Desenvolvimento de aplicações tridimensionais com OpenGL

Pontifícia Universidade Católica de Minas Gerais, 2004

Alessandro Ribeiro

spdoido@yahoo.com.br

Bruno Evangelista

bpevangelista@yahoo.com.br

Orientador: Marcelo Nery

msnery@terra.com.br

Introdução

Sumário

- **Apresentação do curso**
- **Introdução ao PUx (Linux da PUC-MG)**
- **Apresentação do Fluxbox**
- **Editores de texto (vi, vim, Emacs, NEdit, Kate)**
- **Compilando programas no Linux (gcc, g++)**
- **Arquivos montadores (Makefiles)**
- **Depuração de programas (GDB)**
- **Padrões de programação multi-plataforma**

Introdução ao SDL

Sumário

- **O que é SDL?**
- **Recursos**
- **Onde é utilizado?**
- **Suporte a plataformas**
- **Diagrama do SDL**
- **Inicialização do SDL**
- **Video**
- **Processamento de eventos**
- **Exercícios**

O que é SDL?

- **SDL é uma API(Application Program Interface) para desenvolvimento de aplicações multimídia, multiplataforma**
- **Desenvolvida para facilitar o desenvolvimento de aplicativos e jogos que possuem versões para várias plataformas diferentes**
- **Utilizando SDL pode-se escrever aplicações portáteis e flexíveis**

Recursos

- **Possui várias interfaces nativas de alto nível**
  - **Vídeo**
  - **Dispositivos de entrada**
  - **Som**
  - **Eventos**
  - **Processos**
  - **Temporizadores**
  - **Outros**

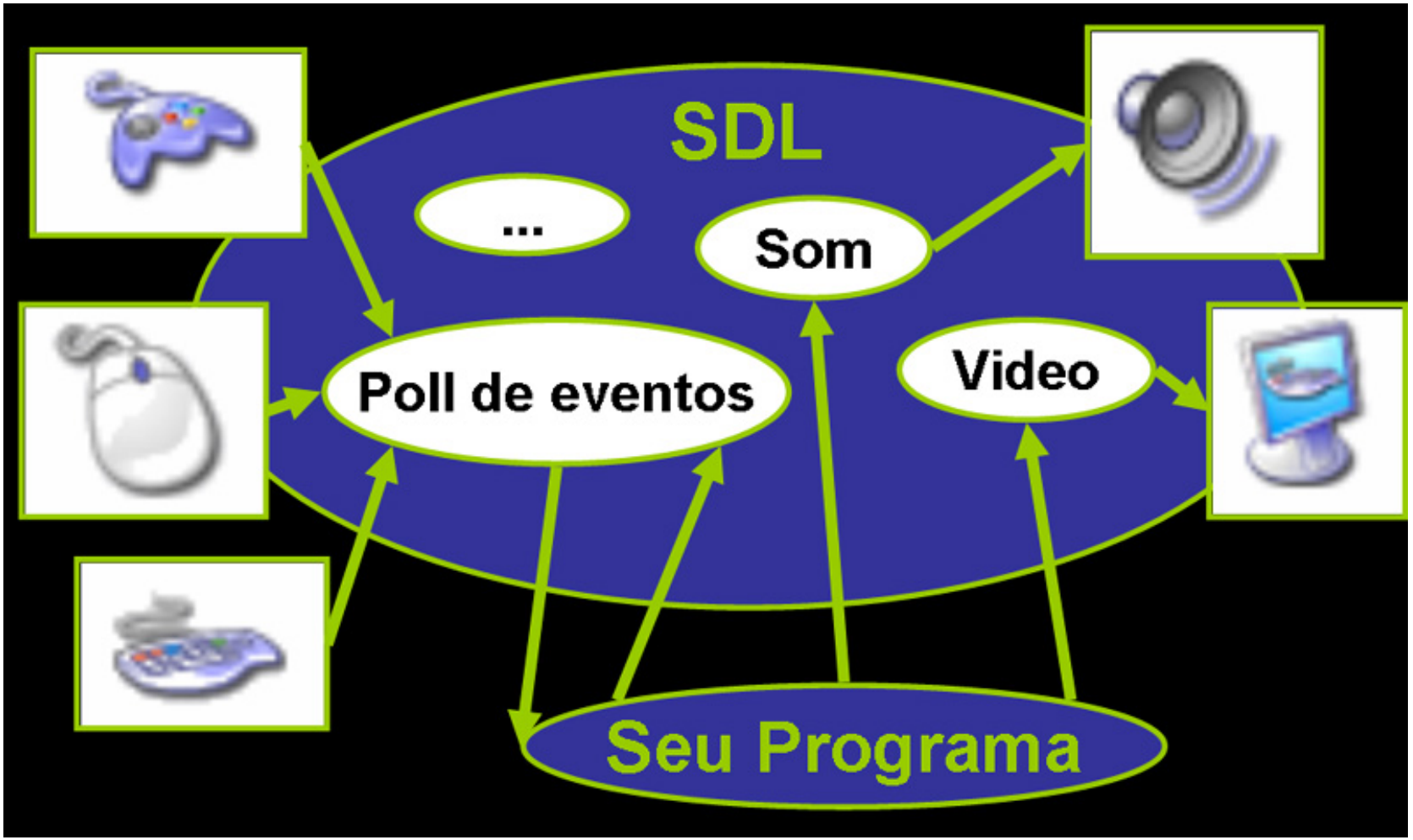
Onde é utilizado?

- **Aplicativos multimídia**
- **Jogos**
- **Kits de desenvolvimento de jogos**
- **Emuladores**
- **Demos**

Suporte as plataformas

- **Windows**
- **Linux**
- **BeOS**
- **MacOS**
- **Playstation 2**
- **Outras**

Diagrama do SDL



Inicialização do SDL

- **O SDL é composto por vários subsistemas. Antes de utilizarmos algum desses subsistemas precisamos iniciá-los utilizando o comando** SDL_Init

int SDL_Init(**Uint32** *flags*);

Inicializa os subsistemas do SDL

Inicialização do SDL

Flags	Descrição
SDL_INIT_TIMER	Temporizador
SDL_INIT_AUDIO	Som
SDL_INIT_VIDEO	Video
SDL_INIT_CDROM	CDROM
SDL_INIT_JOYSTICK	Controle
SDL_INIT_EVERYTHING	Todos os acima
SDL_INIT_NOPARACHUTE	Previne sinais fatais
SDL_INIT_EVENTTHREAD	

Inicialização do SDL

```
#include "SDL.h"
int main() {
    /** Initialize SDL Video */
    if (SDL_Init(SDL_INIT_VIDEO) < 0) {
       /** Error */
    }
    /* Shutdown all subsystems */
    SDL_Quit();
}
```

Video

- **Video é provavelmente o recurso mais utilizadoo do SDL e também um dos mais completos**
- **Após iniciarmos o subsistema de vídeo devemos configurar o modo de vídeo com o comando** $SDL\_SetVideoMode$

Video

SDL_Surface* SDL_SetVideoMode(**int** *width*, **int** *height*, **int** *bpp*, **Uint32** *flags*);

Configura o modo de vídeo com a largura, altura e bits por pixel

- **Quando o** *bpp* **for igual a 0, o SDL utiliza a configuração de bits por pixel atual do video**
- **O** *flags* **pode ser uma combinação de várias configurações**

Video

Principais flags	Descrição
SDL_DOUBLEBUF	Habilita *Double buffering*
SDL_FULLSCREEN	Usa tela cheia
SDL_OPENGL	Suporte ao OpenGL
SDL_RESIZABLE	Redimensionar janela

Processamento de eventos

- **O processamento de eventos permite a aplicação receber dados do usuário e da própria aplicação**
- **O processamento de eventos é inicializado junto com os subsistemas do SDL(Vídeo, controle, etc)**
- **A chave no tratamento de eventos do SDL é a união** SDL_Event

Processamento de eventos

- **SDL_Event é a união de todos os possíveis eventos do SDL**
  - **SDL_UserEvent**
  - **SDL_KeyboardEvent**
  - **SDL_MouseMotionEvent**
  - **Outros**
- **O SDL guarda os eventos em uma fila**
- **A fila de eventos é composta por uma série de uniões** SDL_Events**, uma para cada evento**

Processamento de eventos

- **Usando funções como** $SDL\_PoolEvent$ **e** $SDL\_PeepEvent$ **podemos tratar os eventos da fila**

int SDL_PollEvent(SDL_Event* $event$**);**

Verifica eventos pendentes. Retorna 1 se existem eventos pendentes ou 0 se não existem eventos

Exercícios

Introdução ao OpenGL

Sumário

- **O que é OpenGL?**
- **Porque utilizar?**
- **Onde é utilizado?**
- **História do OpenGL**
- **Qual linguagem utilizar?**
- **Um exemplo de código**
- **Sintaxe de comando**
- **Bibliotecas relacionadas**
- **Programas tutoriais do "Nate Robins"**

O que é OpenGL?

- **Uma interface de software(API) para hardware gráfico**
- **Interface com aproximadamente 250 comandos distintos (OpenGL 1.2)**
  - **Aproximadamente 200 comandos no núcleo do OpenGL**
  - **Aproximadamente 50 comandos na biblioteca de utilidades "GLU"**

O que é OpenGL?

- **Não possui comandos para manipulação de janelas, ou entrada de dados do usuário**
  - **Mouse, teclado, etc**
- **Projetado para ser independente de hardware, podendo ser implementado em várias plataformas**
  - **Computadores pessoais**
  - **Celulares, Palms e outros sistemas embutidos "OpenGL ES"**
  - **Video games**

O que é OpenGL?

- **Cada plataforma deve possuir sua implementação do OpenGL em hardware ou software**

O que é OpenGL?

http://www.opengl.org

Endereço da página oficial do OpenGL, onde podem ser encontradas documentações, exemplos, artigos, foruns, dentre vários outros

Porque utilizar?

- **Biblioteca gráfica tridimensional de baixo nível, portátil e extensível**
- **Possui linguagem própria de shaders**
- **Simples e de fácil aprendizado**
- **Muito rápida, possuindo suporte a aceleração por hardware**
- **Extensa documentação**

Onde é utilizado?

- **Aplicações científicas**
- **Softwares de engenharia (Autocad)**
- **Softwares de modelagem e animação 3D (3D Studio Max, Maya, Blender)**
- **Jogos (Doom 3, Never Winter Nights, Counter Strike)**

História do OpenGL

- **O precursor do OpenGL era o GL(Graphics Library) da Silicon Graphics**
- **"IRIS GL" era a API gráfica utilizada nas estações gráficas IRIS**
- **OpenGL é o resultado do trabalho da SGI para melhorar a portabilidade do "IRIS GL"**
- **OpenGL oferece os recursos do GL, mas é aberto(Open), facilitando a portabilidade**

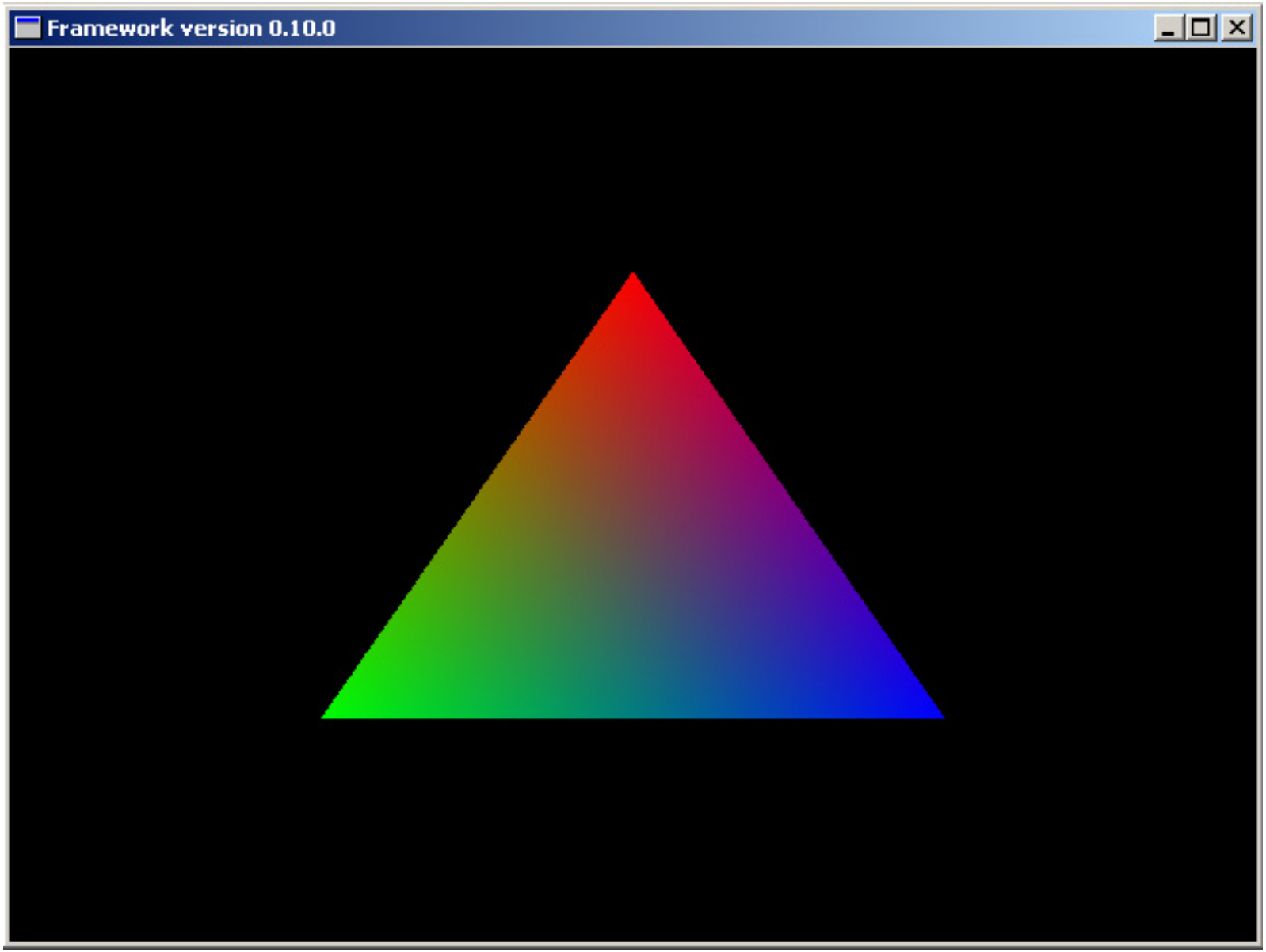
História do OpenGL

- **Atualmente o OpenGL é mantido pelo OpenGL Architecture Review Board(ARB)**
- **Esse grupo é formado por profissionais de várias empresas 3Dlabs, Apple, SGI, ATI, nVidia, Sun**
- **O OpenGL ARB se encontra a cada três meses para discutir possiveis melhorias na arquitetura do OpenGL**

Qual linguagem utilizar?

- **Exitem implementações do OpenGL para diferentes linguagens e compiladores**
  - **Ada**
  - **C/C++ (Visual C++, Borland C++, GCC)**
  - **C#**
  - **Delphi**
  - **Fortran**
  - **Java**
  - **SmallTalk**
  - **Visual Basic**
  - **Outros**

Um exemplo de código


Framework version 0.10.0

```
InicializaJanela();
glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLES);
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(0.25f, 0.25f, 0.0f);
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(0.75f, 0.25f, 0.0f);
    glColor3f(0.0f, 0.0f, 1.0f);
    glVertex3f(0.50f, 0.75f, 0.0f);
glEnd();
glFlush();
AtualizaJanela();
```

Sintaxe de comando

- **Comandos do OpenGL utilizam o prefixo "gl" e letras maiúscula no inicio do nome do comando**
  - $glClearColor$
  - $glBegin$
- **As constantes do OpenGL possuem o prefixo "GL_", e utilizam apenas letras maiúsculas**
  - $GL\_COLOR\_BUFFER\_BIT$
  - $GL\_TRIANGLES$

Sintaxe de comando

- **Algumas comandos do OpenGL aceitam mais de oito tipos diferentes de dados para seus argumentos**
- **Os comandos do OpenGL utilizam o sufixo para especificar o número de parâmetros e o tipo de dado que recebem**
  - *"glColor3f"* **Define a cor, recebendo três floats como parâmetro**
  - *"glVertex2i"* **Define um vértice, recebendo dois inteiros como parâmetro**

Sintaxe de comando

Sufixo	Tipo de dado	Bits	ANSI C	OpenGL
b	inteiro	8	char	GLbyte
ub	inteiro s/ sinal	8	uchar	GLubyte
s	inteiro	16	short	GLshort
us	inteiro s/ sinal	16	ushort	GLushort
i	inteiro	32	int	GLint
ui	inteiro s/ sinal	32	uint	GLuint
f	ponto-flutuante	32	float	GLfloat
d	ponto-flutuante	64	double	GLdouble

Sintaxe de comando

- **Alguns comandos do OpenGL podem possuir uma última letra "v", que indica que o comando recebe um ponteiro para um vetor de valores, ao invés de uma série de parâmetros individuais**
  - $glColor3fv(colorArray)$
  - $glVertex3fv(vertexArray)$

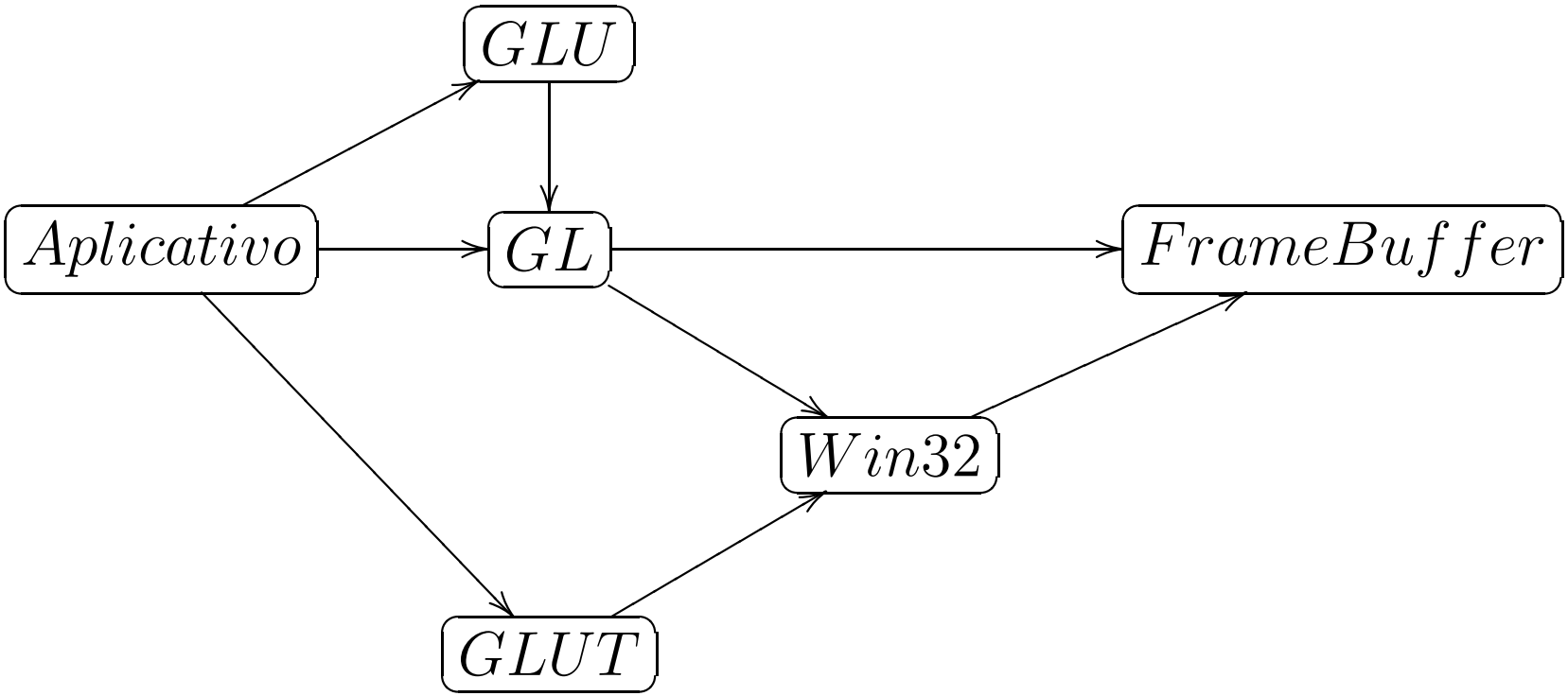
Bibliotecas relacionadas

- **GLU - OpenGL Ultility**
  Biblioteca de utilidades do OpenGL. Possui um conjunto de funções para desenhar objetos, criar mipmaps de texturas, etc
- **A biblioteca GLU 1.2 é distribuida junto com a versão 1.1 do OpenGL**

Bibliotecas relacionadas

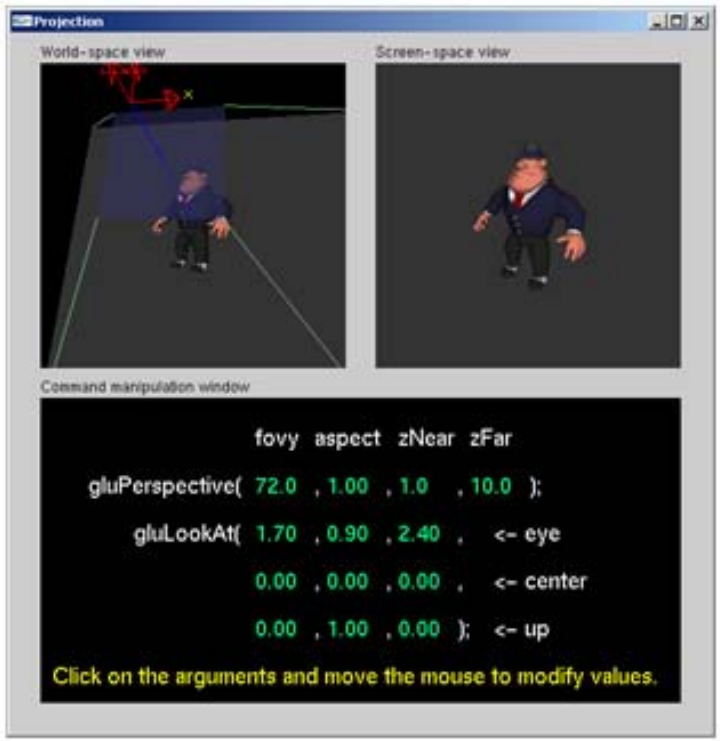
- **GLUT - The OpenGL Utility Toolkit**
  Biblioteca para gerenciamento de janelas, entrada de dados, dentre outros. Possibilita escrever programas para OpenGL independentes de plataforma
- **Recursos:**
  - **Funções para gerenciamento de janelas**
  - **Processamento de eventos**
  - **Dispositivos para entrada de dados**
  - **Geração de vários sólidos**

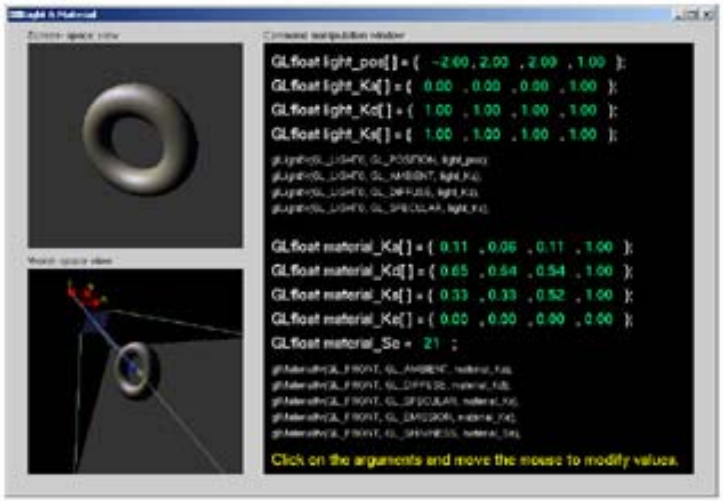
Bibliotecas relacionadas

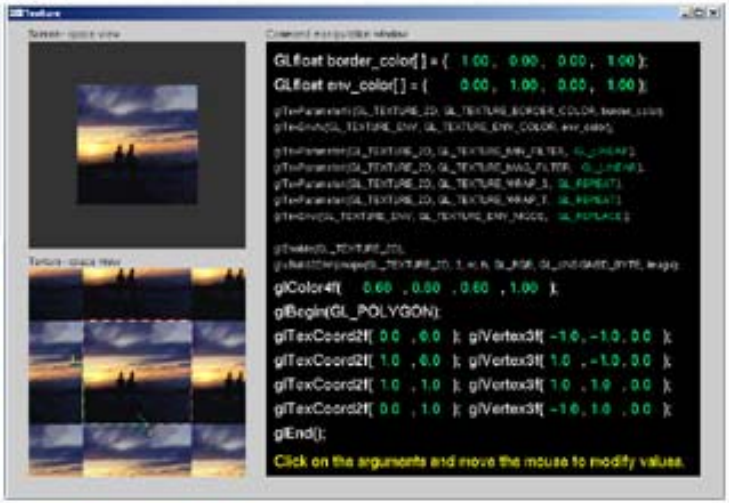


Programas tutoriais do "Nate Robins"

- **Nate Robins desenvolveu alguns programas tutoriais que podem ajudar na visualização dos conceitos básicos de computação gráfica e OpenGL**
- **Esses programas permitem a alteração de parâmetros dos comandos do OpenGL em tempo real, possibilitando um entendimento melhor do seu funcionamento**


Projection
World-space view
Screen-space view
Command manipulation window
fovy aspect zNear zFar
gluPerspective(72.0 , 1.00 , 1.0 , 10.0);
gluLookAt(1.70 , 0.90 , 2.40 , <- eye
0.00 , 0.00 , 0.00 , <- center
0.00 , 1.00 , 0.00); <- up
Click on the arguments and move the mouse to modify values.


Click on the arguments and move the mouse to modify values.


Click on the arguments and move the mouse to modify values.

Nate Robins

Programas tutoriais do "Nate Robins"

http://www.xmission.com/~nate/opengl.html

Endereço da página oficial, onde os programas podem ser encontrados para download

Renderização de primitivas

Sumário

- **Conceitos básicos de animação**
- **Limpando a tela**
- **Terminando o desenho**
- **Sistema de coordenadas básico**
- **Desenhando primitivas**
- **Tipos básicos de primitivas geométricas**
- **Tipos derivados de primitivas geométricas**
- **Especificando informações dos vértices**
- **Tipos derivados de primitivas geométricas**

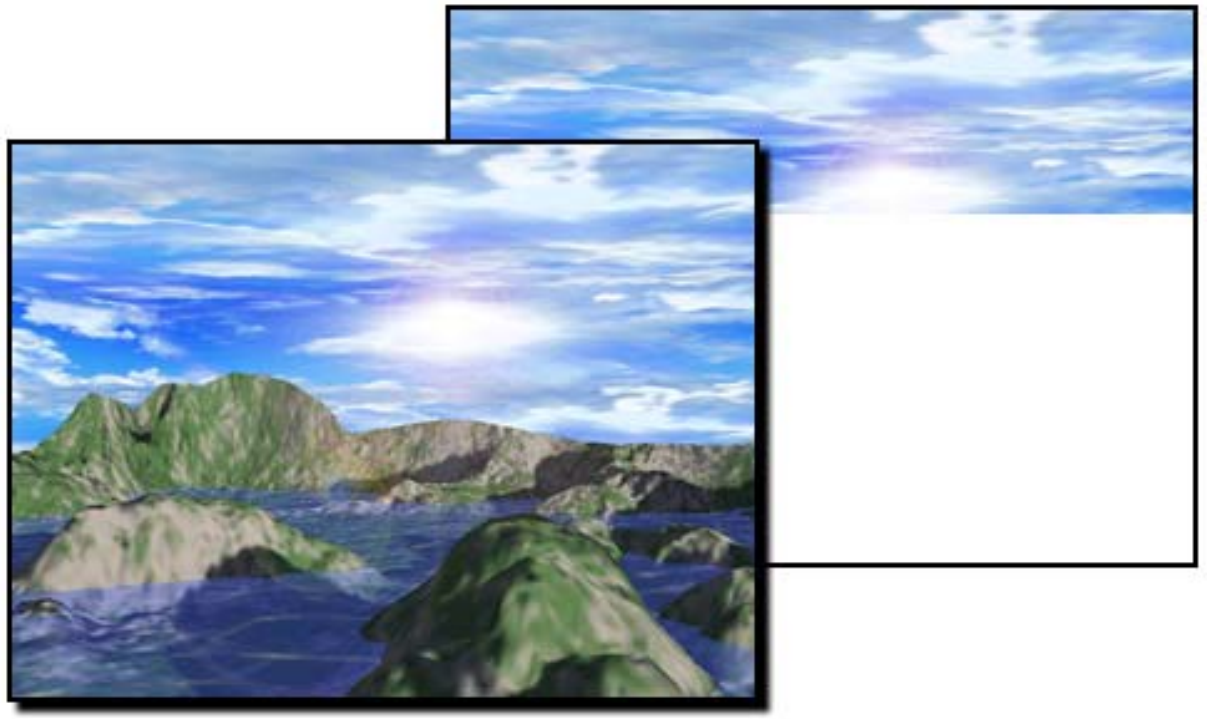
Sumário

- **Especificando informações dos vértices**
- **Gerenciamento de estados**
- **Modos de desenho**
- **Seleção de face**
- **Exercícios**

Conceitos básicos de animação

- **Para criarmos o efeito de animação precisamos redesenhar a tela várias vezes**
  Animação = Redesenhar + Trocar os desenhos
- **Para trabalharmos com animações suaves, utilizaremos dois "buffers"**
  - **Front Buffer: Buffer que esta sendo exibido para o usuário**
  - **Back buffer: Buffer oculto onde estamos desenhando**

Conceitos básicos de animação



Conceitos básicos de animação

- **Após desenharmos no "back buffer" trocamos os buffers, e redesenhamos toda a cena, impedindo que o usuário veja o conteudo da cena sendo desenhado**
- **O OpenGL não possui um comando para troca de buffers**
- **Extensões do OpenGL implementam a troca de buffers em diversas plataformas, utilizando comandos distintos**

Conceitos básicos de animação

Plataforma	Extensão	Comando
Apple Macintosh	AGL	aglSwapBuffers
Linux	GLX	glXSwapBuffers
Playstation2 ou OS2	PGL	pglSwapBuffers
Windows	WGL	SwapBuffers

Conceitos básicos de animação

Limpando a tela

- **Quando algum objeto é desenhado seus** $pixels$ **são escritos no buffer de cor, este buffer contêm a cor de cada** $pixel$ **visível da cena**
- **Antes de qualquer objeto ser desenhado é preciso limpar o buffer de cor, que geralmente está preenchido com a última cena desenhada**
- **Para limpar o buffer de cor devemos preenchê-lo com alguma cor de fundo**
- **Geralmente a cor dos pixels são armazenadas no formato RGBA(Red, Green, Blue, Alpha)**

Limpando a tela

- **Existem vários buffers(Hardware ou Software), adicionais ao buffer de cor que precisam ser limpos constantemente**
- **Um comando para limpeza de buffers deve permitir que qualquer combinação de buffers possa ser limpa ao mesmo tempo**
- **Os valores utilizados para limpeza dos buffers devem ser previamente definidos**

Limpando a tela

void glClear(**GLbitfield** *mask*);

Limpa os *buffers* especificados pela mascara, preenchendo-os com os valores de limpeza previamente definidos

void glClearColor(**GLclampf** *red*, **GLclampf** *green*, **GLclampf** *blue*, **GLclampf** *alpha*);

Define a cor utilizada para limpar do buffer de cor no formato RGBA

Limpando a tela

void glClearDepth(**GLclampd** *depth*);

Define o valor de profundidade utilizado para limpar o buffer de profundidade

Limpando a tela

Buffer	Nome
Buffer de cor	GL_COLOR_BUFFER_BIT
Buffer de profundidade	GL_DEPTH_BUFFER_BIT
Buffer acumulador	GL_ACCUM_BUFFER_BIT
Buffer stencil	GL_STENCIL_BUFFER_BIT

Terminando o desenho

- **O OpenGL foi desenvolvido para ser utilizado em estações gráficas, possuindo uma arquitetura cliente/servidor**
- **Cliente e servidor não necessáriamente precisam ser a mesma máquina na rede**
- **Cliente - Onde o programa é executado**
- **Servidor - Onde os resultados do** *processamento gráfico* **realizado pelo cliente são exibidos**

Terminando o desenho

- **O cliente guarda um grande número de comandos a serem executados em um pacote antes de enviá-lo pela rede**
- **O servidor não tem como descobrir quando o desenho de uma cena está completo para exibi-lo**
- **Para terminamos o desenho utilizamos os comandos** $glFlush$ **e** $glFinish$

Terminando o desenho

void glFlush(void);

Manda o pacote com todos os comandos anterires para o servidor, forçando-os a serem executados. Não aguarda o termino da execução dos comandos

void glFinish(void);

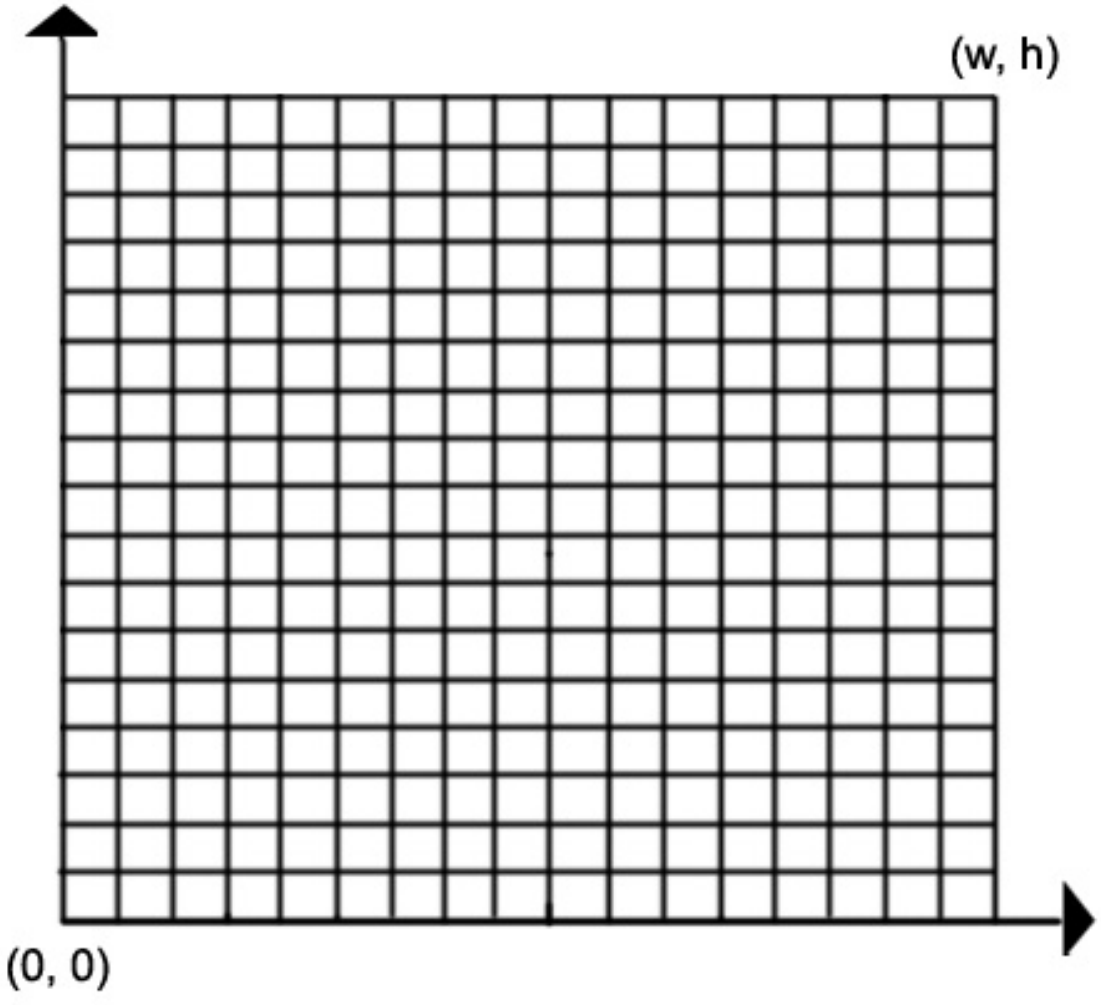
Força todos os comandos anteriores do OpenGL a serem completados. Retorna após a confirmação da execução de todos os comandos anteriores

Sistema de coordenadas básico

- **Sempre que uma janela é criada ou quando seu tamanho é modificado, o gerenciador de janelas ativa um evento**
- **Quando esse evento for ativado devemos chamar uma função que configure a área de desenho do OpenGL**
- **Essa função deve realizar os seguintes passos:**
  - **Definir a área da tela onde a figura será desenhada**
  - **Definir o sistema de coordenadas utilizado para desenhar os objetos**

Sistema de coordenadas básico

```
/** Configura a área de desenho */
void reshape(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0, w, 0.0, h, -1.0, 1.0);
}
```


(w, h)
(0, 0)

Desenhando primitivas

- **Para desenharmos primitivas geométricas com o OpenGL utilizamos os comandos** $glBegin$ **e** $glEnd$
- **Utilizamos o comando** $glBegin$ **para definirmos o inicio da construção de uma primitiva geométrica, e especificarmos o seu tipo**
- **Utilizamos o comando** $glEnd$ **para definirmos o fim da construção de uma primitiva geométrica**
- **Entre os comandos** $glBegin$ **e** $glEnd$ **são passadas as informações dos vértices que compõem a primitiva**

Desenhando primitivas

void glBegin(GLenum *mode***);**

Marca o início da lista de informações dos vértices que compõem a primitiva geométrica. O tipo de primitiva é indicado por *mode***.**

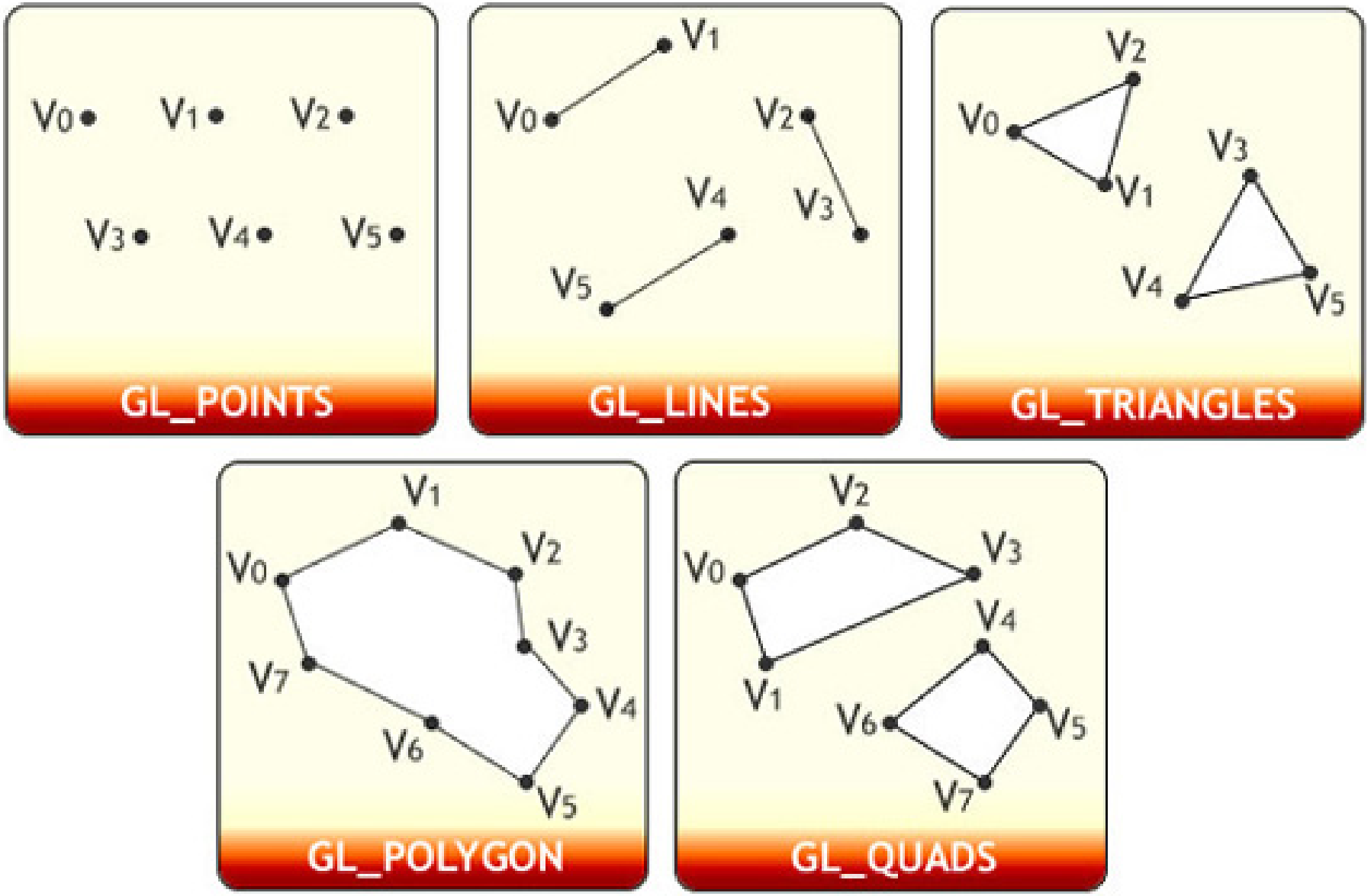
void glEnd(void);

Marca o fim da construção da primitiva geométrica

Desenhando primitivas

- **Entre os comandos $glBegin$ e $glEnd$ só podem existir comandos do OpenGL que especifiquem informações dos vértices**
- **O uso de outros comandos do OpenGL geram erros**
- **Essas restrições só se aplicam aos comandos do OpenGL, sendo permitido o uso de construções da linguagem de programação utilizada**

Tipos básicos de primitivas geométricas


V0
V1
V2
V3
V4
V5
GL_POINTS
GL_LINES
GL_TRIANGLES
V6
V7
GL_POLYGON
GL_QUADS

Tipos básicos de primitivas geométricas

- **GL_POINTS**

  Cada vértice representa um ponto

- **GL_LINES**

  Cada dois vértices representam uma linha

- **GL_TRIANGLES**

  Cada três vértices representam um triângulo

Tipos básicos de primitivas geométricas

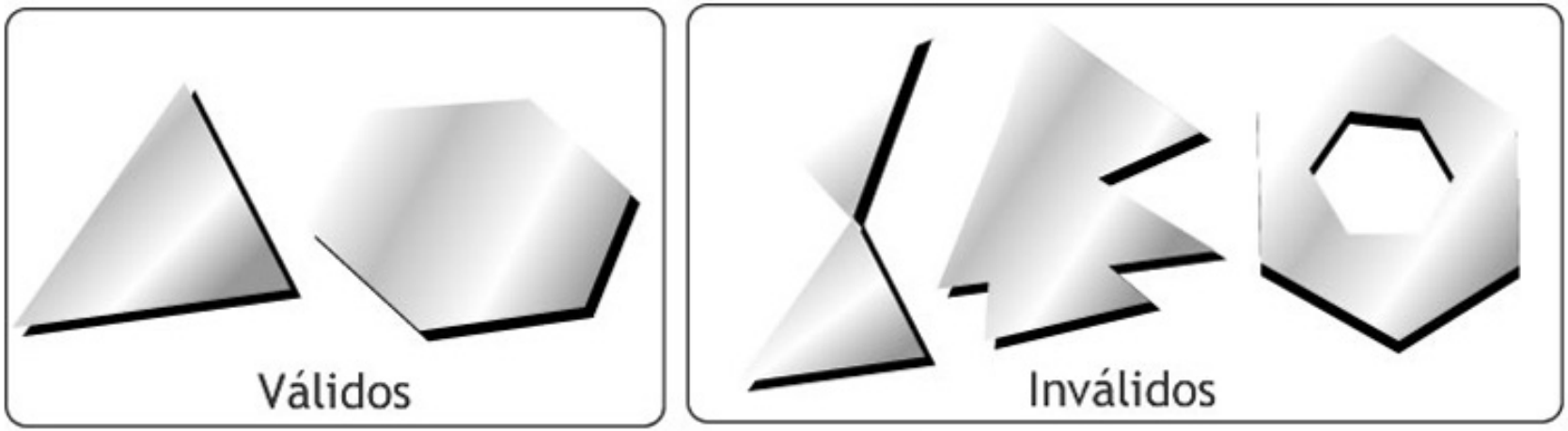
- **GL_QUADS**

  Cada quatro vértices representam um quadrilátero

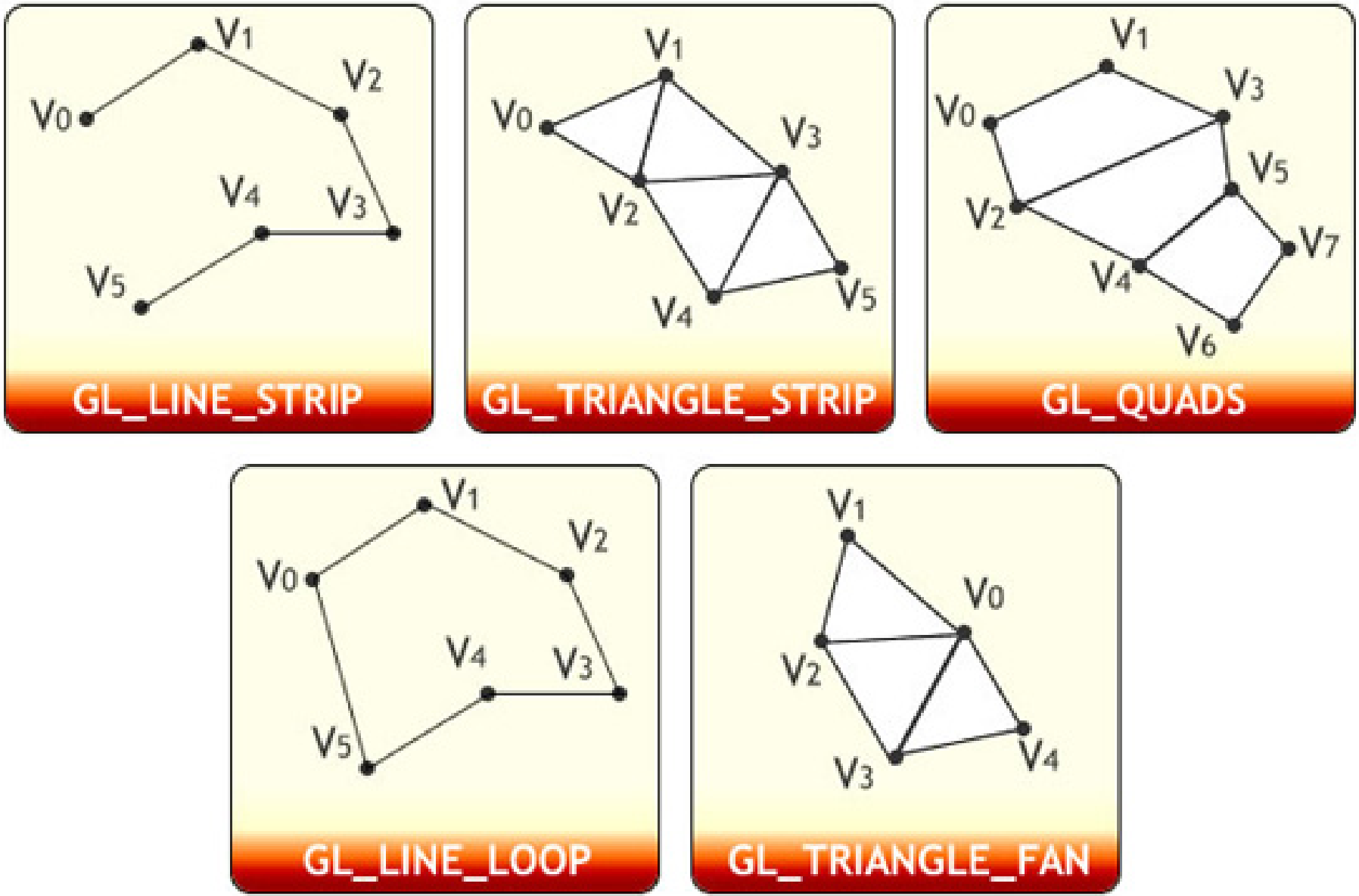
- **GL_POLYGON**

  Vários vértices formam um poligono que deve necessariamente ser convexo

Tipos básicos de primitivas geométricas



Tipos derivados de primitivas geométricas


V0
V1
V2
V3
V4
V5
GL_LINE_STRIP
GL_TRIANGLE_STRIP
V6
V7
GL_QUADS
GL_LINE_LOOP
GL_TRIANGLE_FAN

Tipos derivados de primitivas geométricas

- **GL_LINE_STRIP, GL_TRIANGLE_STRIP ou GL_QUAD_STRIP**

  Após a primeira primitiva ser gerada, cada nova primitiva utiliza o(s) vértices da primitiva anterior para gerar uma nova primitiva, formando uma "tira"

Tipos derivados de primitivas geométricas

- **GL_LINE_LOOP**

  Similar ao "line strip", é adicionada uma nova linha entre o primeiro e último vértice formando um laço

Tipos derivados de primitivas geométricas

- **GL_TRIANGLE_FAN**

  Após a primeira primitiva ser gerada, cada nova primitiva utiliza o(s) vértices da primitiva anterior para gerar uma nova primitiva. Todas as primitas compartilham um mesmo vértice central

Especificando informações dos vértices

- **glVertex*()**
  Define a coordenada do vértice
- **glColor*()**
  Define a cor do vértice
- **glIndex*()**
  Define a cor em modo $indexado$ **do vértice**
- **glNormal*()**
  Define a normal do vértice

Especificando informações dos vértices

- **glTexCoord*()**
  Define as coordenadas de textura do vértice
- **glMultiTexCoord*ARB()**
  Define as coordenadas de textura para multi-texturização
- **glEdgeFlag*()**
  Controla o desenho de arestas
- **glMaterial*()**
  Define as propriedades do material

Especificando informações dos vértices

- **glArrayElement*ARB()**
  Extrai informações do "vertex array"
- **glEvalCoord*(), glEvallPoint*()**
  Gera coordenadas dos vértices a partir de evoluidores
- **glCallList(), glCallLists()**
  Executa uma "Display List"

Gerenciamento de estados

- **O OpenGL é uma máquina de estados, possuindo diversos estados**
- **Podemos alterar seus estados e a nova configuração é mantida inalterada até que sejam realizadas novas modificações**
- **A cor de limpeza da tela, por exemplo, é uma variável de estado. Definimos o seu valor e ele é mantido até o final da execução do programa, caso não seja modificado**

Gerenciamento de estados

- **Vários estados do OpenGL podem ser habilitados ou desabilitados com os comandos** $glEnable$ **e** $glDisable$

void glEnable(**GLenum** cap);

Habilita um estado do OpenGL

void glDisable(**GLenum** cap);

Desabilita um estado do OpenGL

Gerenciamento de estados

- **Para verificarmos se uma estado está atualmente habilitado utilizamos o comando** *glIsEnabled*

GLboolean glIsEnabled(**GLenum** *capability*);

Verifica se um estado está habilitado, retornando *GL_TRUE* **ou** *GL_FALSE*

Gerenciamento de estados

- **Todos os estados possuem um valor padrão**
- **Vários estados como luz, texturização, neblina estão inicialmente desabilitados**

Modos de desenho

- **Um polígono possui duas faces, face da frente e face de trás, que precisam ser desenhadas**
- **Cada face deve ser desenhada de maneira diferente dependendo de qual face está virada para o observador**
- **Por padrão, ambas as faces são desenhadas da mesma maneira**
- **Para mudarmos a forma como as faces são desenhadas utilizamos o comando** $glPolygonMode$

Modos de desenho

void glPolygonMode(**GLenum** *face*, **GLenum** *mode*);

Controla o modo de desenho das faces de um polígono

Face	Descrição
GL_FRONT	Face da frente
GL_BACK	Fade de trás
GL_FRONT_AND_BACK	Ambas as faces

Modos de desenho

Modo	Descrição
GL_POINT	Desenha pontos nos vértices
GL_LINE	Desenha arestas
GL_FILL	Desenha polígono preenchido

Seleção de face

- **Por convenção, a face do polígono onde os vértices são definidos em sentido anti-horário é chamada de face da frente**
- **Existem dois sistemas de orientação utilizados para especificar a face frontal de um polígono, horário e anti-horário**
- **Alguns programas geram sólidos utilizando o sistema de orientação horário**

Seleção de face

- **Para resolver esse problema, o OpenGL nos permite definir qual sentido de orientação será utilizado para especificar a face frontal dos polígonos**
- **Para definirmos como a face da frente é selecionada utilizamos o comando** $glFrontFace$

Seleção de face

void glFrontFace(**GLenum** *mode*);

Controla como a face frontal do polygonos é determinada

Modo	Descrição
GL_CCW	Orientação anti-horária
GL_CW	Orientação horária

Seleção de face

- **Em sólidos fechados e opacos as faces interiores nunca estão visíveis, pois sempre são sobrepostas pelas faces frontais**
- **Portanto se estivermos na parte exterior do sólido podemos descartar suas faces interiores**
- **Descartando as faces ocultas aumentamos o desempenho de nossa aplicação**

Seleção de face

- **Para descartarmos as faces com o OpenGL devemos fazer os seguintes passos:**
  - **Habilitar o corte de faces**
  - **Especificar quais faces serão descartadas(cortadas)**
- **Para especificarmos quais faces serão descartadas utilizamos o comando** $glCullFace$

Seleção de face

void glCullFace(**GLenum** *mode*);

Especifica qual face do polígono deve ser descartada

Modo	Descrição
GL_FRONT	Face da frente
GL_BACK	Face de trás
GL_FRONT_AND_BACK	Ambas as faces

Exercícios

Transformações

Sumário

- **Sistemas de coordenadas**
- **Coordenadas homogêneas**
- **Comandos gerais de tranformação**
- **Transformações básicas**
- **Transformando normais**
- **Concatenação de transformações**
- **Projeções (OpenGL, GLU)**
- **Tranformações de visão**
- **Utilizando as pilhas de matrizes**

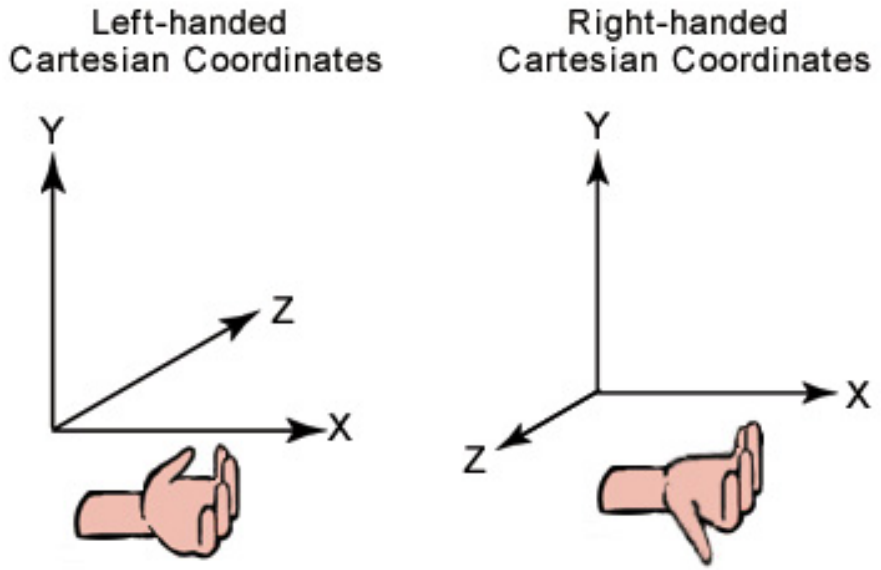
Sumário

- **Mapeamento da tela**
- **Exercícios**

Sistemas de coordenadas

- **Existem dois sistemas de coordenadas tridimensionais**
  - **Sistema de coordenadas da mão esquerda**
  - **Sistema de coordenadas da mão direita**
- **Esses sistemas de diferem em relação ao eixo Z**

Sistemas de coordenadas



Sistemas de coordenadas

- **Na mão esquerda consideramos o eixo Z positivo** *entrando* **na tela**
- **Na mão direita consideramos o eixo Z positivo** *saindo* **da tela**

Sistemas de coordenadas

- **O sistema de coordenadas da mão direita é mais utilizado, e costuma ser adotado em cursos de matemática**
- **Algumas APIs como o "DirectX" utilizam o sistema de coordenadas da mão esquerda**

Coordenadas homogêneas

- **O sistema de coordenadas homogêneas foi desenvolvido pela geometria e em seguida aplicado em computação gráfica**
- **Utilizamos coordenadas homogêneas para tratarmos todas as transformações com pontos e vetores de uma mesma maneira consistente, podendo combiná-las facilmente**

Coordenadas homogêneas

- **Nas coordenadas homogêneas adicionamos uma quarta coordenada para um ponto no espaço**

$$v = (x, y, z, w)^T$$

- **Várias coordenadas homogêneas podem representar o mesmo ponto**

$$v = (x, y, z, w)^T = (x/w, y/w, z/w, 1)^T$$
$$v = (2, 25, 13, 8)^T = (4, 50, 26, 16)^T$$

Coordenadas homogêneas

- **Um ponto tridimensional no espaço euclidiano** $v = (x, y, z)^T$ **se torna** $v = (x, y, z, 1.0)^T$
- **Um vetor tridimensional no espaço euclidiano** $v = (x, y, z)^T$ **se torna** $v = (x, y, z, 0)^T$
- **Os comandos do OpenGl trabalham com coordenadas de vértices bidimensionais e tridimensionais, mas internamente ambas são tratadas como coordenadas tridimensionais homogêneas**

Coordenadas homogêneas

- **O OpenGL pode não funcionar corretamente com o valor da coordenada** $w < 0$

Comandos gerais de tranformação

- **O OpenGl possui vários comandos úteis que nos ajudam a trabalhar com tranformações**

void glMatrixMode(**GLenum** *mode*);

Especifica qual matriz será modificada, GL_MODELVIEW, GL_PROJECTION ou GL_TEXTURE. Todos os comandos posteriores afetarão a matriz selecionada

Comandos gerais de tranformação

void glLoadIdentity(**void**);

Define a matriz corrente como matriz identidade

void glLoadMatrix{fd}(**const TYPE** $*m$);

Carrega um vetor com 16 valores para a matriz corrente

Comandos gerais de tranformação

void glMultMatrix{fd}(**const TYPE** $*m$);

Multiplica a matriz corrente pela matriz formada a partir do vetor com 16 valores

- **As matrizes no OpenGL são orientadas a colunas. Ao declarar uma matriz** $mat[i][j]$**,** i **é a coluna e** j **a linha da matriz**

Comandos gerais de tranformação

$$M = \begin{pmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{pmatrix}$$

Transformações básicas

- **É extremamente importante aprendermos as tranformações**
- **Transformações são a ferramenta básica para manipularmos geometrias**
- **Tranformações são aplicadas a partir de matrizes 4x4**

Transformações básicas

- **Várias APIs gráficas como o OpenGL, incluem operações com matrizes que implementam grande parte das transformações**

Transformações básicas

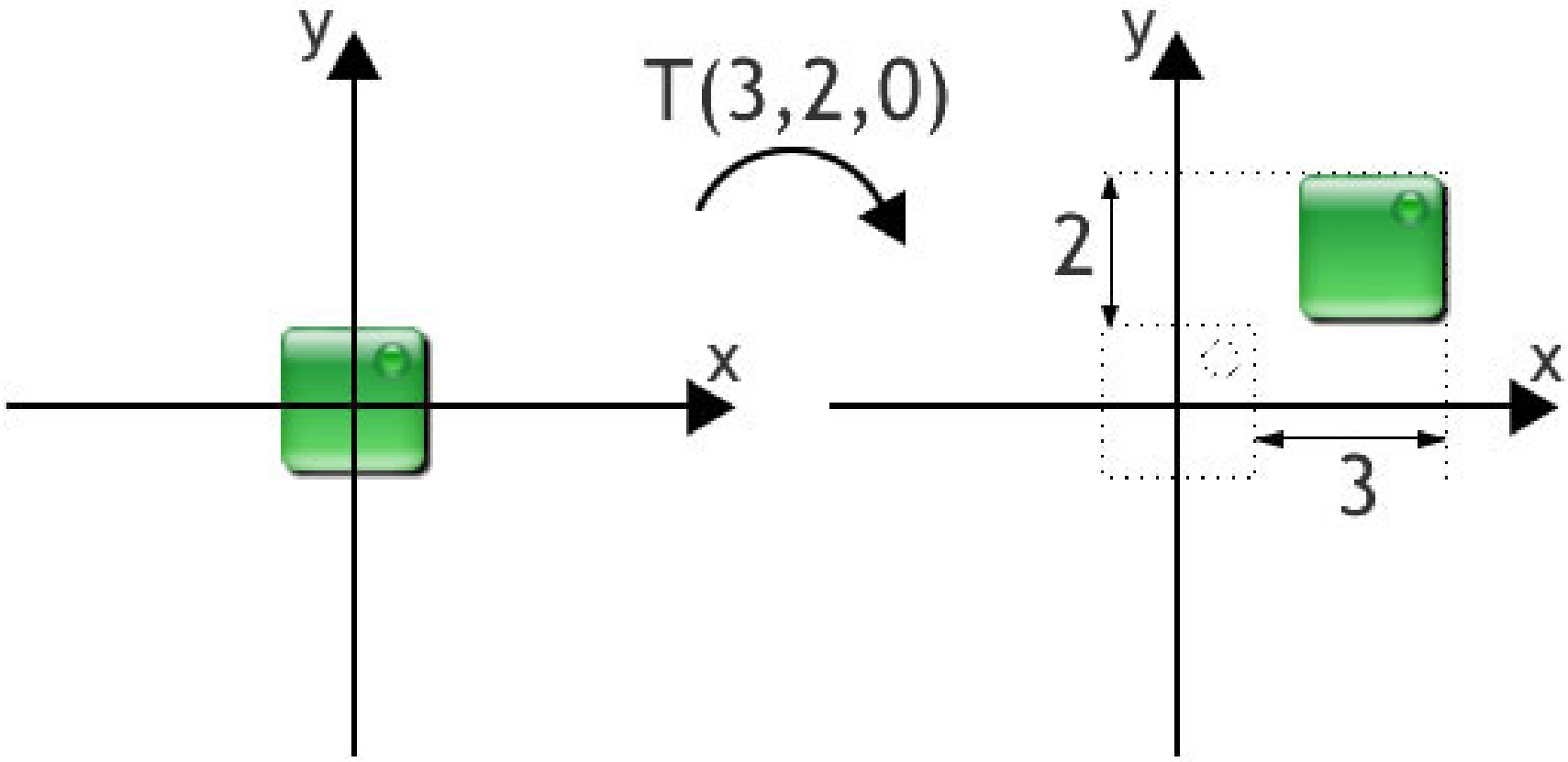
- **Transformações básicas**
  - **Translação**
  - **Rotação**
  - **Escala**

Transformações básicas

Translação $T(t_x, t_y, t_z)$**:**

$$T(t) = T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformações básicas



Transformações básicas

void glTranslate{**fd**}(**TYPE** x, **TYPE** y, **TYPE** z);

Multiplica a matriz corrente pela matriz de translação, transladando o sistema de coordenadas local

Transformações básicas

Rotação $R_x(\phi)$:

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & \sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformações básicas

Rotação $R_y(\phi)$:

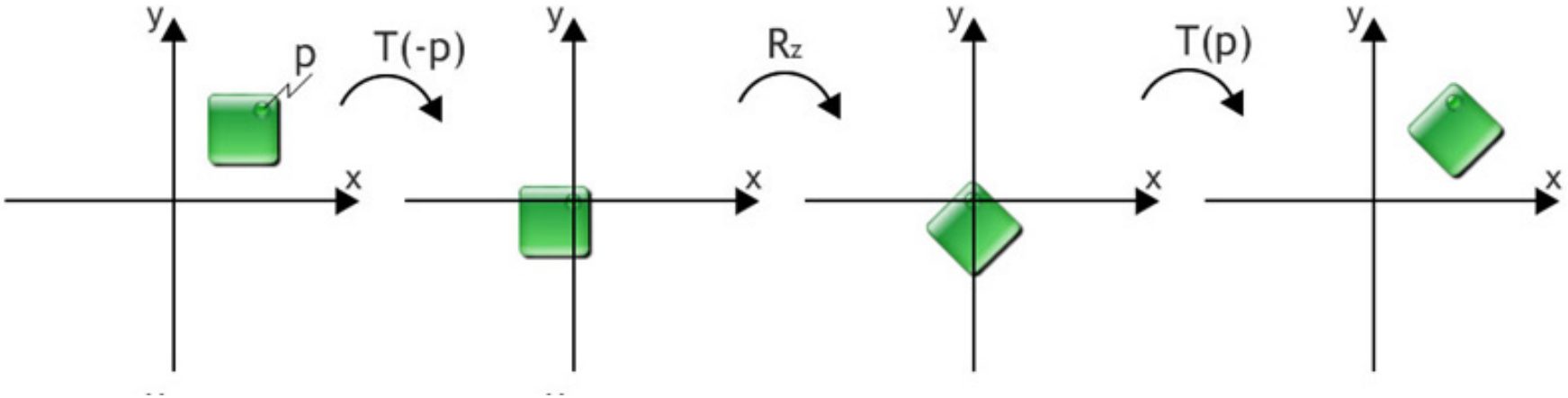
$$R_y(\phi) = \begin{pmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformações básicas

Rotação $R_z(\phi)$:

$$R_y(\phi) = \begin{pmatrix} \cos(\phi) & -\sin(\phi) & 0 & 0 \\ \sin(\phi) & \cos(\phi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformações básicas



Transformações básicas

void glRotate{**fd**}(**TYPE** $angle$, **TYPE** x, **TYPE** y, **TYPE** z);

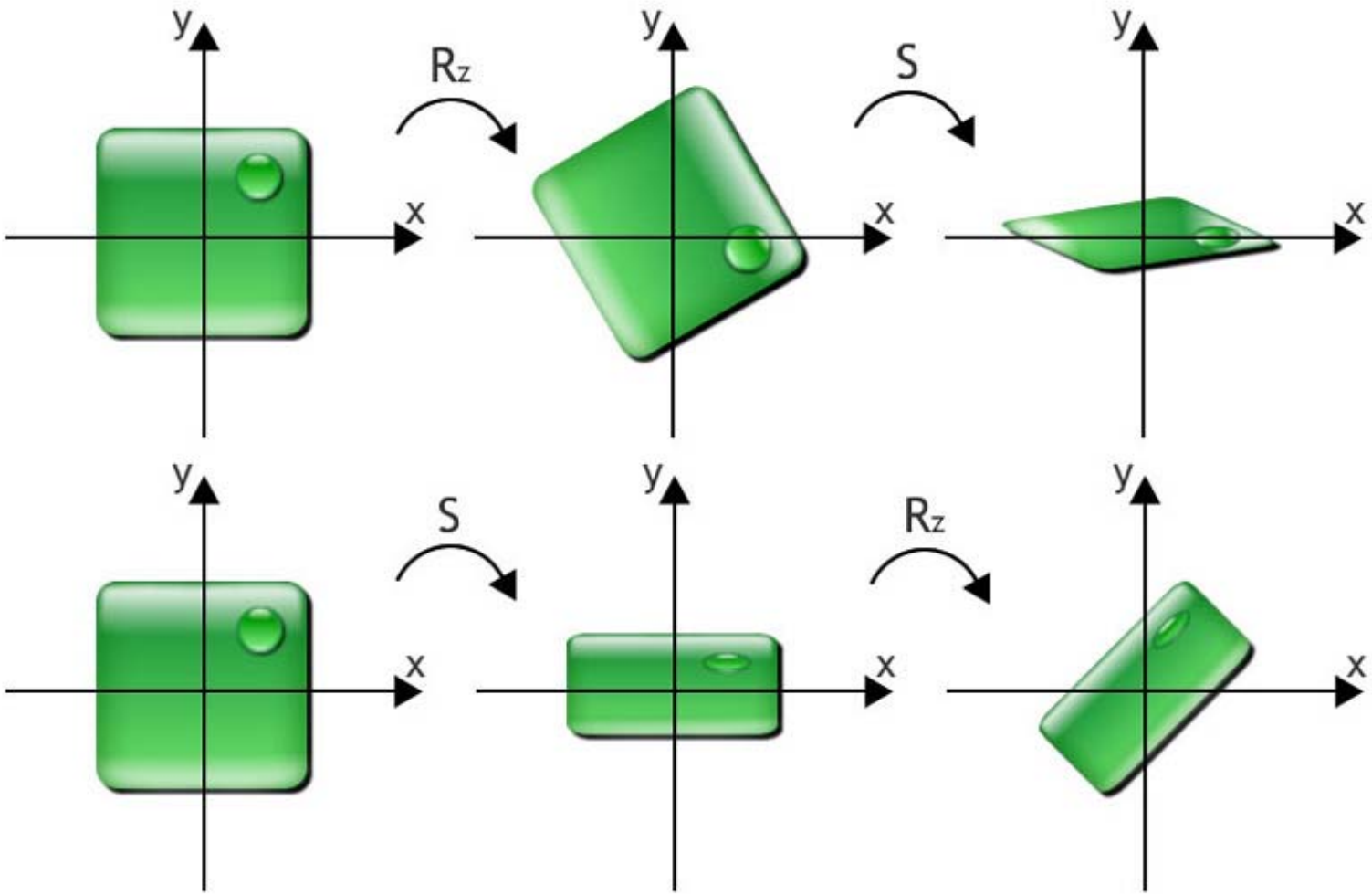
Multiplica a matriz corrente pela matriz de rotação, rotacionando o sistema de coordenadas local

Transformações básicas

Escala $S(s_x, s_y, s_z)$ **:**

$$S(s) = S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformações básicas



Transformações básicas

void glScale{**fd**}(**TYPE** x, **TYPE** y, **TYPE** z);

Multiplica a matriz corrente pela matriz de escala. As coordenadas x, y e z são multiplicadas pelos valores correspondentes

Transformando normais

- **Podemos utilizar matrizes para transformarmos pontos, linhas polígonos e outras geometrias**
- **Também podemos usar matrizes para transformar vetores**
- **Entretanto a matriz usada para transformar um objeto nem sempre pode ser usada para transformar a normal do mesmo**

Transformando normais

- **As normais devem ser transformadas pela matriz transposta da inversa da matriz usada para transformar o objeto**
- **Se a matriz usada para transformar o objeto é chamada de M, então precisamos utilizar uma matriz N para transformar as normais do objeto**

$$N = (M^1)^T$$

Transformando normais

- **Se uma matriz é ortogonal, então:**

$$M^1 = M^T$$

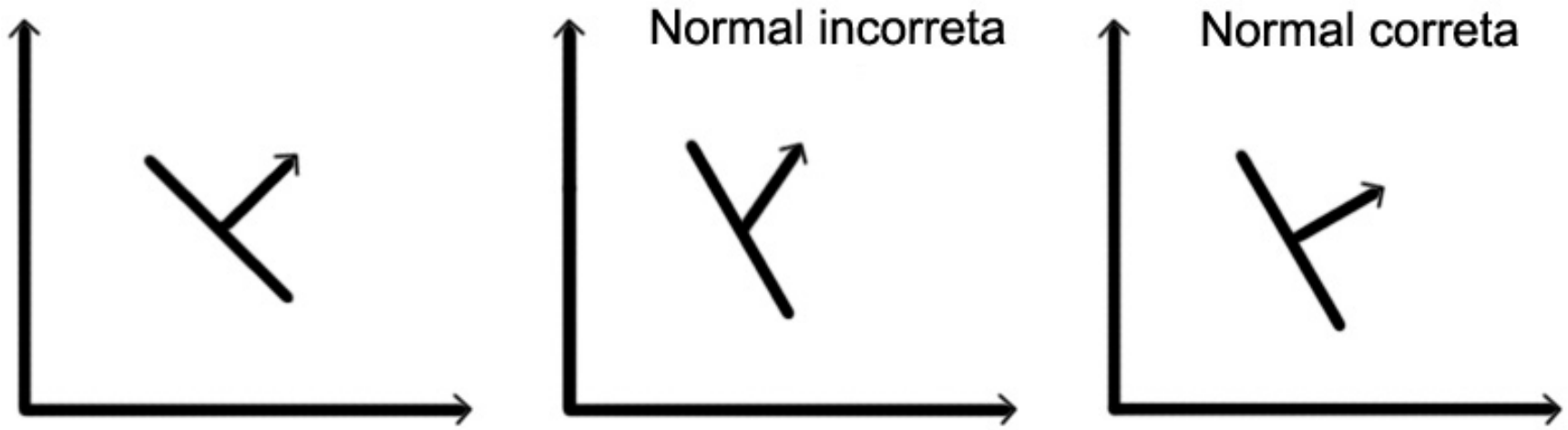
- **Se a matriz da transformação utilizada for ortogonal, por exemplo, formada apenas de rotações, não precisamos calcular sua inversa**

$$N = (M^T)^T$$

Transformando normais

- **Translações não afetam vetores, podendo ser usadas sem efeito sobre as normais**
- **Transformações de escala uniforme não precisam ter sua inversa calculada, pois afetam apenas o tamanho do vetor não sua direção**
- **No caso de uma escala uniforme ser aplicada precisamos apenas recalcular o tamanho da normal do vetor**
- **Em todos os demais casos a inversa da matriz da transformação aplicada precisa ser calculada**

Transformando normais



Concatenação de tranformações

- **A principal razão de concatenarmos uma seqüência de transformações em uma única é aumentar a performance da aplicação**
- **Utilizamos multiplicação de matrizes para concatenarmos as transformações**
- **Multiplicação de matrizes não é uma operação comutativa** $(A \times B \neq B \times A)$
- **A ordem em que as matrizes de transformação são concatenadas é muito importante**

Concatenação de tranformações

Executar exercício 1

Concatenação de tranformações

- **O estado final de um objeto depende da ordem em que as transformações foram aplicadas**
- **Quando aplicamos alguma transformação a um objeto, todos os vértices desse objeto são transformados pela matriz de transformação aplicada**

Concatenação de tranformações

- **Podemos aplicar uma transformação que seja a combinação de todas as transformações desejadas, ao invés de aplicarmos cada transformação em separado**

Concatenação de tranformações

- **A formula da matriz de transformação composta é: C = TRS**
- **Onde as transformações são aplicadas da direita para esquerda**
  - **T(R(S(p)))**
  - **Escala**
  - **Rotação**
  - **Translação**

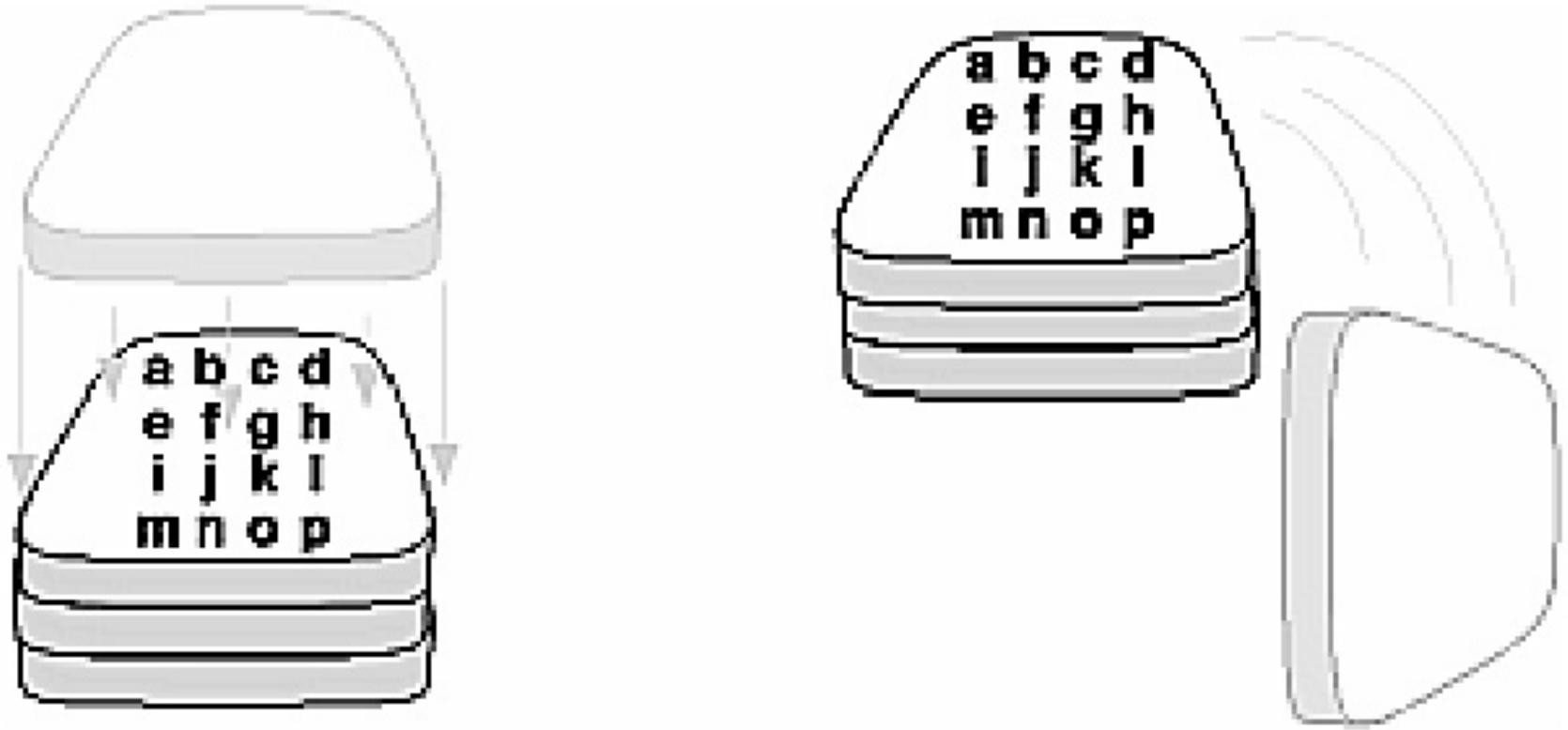
Utilizando pilhas de transformações

- **A matriz que estamos utilizando, seja ela de projeção, visão de modelo(Modelview) ou textura, na verdade é o elemento do topo de uma pilha de matrizes**
- **A pilha de matrizes é útil para contruirmos modelos hierárquicas**
- **Para salvar a configuração atual da cena, criamos uma cópia da matriz de transformação atual e colocamos essa nova matriz no topo da pilha**

Utilizando pilhas de transformações

- **Para restaurarmos a configuração da cena, descartamos a matriz que está no topo da pilha**
- **Utilizando pilhas de matrizes podemos salvar a configuração atual da cena, aplicar novas transformações em algum objeto, e em seguida restaurar a configuração da cena**

Utilizando pilhas de transformações



Utilizando pilhas de transformações

void glPushMatrix(void);

Empilha a matriz de tranformação corrente. A pilha utilizada é determinada pelo comando *glMatrixMode*

void glPopMatrix(void);

Desempilha uma matriz da pilha e a define como a matriz de tranformação corrente. A pilha utilizada é determinada pelo comando *glMatrixMode*

Projeções

- **Antes de um cena ser desenhada(rasterizada), todos os objetos relevantes a cena precisam ser projetados em alguma espécie de plano ou volume simples**
- **Após projetarmos a cena cortamos os objetos, ou partes dos objetos, que se encontram fora do plano ou volume de projeção**

Projeções

- **Utilizando projeções podemos**
  - **Definir o sistema de coordenadas a ser utilizado(bidimensional ou tridimensional)**
  - **Modificar os eixos utilizados**
  - **Simular o comportamento de vários tipos de câmeras(lentes)**
  - **Dentre vários outros**

Projeções

- **Projeção ortográfica**
  - **Uma característica da projeção ortográfica é que linhas paralelas continuam paralelas após a projeção**
  - **Na projeção ortográfica o volume de visão é um paralelepípedo**
  - **À distância dos objetos a câmera não afeta o seu tamanho**
  - **Este tipo de projeção é usada quando queremos preservar o tamanho dos objetos e o ângulo entre eles**

Projeções

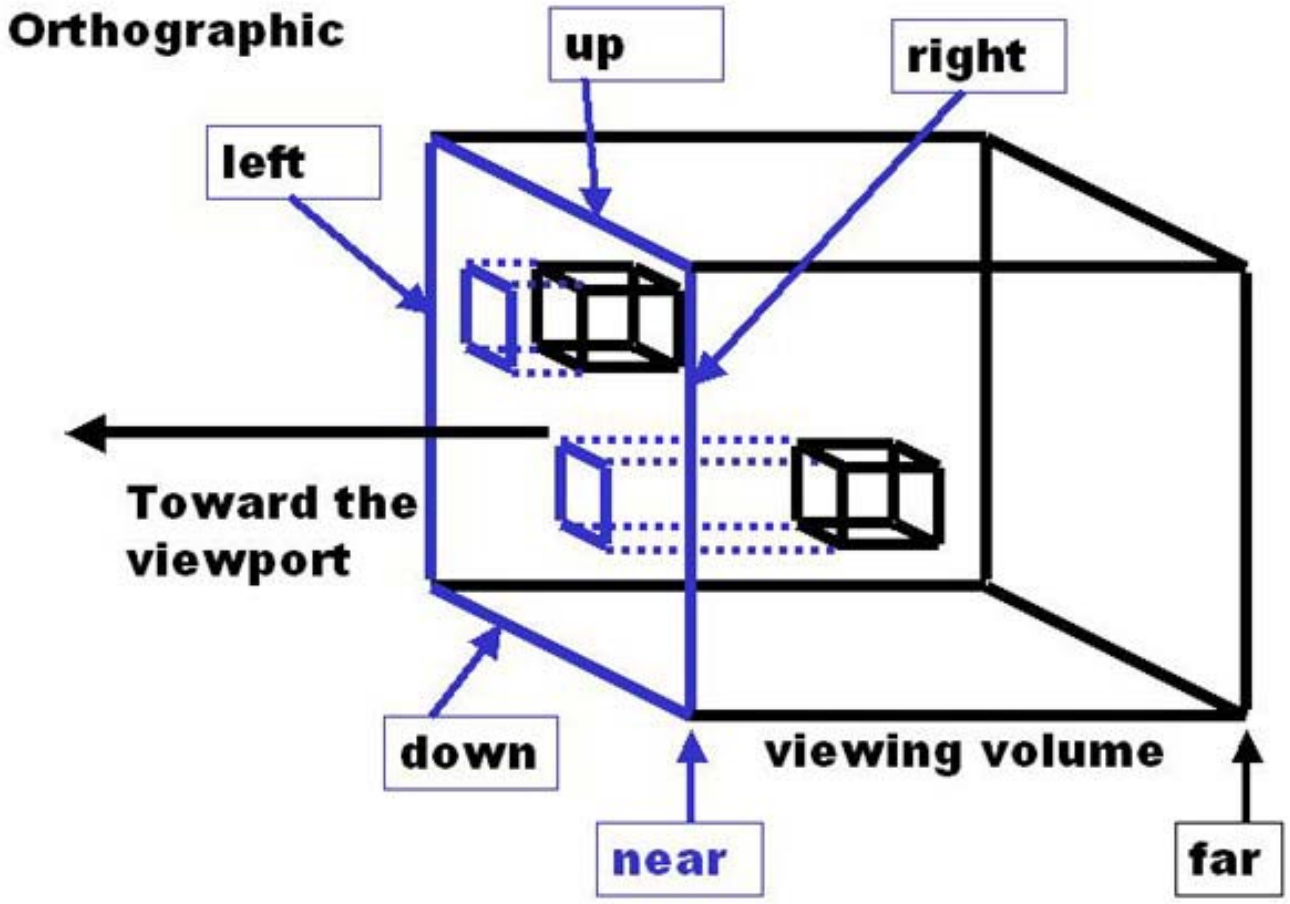
- **Projeção perspectiva**
  - **Usada na maioria das aplicações de computação gráfica**
  - **Linhas paralelas geralmente não são mais paralelas após a projeção**
  - **Na projeção perspectiva o volume de visão é um tronco de pirâmide**
  - **A principal característica da projeção perspectiva é que quanto mais longe os objetos estão da câmera eles aparecem menores**

Projeções

void glOrtho(**GLDouble** *left*, **GLDouble** *right*, **GLDouble** *bottom*, **GLDouble** *top*, **GLDouble** *near*, **GLDouble** *far*);

Cria uma matriz de projeção ortográfica e multiplica a matriz atual por ela. O ponto (*left***,** *bottom*, *-near*) **representa o canto inferior esquerdo e (***right***,** *top*, *-near*) **o canto superior direito mapeado no plano mais próximo.** *Near* **e** *far* **podem ser positivos, negativos ou até mesmo zero. Mas não podem ser o mesmo valor**

Projeções

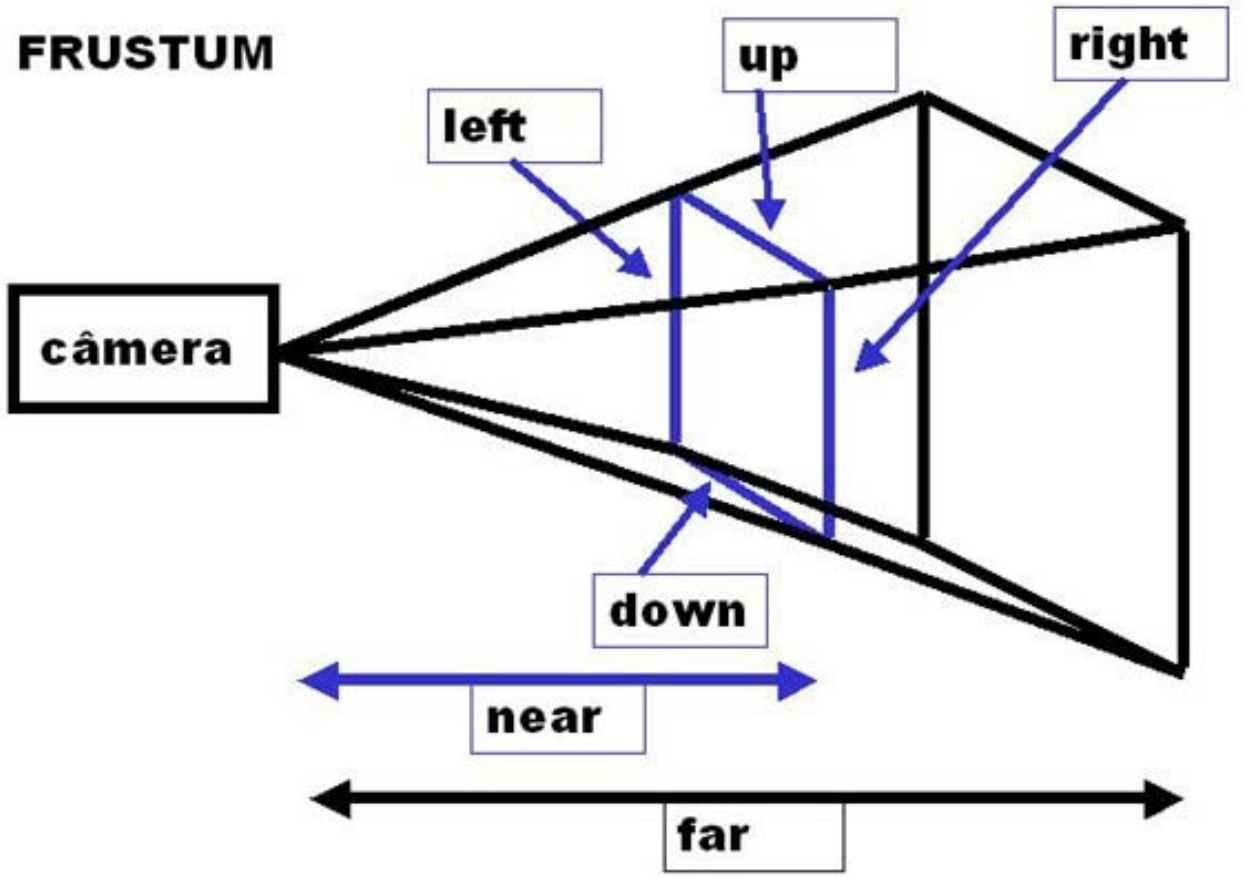

Orthographic
up
right
left
Toward the viewport
down
viewing volume
near
far

Projeções

void glFrustum(**GLDouble** *left*, **GLDouble** *right*, **GLDouble** *bottom*, **GLDouble** *top*, **GLDouble** *near*, **GLDouble** *far*);

Cria uma matriz de projeção perspectiva e multiplica a matriz corrente por ela. O ponto (*left*, *bottom*, *-near*) representa o canto inferior esquerdo e (*right*, *top*, *-near*) o canto superior direito mapeado no plano mais próximo. *Near* e *far* representam a distância do ponto de visão ao plano mais próximo e ao plano mais distante

Projeções



Projeções

void gluOrtho2D(**GLDouble** *left*, **GLDouble** *right*, **GLDouble** *bottom*, **GLDouble** *top*);

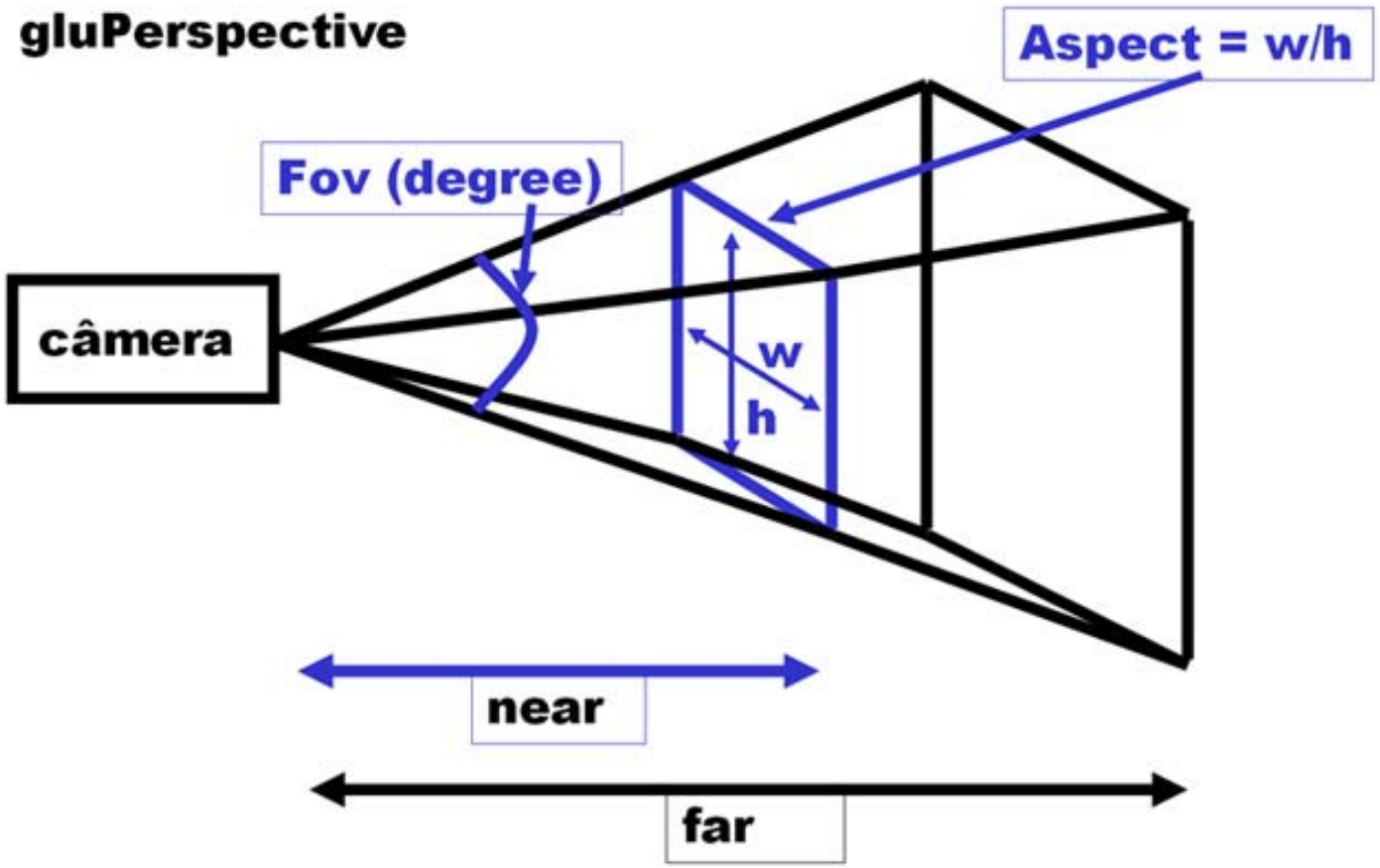
Cria uma matriz de projeção ortográfica e multiplica a matriz atual por ela. O ponto (*left*, *bottom*) representa o canto inferior esquerdo e (*right*, *top*) o canto superior direito

Projeções

void gluPerspective(**GLDouble** fov, **GLDouble** $aspect$, **GLDouble** $near$, **GLDouble** far);

Cria uma matriz simétrica de projeção perspectiva e multiplica a matriz corrente por ela. FOV **é o ângulo do campo de visão no plano yz, que deve estar entre** 0^0 **e** 180^0**.** $Aspect$ **é a largura da tela dividido pela sua altura.** $Near$ **e** far **representam a distância do ponto de visão ao plano mais próximo e ao plano mais distante**

Projeções



Tranformações de visão

- **Utilizamos a matriz de visão de modelo para posicionarmos e orientarmos nossos objetos**

Tranformações de visão

void gluLookAt(**GLDouble** *eyex*, **GLDouble** *eyey*, **GLDouble** *eyez*, **GLDouble** *centerx*, **GLDouble** *centery*, **GLDouble** *centerz*, **GLDouble** *upx*, **GLDouble** *upy*, **GLDouble** *upz*);

Cria uma matriz de visão e multiplica a matriz atual por ela. O ponto de visão é especificado por (*eyex*, *eyey*, *eyez*). O ponto que a camera está direcionada é especificado por (*centerx*, *centery*, *centerz*). Os valores (*upx*, *upy*, *upz*) indicam a direção que aponta para cima

Mapeamento da tela

- **O sistema de janelas é responsável pela abertura da janela na tela**
- **Por padrão a área de desenho é definida como toda a área da janela aberta**
- **Utilizamos o comando $glViewPort$ para definirmos uma área de desenho menor, podendo criar mais de uma área de desenho na tela**

Mapeamento da tela

void glViewPort(**GLint** x, **GLint** y, **GLsizei** $width$, **GLsizei** $hight$);

Define o retângulo de pixels na janela onde a imagem final será mapeada

Exercícios