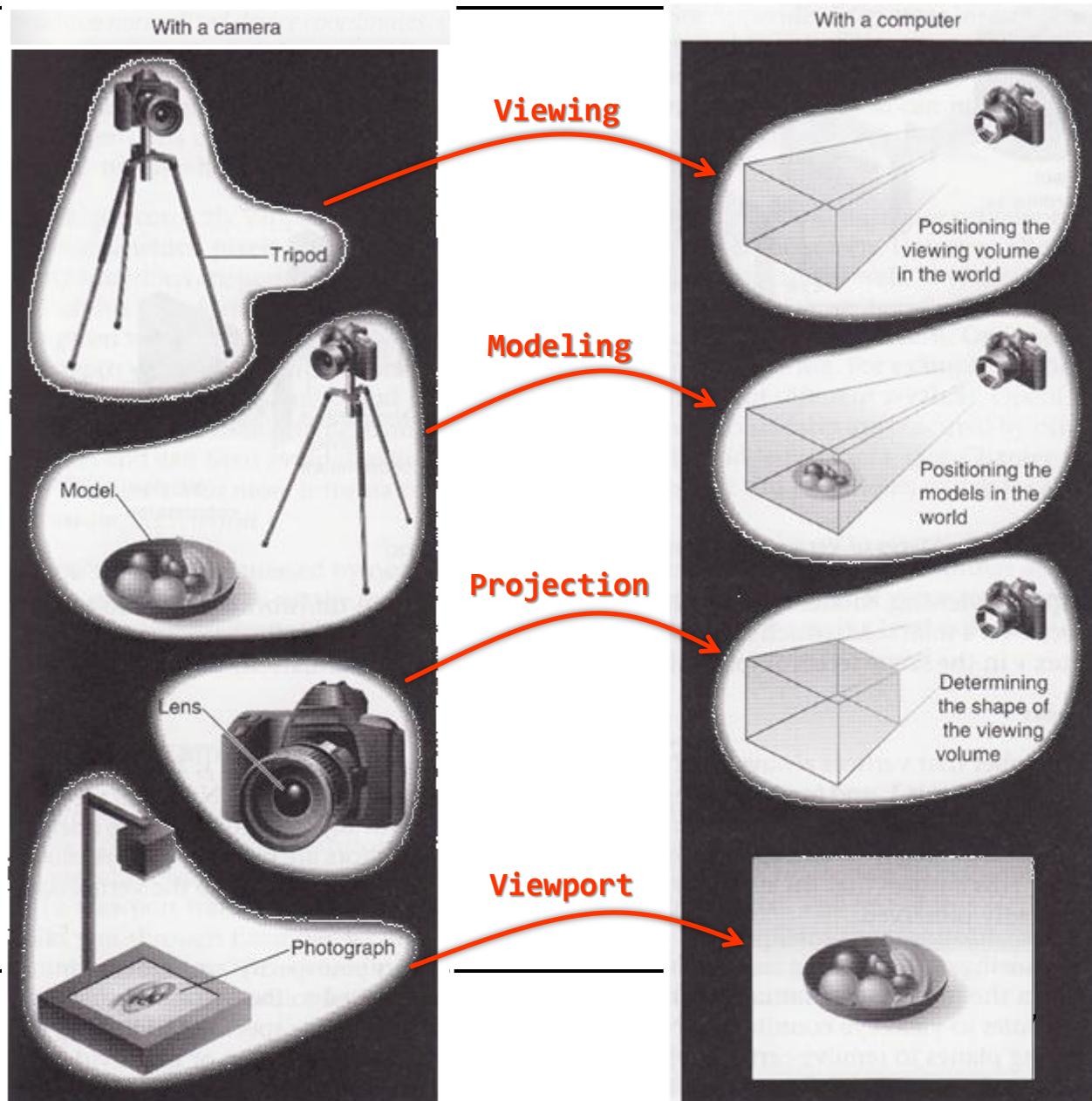
# Sumário

---

- Introdução ao OpenGL
- Projecção, modelação e visualização (*capítulo 3 do red book*)
  - Câmara Virtual
  - Transformação de modelação e visualização
  - Transformação de projecção
  - Transformação de *viewport*
  - Composição de transformações
- Cor

# Analogia com a câmara virtual (1)

- Passos necessário para tirar uma fotografia:
  - [1] Posicionar a câmara
  - [2] Ajustar a cena a ser fotografada
  - [3] Definir a forma como vamos ver a cena
    - Ex: *zoom*
  - [4] Revelar a fotografia com determinado formato
    - Ex: 10x15 ou 13x18

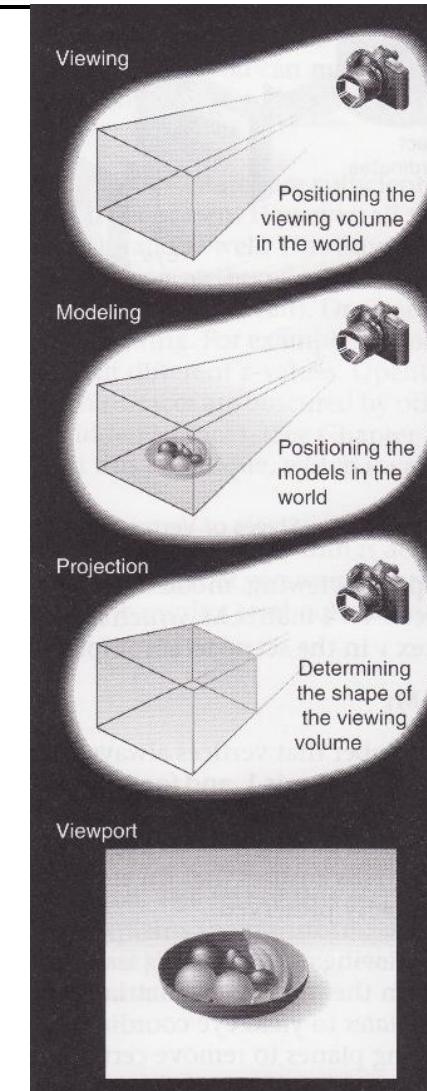


# Analogia com a câmara virtual (2)

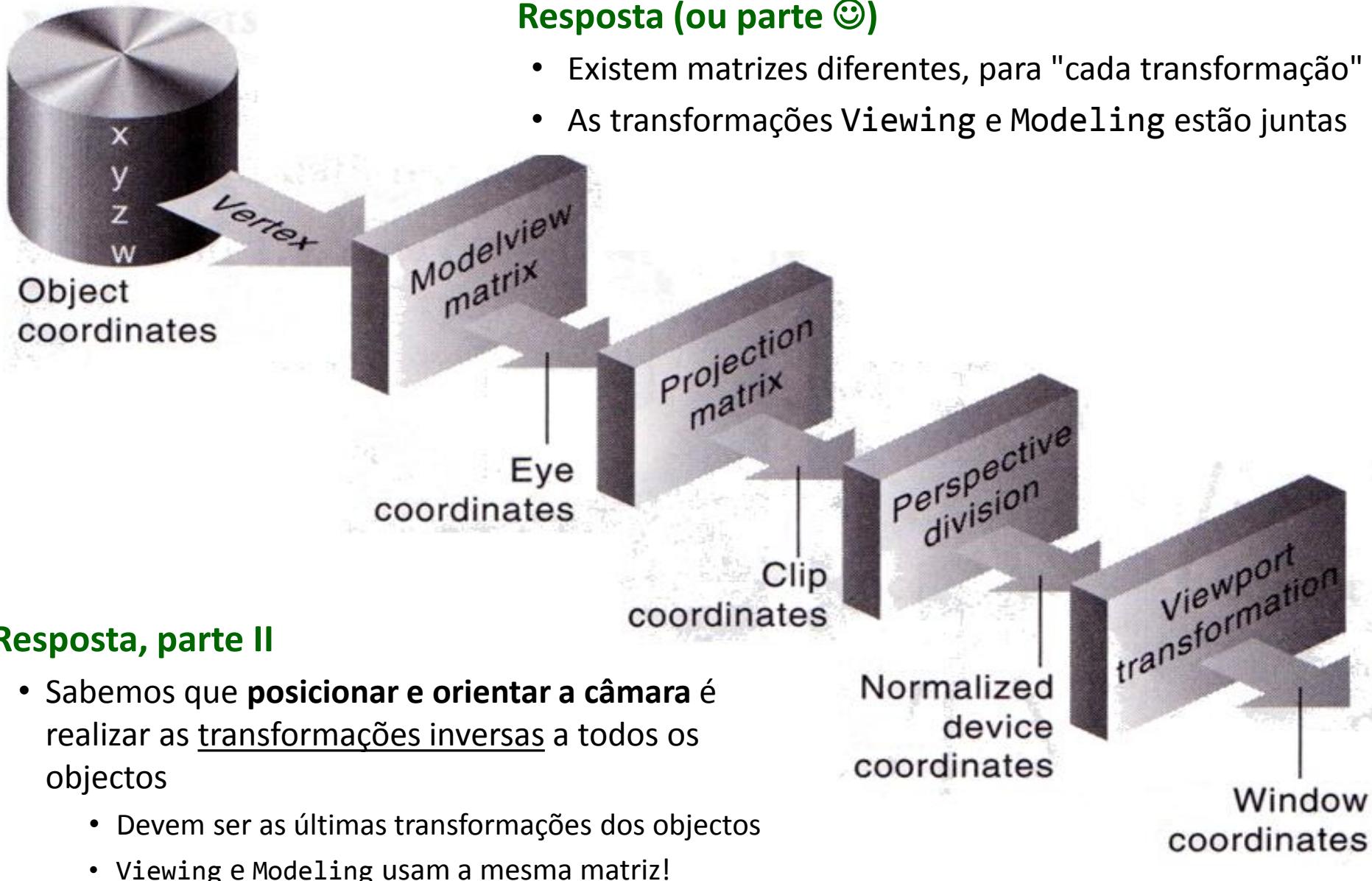
- **Atenção**

- A ordem indicada no slide anterior **não é a ordem** pela qual são aplicadas as transformações aos vértices dos objectos
- As transformações de Viewing têm que preceder as de Modeling
- Por seu lado, as transformações de Projection e Viewport podem ser definidas em qualquer altura, antes do *render*

Porquê ?



# OpenGL: Transformação de vértices



# Transformações de Viewing (Câmara)

---

- **Objectivo:**
  - Definir a posição e orientação **da câmara** no mundo
- **Soluções:**
  - Aplicar as transformações inversas às da câmara  
ou
  - Usar o método **void gluLookAt(eye, center, up)**, da GLU<sup>(1)</sup>
    - *eye, center* e *up* são conjuntos de três valores e vírgula flutuante

- **Exemplo:**

```
gluLookAt( 10.0, 10.0, 0.0, /* eye: posição da câmara */  
           0.0, 0.0, 0.0, /* center: local para onde está a olhar */  
           0.0, 1.0, 0.0 /* up: rotação da câmara em torno do */  
); /* vector direcção (center-eye)  
*/
```

---

# Transformações de Modeling (1)

---

- **Objectivo:**
  - Definir a posição e orientação **dos objectos** no mundo
- **Transformações/comando disponíveis:**
  - Translação
  - Rotação
  - Escala
  - *Transformação livre*
- **Exemplo:**

```
glTranslate3f(0.0f, 5.0f, 0.0f);  
glScale3f(1.0f, 1.0f, 2.0f);  
glRotate3f(45, 0.0f, 1.0f, 0.0f);  
glutSolidTeapot(1);
```



Como funciona a composição?  
Qual é a ordem de aplicação  
de transformações em OpenGL ?

# Transformações de Modeling (2)

---

- **Funcionamento:**
  - Cada comando (transformação) tem uma matriz que o representa
  - **Ex:**  
`glTranslate3f(1.0f, 2.0f, 4.0f);` → 
$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
  - Ao executar uma transformação estamos a multiplicar a matriz que a representa pela matriz Modelview<sup>1</sup> e guardar o resultado novamente na matriz ModelView<sup>1</sup>

$$ModelView = ModelView * T(1,2,4)$$

**Q:** Qual deve ser o valor inicial da **ModelView**?

**R:** Matriz Identidade

**Q:** Como definir esse valor?

**R:** Com comandos OpenGL ☺...

# Transformações de Modeling (3)

---

- Para definir a matriz identidade utiliza-se o comando

```
glLoadIdentity();
```

- Será que basta apenas este comando?
  - Não!
- Em que matriz é colocada o valor (identidade)?
  - Naquela que é a **matriz actual** da máquina de estados do OpenGL
- A matriz actual é definida com o comando

```
glMatrixMode( <matrixID> );
```

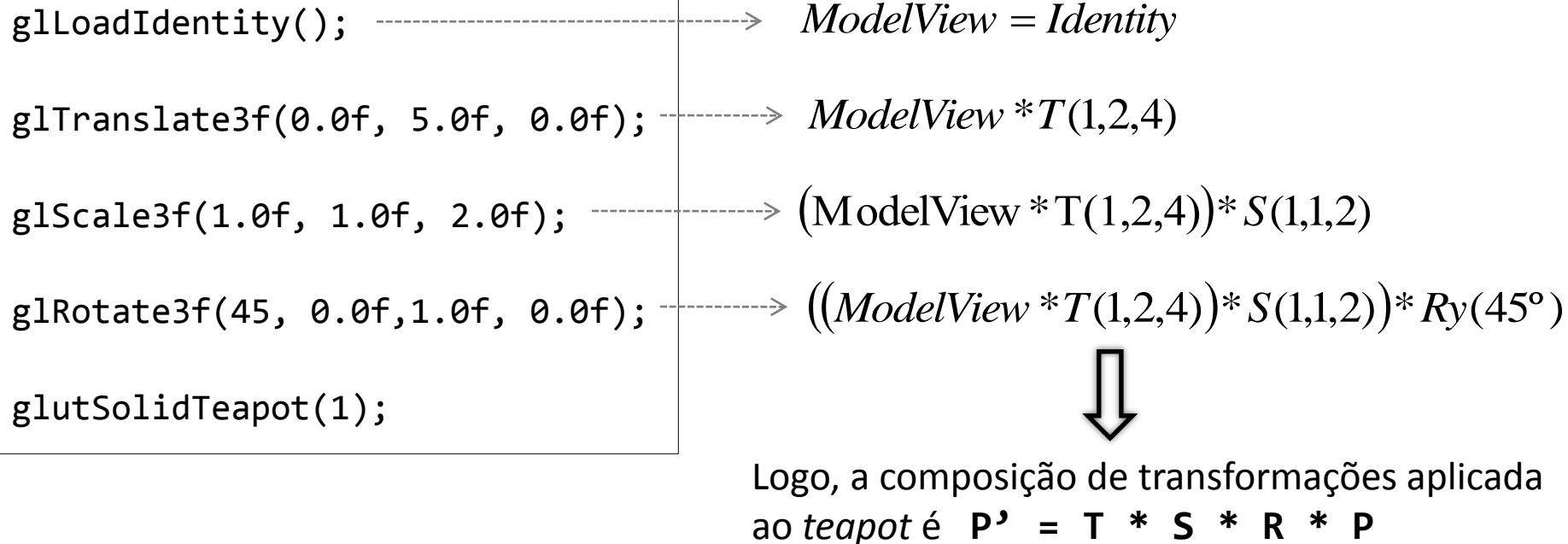
- Para as transformações de modelação e visualização usa-se a matriz

**GL\_MODELVIEW**

---

# Transformações de Modeling (4)

- Como funciona a composição de transformações em OpenGL?!
- Sabe-se que cada comando representa uma matriz
  - Essa matriz é multiplicada pelo conteúdo da matriz GL\_MODELVIEW
- Ao realizar várias transformações teremos, por exemplo:



# Transformações de Modeling (5)

---

- Outros comandos OpenGL
  - Definir explicitamente os valores da matriz 4x4

**glLoadMatrixf(const float \*m);**

- Multiplicar determinada matriz pela matriz actual:

**glMultMatrixf(const float \*m);**

- M é um *array* de 16 elementos que representam a matriz

$$m = [m_1, m_2, \dots, m_{16}] \Leftrightarrow \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

---

# Demo

---

## Mostrar funcionamento da matriz ModelView

- Usar método **gluLookAt(...);**
- Usar transformações para obter o mesmo resultado

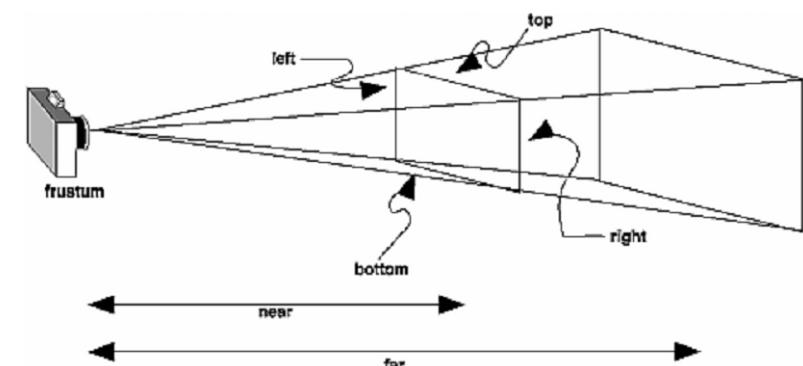
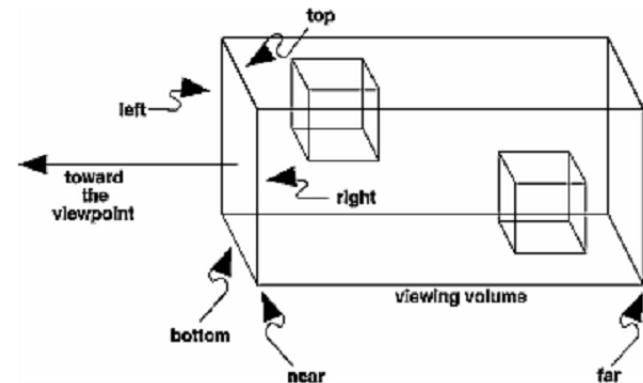


# Transformação de Projection

- **Objectivo:**
  - Define o tipo de projecção a aplicar à cena

- **Projeções disponíveis:**
  - Paralela (ortográfica)
  - Perspectiva

- **Comandos OpenGL/GLU:**
  - `glOrtho` ou `gluOrtho2D`
  - `glFrustum` ou `gluPerspective`
- **Atenção:**
  - Estes comandos alteram a matriz actual!
  - Antes de os usar é necessário alterar para a matriz **GL\_PROJECTION**
  - Não esquecer de “limpar” o conteúdo da matriz (`glLoadIdentity`)



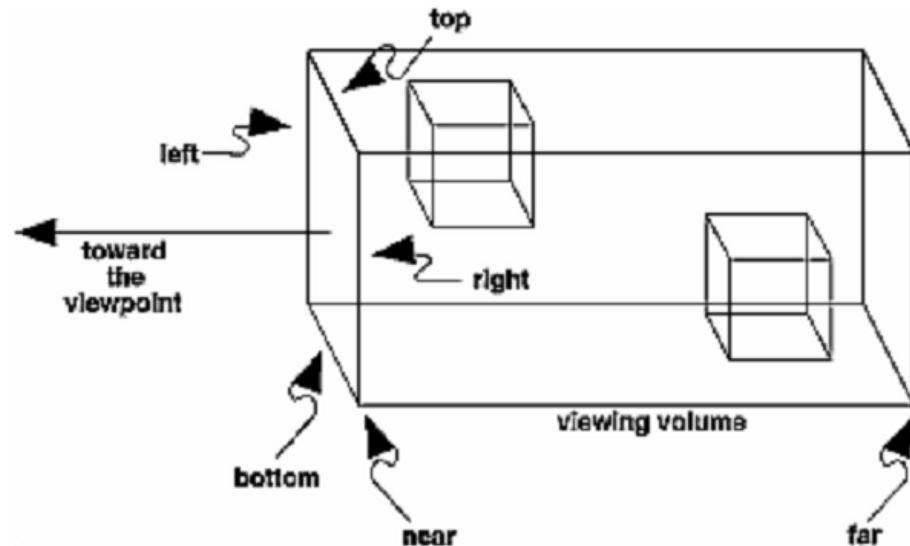
# Transformação de Projection (1)

---

- Valores por omissão, no OpenGL, para a “câmara” e projecção
- “Câmara”
  - Como o valor inicial da matriz ModelView é a matriz identidade, **não existe nenhuma transformação aplicada à câmara**
  - Está posicionada na origem do mundo a “olhar” para Z negativo
  - “Vê” o X a crescer para a direita e o Y a crescer para cima
- Projecção
  - A projecção por omissão é paralela
  - O volume de visão é um cubo de lado 2 centrado na origem
  - Significa que, **do ponto de vista da “câmara”**, vemos uma unidade para a esquerda, uma para a direita, uma para cima e uma para baixo
  - Também significa que se vê uma unidade para trás e uma para a frente
    - Não é real, mas é assim que funciona

# Transformação de Projection (2)

- Projecção Paralela:

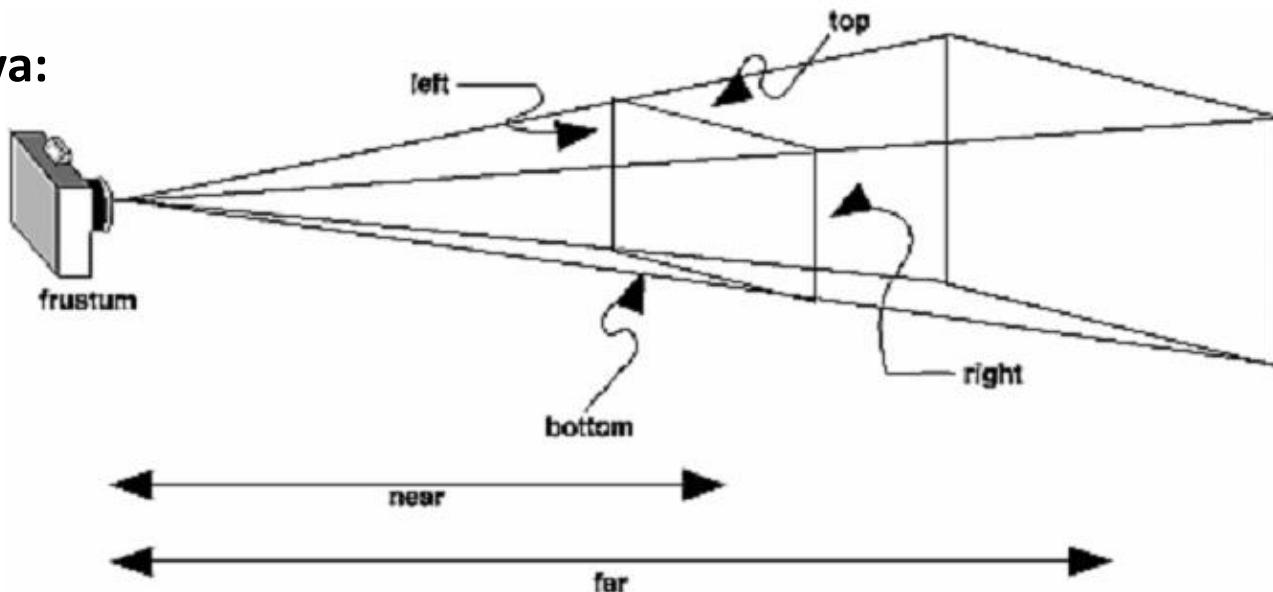


```
void glOrtho(GLdouble left, GLdouble right,  
             GLdouble bottom, GLdouble top,  
             GLdouble near, GLdouble far);
```

- Atenção:
  - Tipicamente utilizam-se valores positivos para near e far (analogia com a câmara)
  - Não se deve colocar o valor 0 no near!

# Transformação de Projection (3)

- Projecção Perspectiva:



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



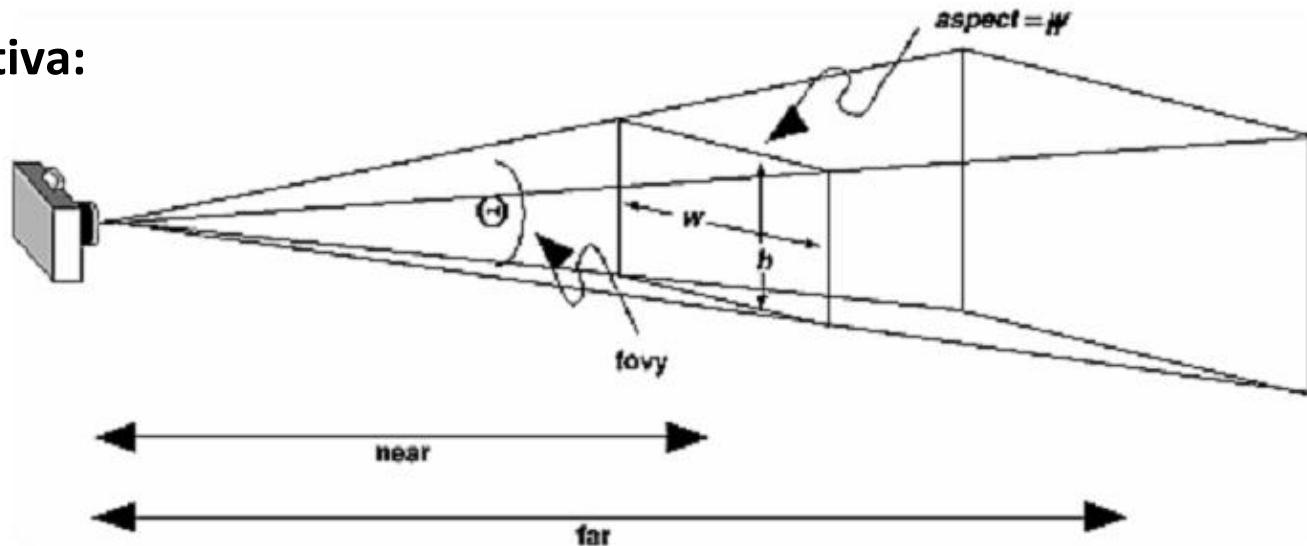
Utilização  
pouco  
intuitiva...

- Atenção:

- *left*, *right*, *bottom* e *top* definem os valores do plano *near*
- *near* e *far* têm funcionamento equivalente à projecção paralela

# Transformação de Projection (4)

- Projecção Perspectiva:



```
void gluPerspective(GLdouble fovy, GLdouble aspect,  
                    GLdouble near, GLdouble far);
```



- Atenção:

- fovy é o ângulo no plano X-Z. Deve ter valores entre [0; 180]
- aspect é a relação de aspecto entre a largura e altura do plano *near* (*w/h*)
- near e far têm funcionamento equivalente à projecção paralela

Utilização  
intuitiva...

# Transformação de Projection (5)

- P: Quando é que se deve definir a projecção?
  - R: Quando a janela (do SO) for redimensionada
  - R: Utiliza-se o *callback* `glutReshapeFunc(...)` do GLUT
- Exemplo de definição de projecção prespectiva:
  - Abertura de 60º, relação de aspecto igual à da janela
  - Volume de visão desde 1 até às 1000 unidades (relativo à direcção da câmara)

```
int main(int argc, char** argv)
{
    /* inicializar a janela */
    ...
    /* callbacks */
    ...
    glutReshapeFunc(onResize);
    /* main loop */
    glutMainLoop();
    return 0;
}
```

```
void onResize(int newWidth, int newHeight) {
    /* alterar viewport para usar toda a janela */
    glViewport(0, 0, newWidth, newHeight);
    /* matrix de projecção */
    glMatrixMode(GL_PROJECTION);
    float aspect = newWidth / newHeight;
    glLoadIdentity();
    gluPerspective(60, aspect, 1, 1000);
    /* voltar à matrix modelView */
    glMatrixMode(GL_MODELVIEW);
}
```

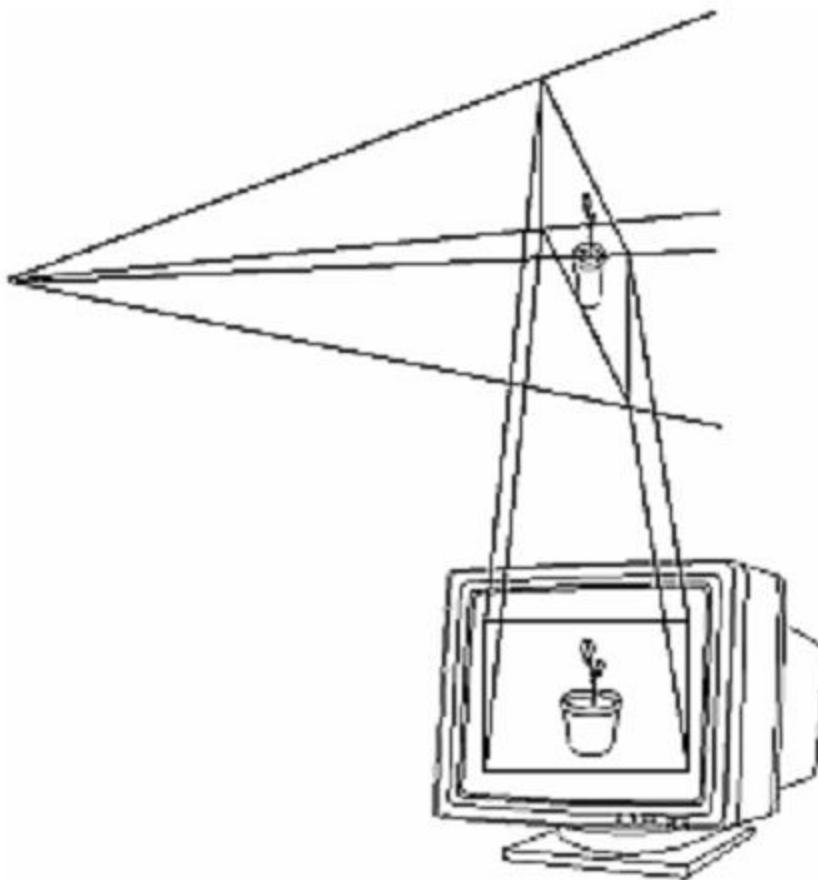
# Transformação de Viewport (1)

---

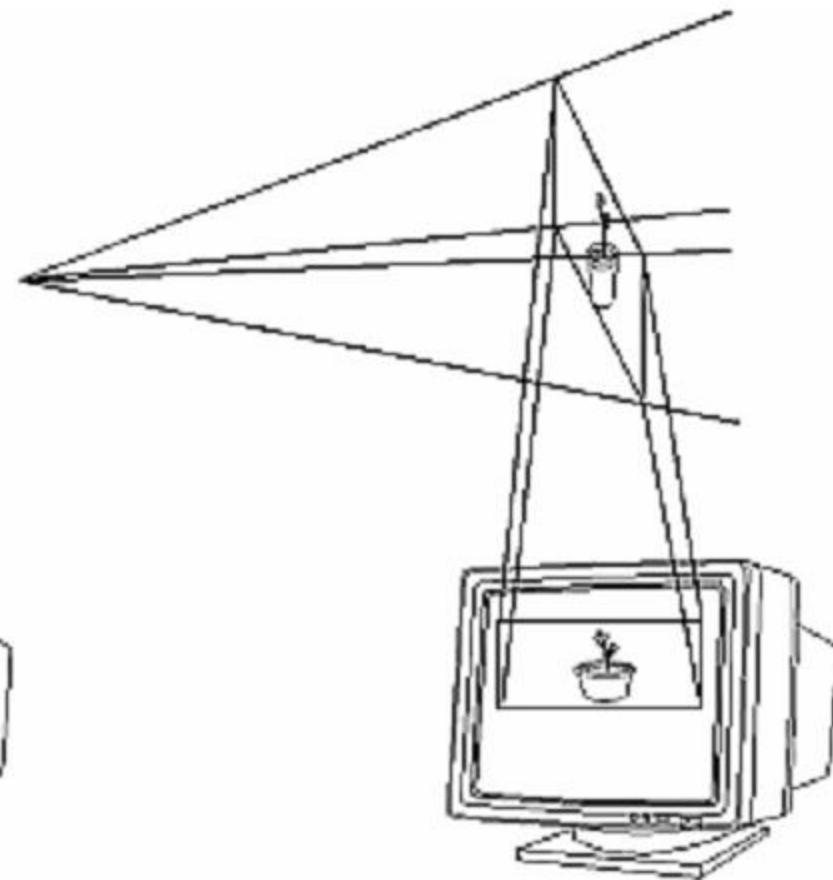
- **Objectivo:**
  - Define a forma como a cena é apresentada no ecrã
  - Transformar coordenadas do mundo em coordenadas de ecrã
- **Comando OpenGL:**
  - `glViewport(x, y, w, h)`
  - Os valores de `x, y, w, h` são definidos em pixeis em relação à área da janela
  - Os valores (`x, y`) representam o canto inferior esquerdo da área do *viewport*
  - `w (width)` representa a largura e `h (height)` representa a altura
- Quando é necessário chamar este comando?
  - Quando a janela é redimensionada
  - Usando o GLUT, é necessário registar no evento `glReshapeFunc`

# Transformação de Viewport (2)

---



**undistorted**



**distorted**

---

# Exercício

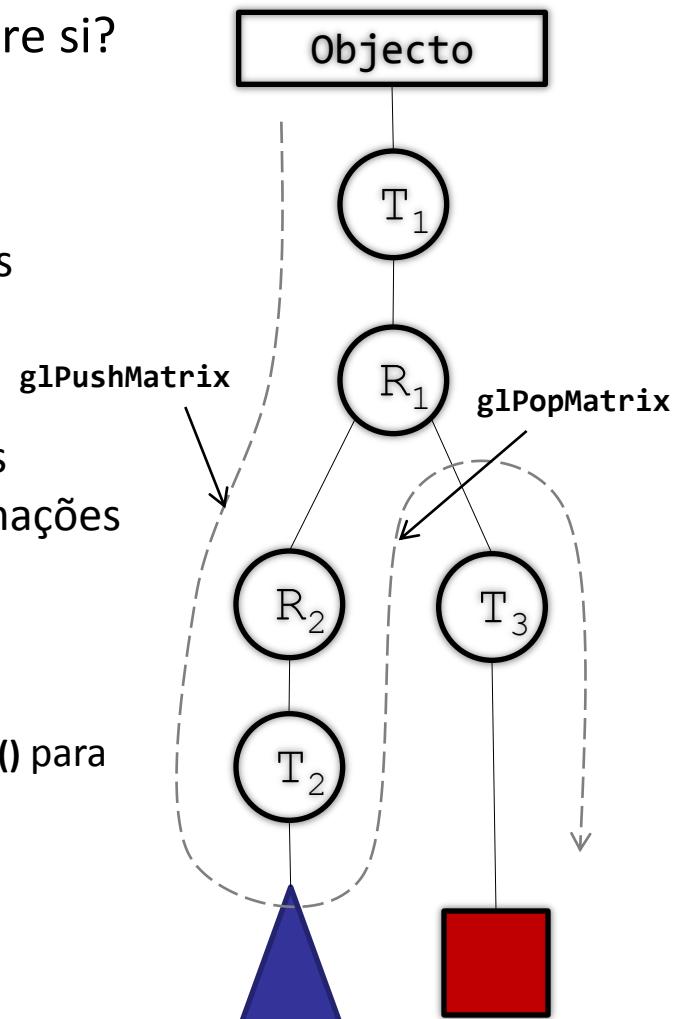
---

Primitivas OpenGL com N vértices



# Push e Pop de matrizes

- Como desenhar vários objectos, relacionados entre si?
- Existem duas formas:
  1. Utilizar o comando `glLoadIdentity()` e aplicar todas as transformações por cada um dos objectos
  2. Fazer as transformações comuns a vários objectos e depois guardar esse estado, aplicar as transformações de um objecto, repor o estado (transformações comuns) e aplicar as transformações do outro objecto
    - Utilizam-se os comando `glPushMatrix()` e `glPopMatrix()` para guardar e repor o estado das transformações (da *modelView*)
- Devem usar, sempre que possível, a opção 2!

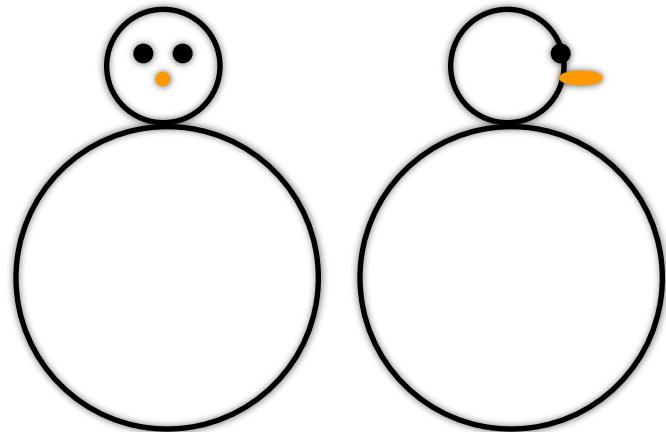


# Demo

---

## Mostrar funcionamento da matriz ModelView

- Desenhar um boneco de neve
- Desenhar vários bonecos de neve



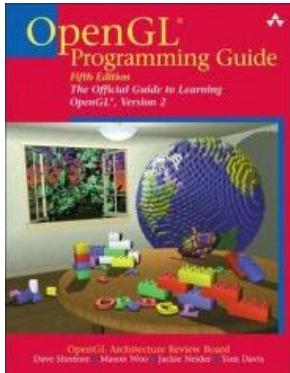
# Iluminação (apenas o essencial!)

---

- O comando `glColor3f(...)` apenas funciona quando não temos iluminação
- Para ter iluminação é necessário definir fontes de luz
  - Por omissão, no OpenGL, existe uma luz (luz 0) “bem” configurada (*luz do sol*)
  - Só é necessário activá-la...
  - Também é necessário activar a iluminação, além das fontes de luz...
- Activação/inactivação da iluminação e luz 0
  - `glEnable(GL_LIGHTING)`  e  `glDisable(GL_LIGHTING)`
  - `glEnable(GL_LIGHT0)`  e  `glDisable(GL_LIGHT0)`
- Em vez do comando `glColor3f(...)` devem-se definir as propriedades do material
  - `GLfloat matBrick[] = {0.9, 0.4, 0.1, 1.0};`
  - `glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, matBrick);`

# Referências

---



**OpenGL® Programming Guide: The Official Guide to Learning OpenGL(R)**, 5th Edition, OpenGL Architecture Review Board, Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, 2005

Capítulos: 1, 2, 3, 4  
Anexos: A, B, D e F

## OpenGL

<http://www.opengl.org/>

## OpenGL API Documentation Overview

<http://www.opengl.org/documentation/>

## GLUT

<http://www.xmission.com/~nate/glut.html>

<http://pyopengl.sourceforge.net/documentation/manual/reference-GLUT.html>

## Nate Robins Tutors

<http://www.xmission.com/~nate/tutors.html>