

# Desenvolvimento de jogos tridimensionais com OpenGL



**Bruno Pereira Evangelista**

[bpevangelista@gmail.com](mailto:bpevangelista@gmail.com)

# **Aula 1**

**Introdução ao curso**

**Desenhando objetos simples**

# Introdução ao curso

- **Apresentação do curso**
- **Pré-requisitos**
- **Cronograma**
- **Exercícios**
- **Ferramentas utilizadas**
- **Considerações sobre hardware**
- **Referências**

# Jogos 3D

- **Do que um jogo 3D é feito ?**

- **Motor 3D (Gráfico, Física, IA, Outros)**

**Na parte gráfica geralmente são utilizadas bibliotecas como OpenGL ou Direct3D**

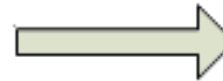
- **Roteiro/GamePlay**

- **Modelos/Texturas/Animações**

- **Efeitos sonoros/Musicas**

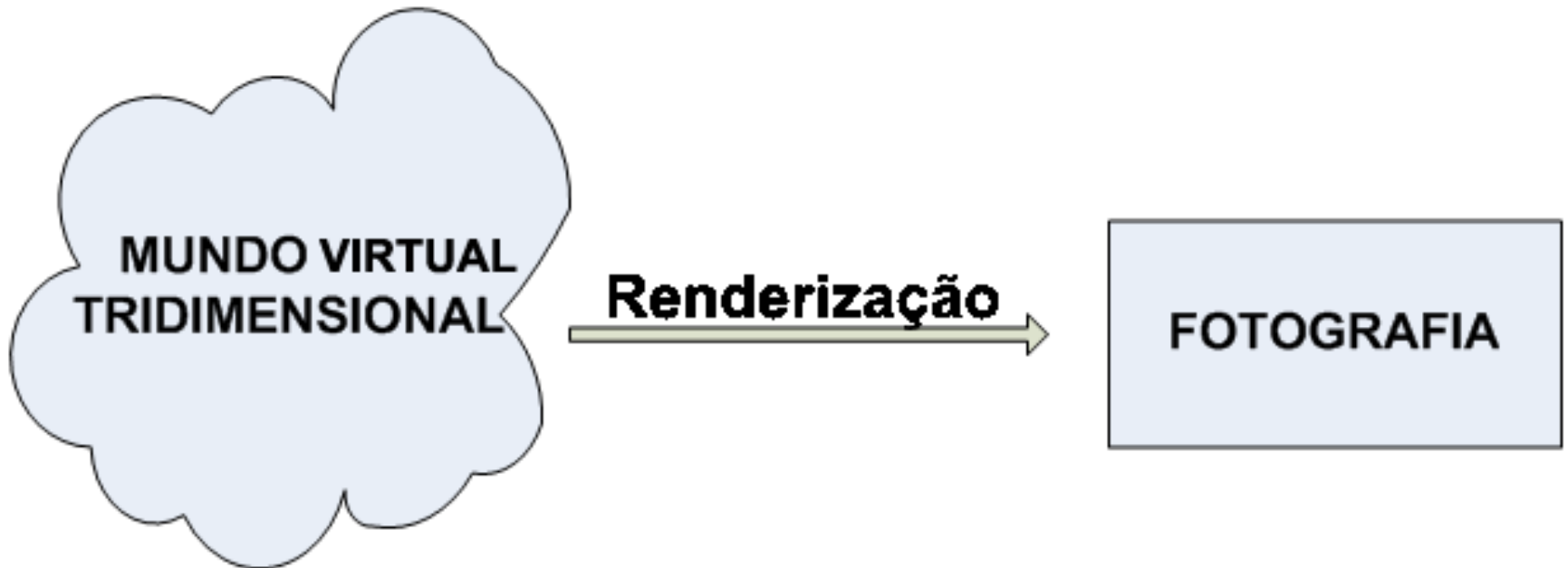
- **Vários outros ingredientes**

# Jogos 3D



# Gráficos 3D

- Como os gráficos 3D funcionam ?
- O que é “renderização” ?



# Bibliotecas gráficas

- **DirectX (Direct3D para gráficos)**
  - **Windows / XBox 360**
  - **Mais profissional (DXInput, DXMusic, ...)**
- **OpenGL**
  - **Windows / Linux / Mac / Playstation 3 / ...**
  - **Mais simples e intuitivo**
- **As duas possuem aceleração por hardware**

# Bibliotecas gráficas

- **Porque OpenGL ?**
  - **Multi-plataforma**
  - **Mais simples e intuitivo**
  
- **Onde é utilizado ?**
  - **Aplicações científicas**
  - **Software de modelagem (3D Max, Maya)**
  - **Jogos (Warcraft 3, Counter Strike, ...)**

# Linguagens

- **OpenGL possui implementações para várias linguagens**
  - **C/C++, C#, Java, dentre outras**
- **Porque utilizar C/C++ ?**
  - **Atualmente é a linguagem mais utilizada no desenvolvimento de jogos**
  - **Alto desempenho e permite otimizações de baixo nível**

# Primeiro aplicativo



# Passos para criação

- **Criar uma janela no sistema operacional utilizado**
- **Criar uma área (contexto) que o OpenGL utilizará para desenho (renderização)**
- **Configurar a área para o tipo de desenho desejado e desenhar um triângulo, contendo vértices de cores diferentes**
- **Atualizar o conteúdo da tela**

# PARTE PRÁTICA

- Criar um projeto console para Windows
- Criar um projeto janela para Windows
- Dificuldades para criação e gerenciamento manual de janelas (*JANELA*)
- Criar um projeto janela utilizando o GLUT (*GLUT1*)
- Utilizar o OpenGL para limpar a cor de fundo da janela (*GLUT2*)

# GLUT (OpenGL Utility Toolkit)

- **Biblioteca para a criação de aplicativos que utilizam OpenGL**
- **Implementa uma interface simples para criação e manipulação de janelas**
- **Multi-plataforma**
- **Simple, fácil e pequeno**
- **Ideal para aplicações de pequeno e médio porte**

# GLUT (OpenGL Utility Toolkit)

- Para utilizar o GLUT é necessário
  - Adicionar o cabeçalho da biblioteca GLUT “glut.h” ao diretório do projeto ou IDE
  - Incluir o cabeçalho da biblioteca “glut.h”
  - Adicionar a biblioteca estática “glut32.lib” ao diretório do projeto ou IDE
  - Adicionar a biblioteca dinâmica “glut32.dll” ao diretório do projeto ou ao diretório do sistema
- Mesmos passos para utilizar o OpenGL \*\*\*

# Limpendo a tela

- **Porque é necessário limpar a tela antes de desenhar?**
- **O *buffer* de cor geralmente contém a última cena desenhada**
- **O desenho atual pode não ocupar a tela inteira**

# Terminando o desenho

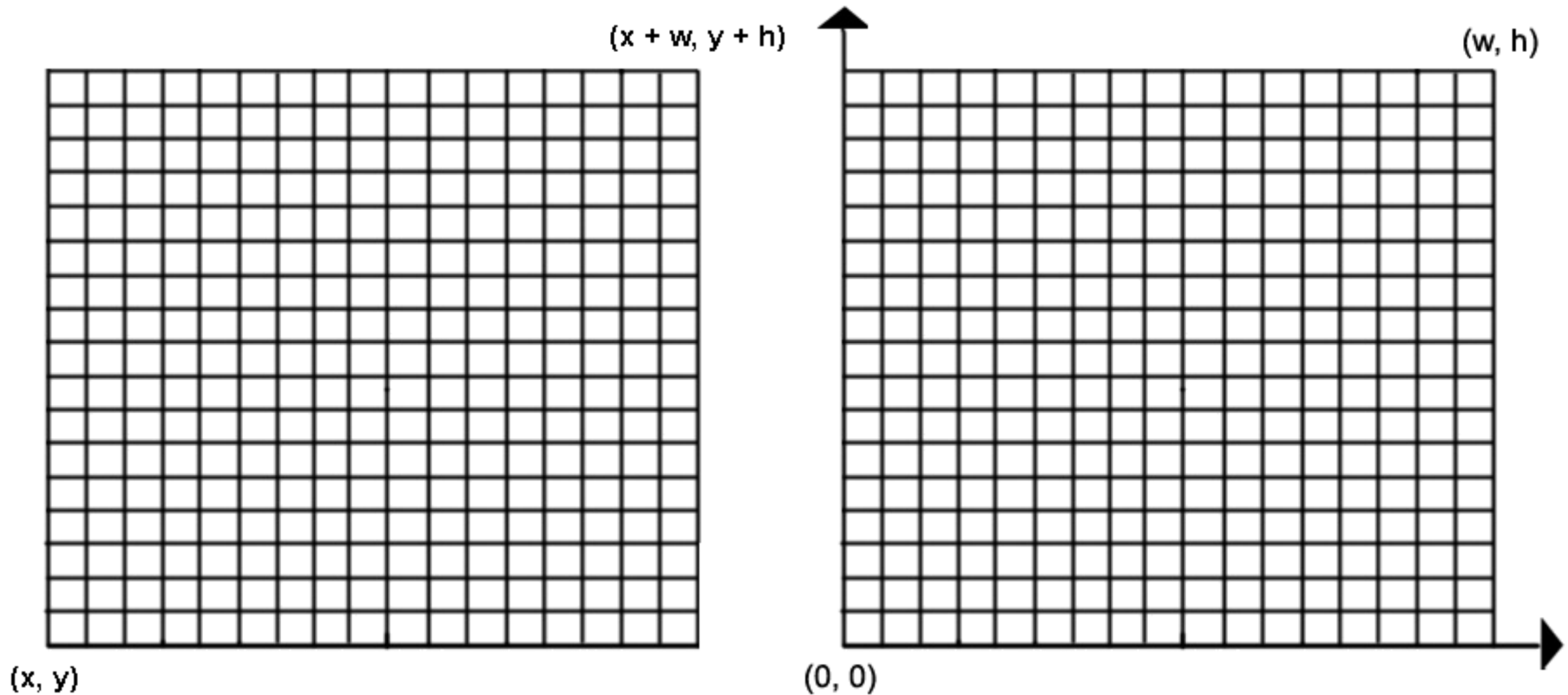
- Não há como descobrir quando o desenho de uma cena está completo
- Arquitetura Cliente/Servidor
- É necessário que o OpenGL seja informado que o desenho foi terminado, para forçá-lo a processar todos os comando anteriores, antes de começar um novo desenho

# PARTE PRÁTICA

- **Configurar uma área de desenho bidimensional e desenhar um triângulo (*GLUT3*)**
- **Definir cores diferentes para cada vértice do triângulo e configurar para as cores serem interpoladas (*GLUT4*)**

# Sistema de coordenadas 2D

- Utilizado quando se define uma visão bidimensional(paralela)



# PARTE PRÁTICA

- **Cria um novo projeto, configurar uma área de desenho bidimensional que se inicia na coordenada (0, 0) e terminando na coordenada (10, 10). Desenhar um polígono convexo e um quadrado (*GLUT5*)**

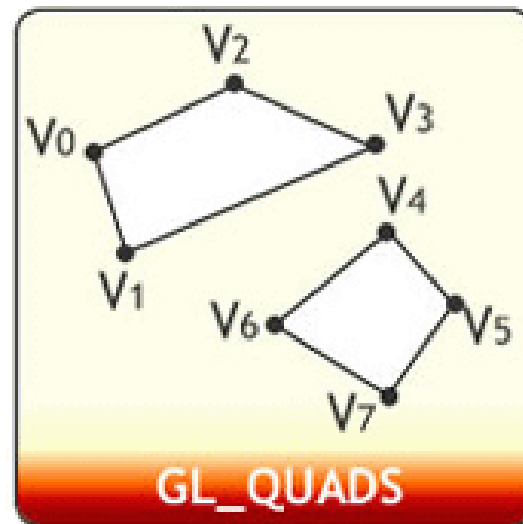
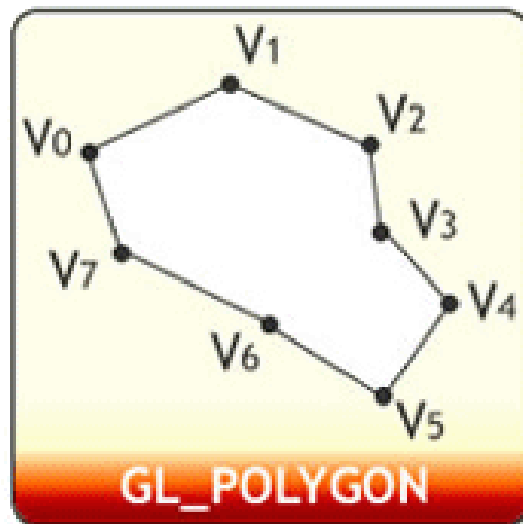
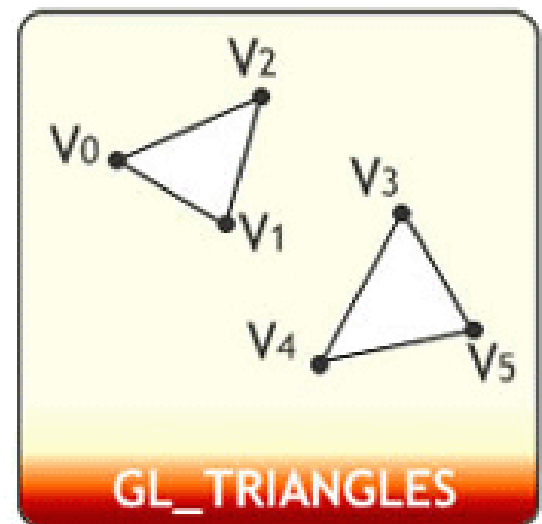
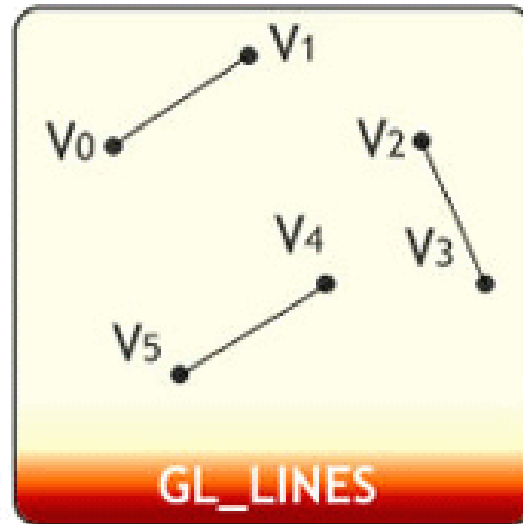
# Desenhando primitivas

- Para se desenhar primitivas geométricas com o OpenGL utiliza-se os comandos *glBegin* e *glEnd*
- *glBegin(...)* – Início da construção de uma primitiva de um tipo específico
- *glEnd()* – Término da construção da primitiva

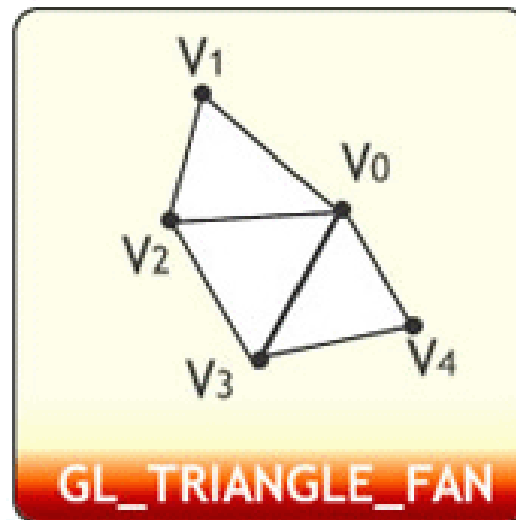
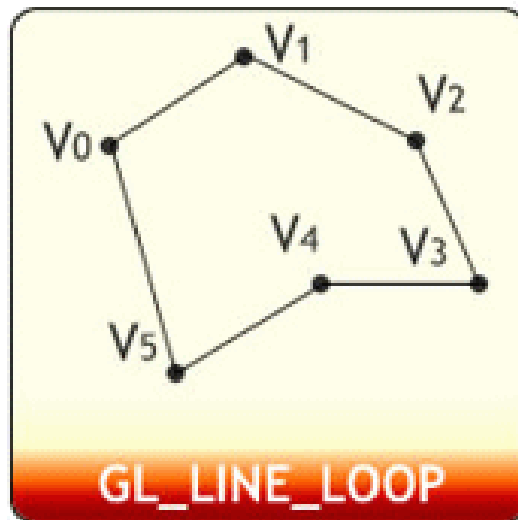
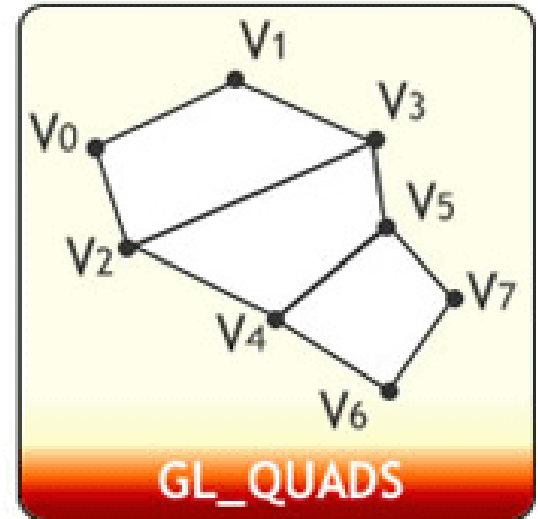
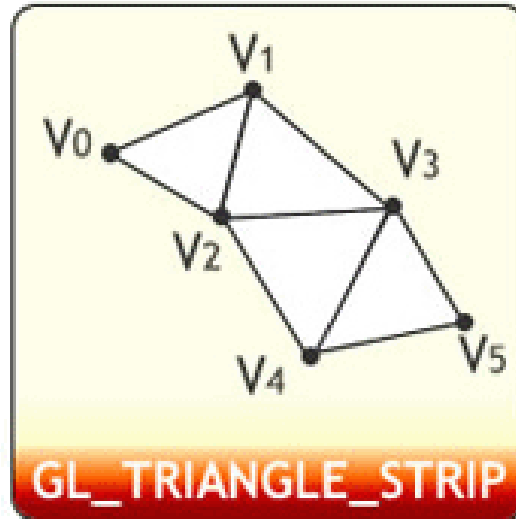
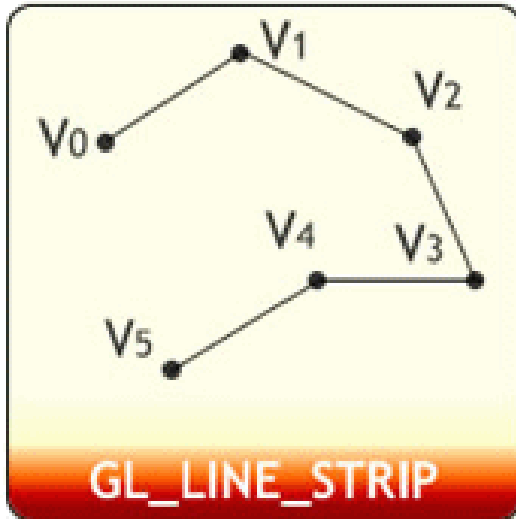
# Desenhando primitivas

- Entre os comandos *glBegin* e *glEnd* só podem existir comandos do OpenGL que especifiquem informações referentes a primitiva construída
- *glVertex\**, *glNormal\**, *glColor\**, vários outros

# Tipos básicos de primitivas



# Outros tipos de primitivas



# PARTE PRÁTICA

- **Configurar o OpenGL para limpar as transformações quando a janela for redeseñhada (*GLUT6*)**
- **Configurar o OpenGL para modificar a *viewport* da janela quando a mesma for redeseñhada (*GLUT7*)**
- **Corrige os problemas encontrados ao redimensionar a janela**

# Sintaxe do OpenGL

- Comandos utilizam o prefixo “gl”
  - *glClear, glBegin, glVertex, glFlush*
- Constantes utilizam o prefixo “GL\_”, e utilizam apenas letras maiúsculas
  - *GL\_TRIANGLES, GL\_LIGHTING*
- Alguns comandos utilizam o sufixo para especificar o tipo de argumento
  - *glVertex3f, glVertex3i, glVertex2d*

# Sintaxe do OpenGL

<b>Sufixo</b>	<b>Tipo de dado</b>	<b>Bits</b>	<b>ANSI C</b>	<b>OpenGL</b>
<b>b</b>	<b>Inteiro</b>	<b>8</b>	<b>char</b>	<b>GLbyte</b>
<b>ub</b>	<b>Inteiro s/ sinal</b>	<b>8</b>	<b>uchar</b>	<b>GLubyte</b>
<b>s</b>	<b>Inteiro</b>	<b>16</b>	<b>short</b>	<b>GLshort</b>
<b>us</b>	<b>Inteiro s/ sinal</b>	<b>16</b>	<b>ushort</b>	<b>GLushort</b>
<b>i</b>	<b>Inteiro</b>	<b>32</b>	<b>int</b>	<b>GLint</b>
<b>ui</b>	<b>Inteiro s/ sinal</b>	<b>32</b>	<b>uint</b>	<b>GLuint</b>
<b>f</b>	<b>Ponto-flutuante</b>	<b>32</b>	<b>float</b>	<b>GLfloat</b>
<b>d</b>	<b>Ponto-flutuante</b>	<b>64</b>	<b>double</b>	<b>GLdouble</b>
<b>v</b>	<b>Sufixo inserido após um dos listados acima. O tipo de dado é um ponteiro para um vetor.</b>			

# Outras bibliotecas

- **SDL (Simple DirectMedia Layer)**
  - **Utilizada para desenvolvimento de aplicativos multi-plataforma**
  - **Possui vários recursos: vídeo, entrada, som, temporizadores, outros**
  - **Um pouco mais complexa que o GLUT**

# Outras bibliotecas

- **GLU (OpenGL Utility)**
  - **Conjunto de funções que ajudam no desenvolvimento de aplicativos que utilizam OpenGL**
  - **Possui vários recursos: Câmera, textura(mip-mapping), NURBS, outros**

# Mesa 3D

- É uma implementação do OpenGL em software
- Baixo desempenho
- Não necessita de placas de vídeo com suporte 3D
- Possui suporte a *shaders*
- Utilizada onde não há suporte por hardware

# PARTE PRÁTICA

- **Tutorial do Nate Robins**
  - **Shapes (Desenhos 2D)**

# Revisão

- **GLUT - Gerenciamento de janelas**
  - **glutInit(...)**
  - **glutInitDisplayMode(...)**
  - **glutCreateWindow(...)**
  - **glutDisplayFunc(...)**
  - **glutReshapeFunc(...)**
  - **glutMainLoop(...)**
- **Esqueleto para manipulação de janelas, ainda será refinado. *Não é necessário decorar!!!***

# Revisão

- OpenGL – Funções utilizadas
  - glClearColor(...)
  - glClear(...)
  - glOrtho(...)
  - glBegin(...)
  - glEnd()
  - glFlush()

# Revisão

## OpenGL

- **glVertex\*(...)**      (**glBegin** <> **glEnd**)
- **glColor\*(...)**
- **glViewport(...)**
- **glLoadIdentity()**

# Exercícios

- Criar uma janela com o título “Exercício 1”, com cor de fundo azul escuro e desenhar uma estrela com 6 ou 7 pontas
- Criar uma janela com o título “Exercício 2”, com cor de fundo amarela e desenhar um círculo centralizado na tela. Use o tipo de primitiva `GL_TRIANGLE_FAN`
- Criar uma janela e desenhar quatro tipos diferentes de primitivas simultaneamente

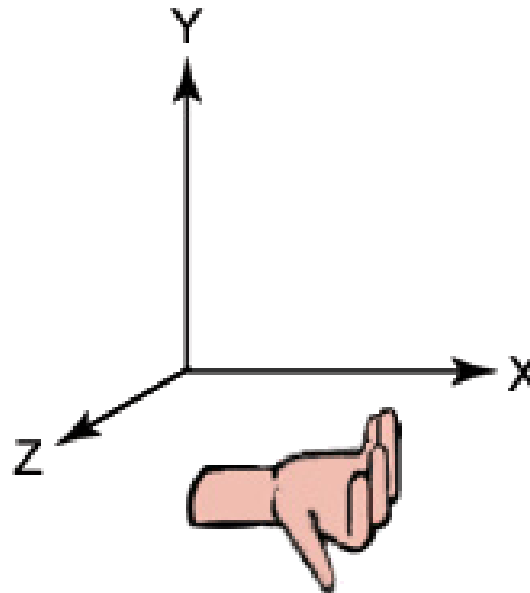
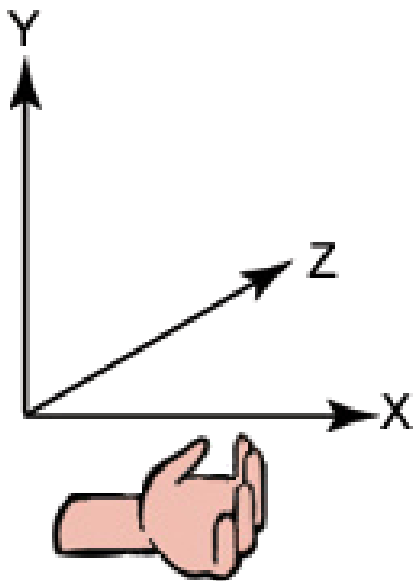


# **Aula 2**

**Desenhando sólidos tridimensionais**

# Sistema de coordenadas 3D

- Na computação gráfica, geralmente são utilizados dois sistemas de coordenadas 3D
  - Sistema de coordenadas da mão esquerda
  - Sistema de coordenadas da mão direita



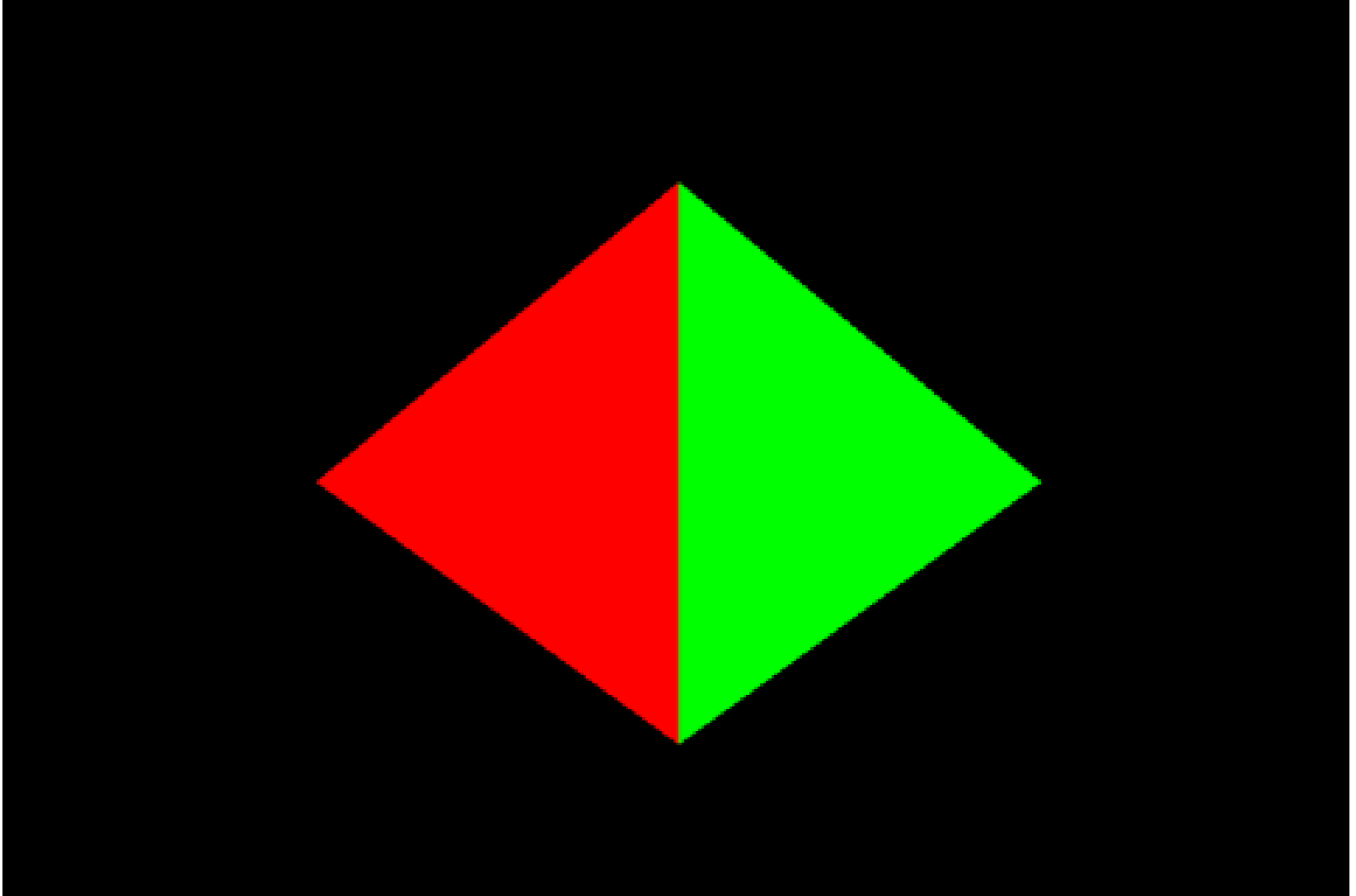
# Sistema de coordenadas 3D

- O sistema de coordenadas da mão direita é o mais utilizado, inclusive em cursos de matemática
- O OpenGL utiliza o sistema de coordenadas da mão direita, enquanto o DirectX utiliza o da mão esquerda

# Desenhando sólidos

- **Sólidos são formados a partir de primitivas geométricas**
  - **Um cubo é formado por 6 quadrados**
  - **Uma pirâmide é formada por um quadrado(base) e quatro triângulos**
- **Área de desenho precisa ser configurada para desenhos tridimensionais**
  - **Tratar profundidade da cena**

# Aplicativo - Sólido

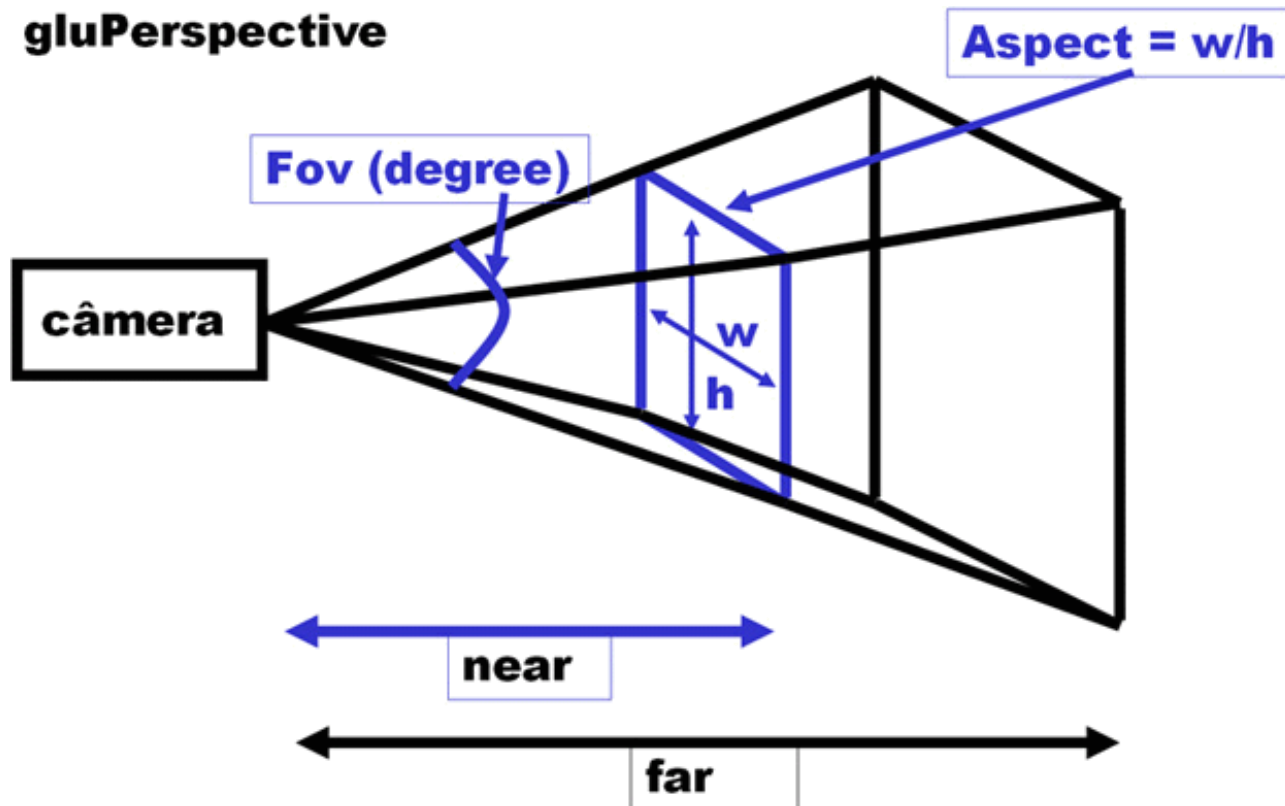


# Passos para criação

- Criar uma janela no sistema operacional utilizado e criar um contexto para o OpenGL (Esqueleto GLUT)
- Configurar uma área para desenho tridimensional
- Desenhar várias primitivas formando o sólido
- Atualizar o conteúdo da tela

# Configurando câmeras

- As câmeras tridimensionais são configuradas através de alguns parâmetros



# PARTE PRÁTICA

- Criar uma janela, configurar uma área de desenho tridimensional utilizando o comando *glFrustum* e desenhar um quadrado com profundidade (*SÓLIDO1*)
- Repetir a prática anterior, agora utilizando o comando *gluPerspective* (*SÓLIDO2*)

# Configurando câmeras

- A câmera possui uma posição fixa na origem da cena (0, 0, 0)
- Pode ser desejável observar a cena de outros pontos arbitrários
  - Topo da cena, lateral, outros
- Pode-se utilizar uma “câmera auxiliar” que permite visualizar a cena de qualquer ponto
  - `gluLookAt(...)`

# Hierarquia de transformações

- **As transformações no OpenGL seguem uma hierarquia**
  - **Mundo, Visão – GL\_MODELVIEW**
  - **Projeção – GL\_PROJECTION**
- **Permite modificar apenas uma parte específica da hierarquia desejada**
  - **Ao mover a câmera só alteramos a visão, mantendo a projeção inalterada**
- **Maiores detalhes serão vistos na aula 3**

# PARTE PRÁTICA

- Criar uma janela, configurar uma câmera tridimensional em um ponto arbitrário e desenhar um quadrado com profundidade (*SÓLIDO3*)

# Gerenciamento de estados

- O OpenGL possui vários estados que podem ser habilitados/desabilitados, dentre outros
- Quando um estado é alterado pelo usuário ele permanece inalterado até que o usuário o altere novamente
- Existe uma configuração inicial padrão para todos os estados

# Gerenciamento de estados

- O OpenGL possui vários estados que podem ser habilitados/desabilitados, dentre outros
- Quando um estado é alterado pelo usuário ele permanece inalterado até que o usuário o altere novamente
- Existe uma configuração inicial padrão para todos os estados

# Teste de profundidade

- Responsável em garantir que os objetos sejam desenhados corretamente de acordo com sua distância da câmera
- Realiza um teste de distância para cada *pixel*, não cada primitiva

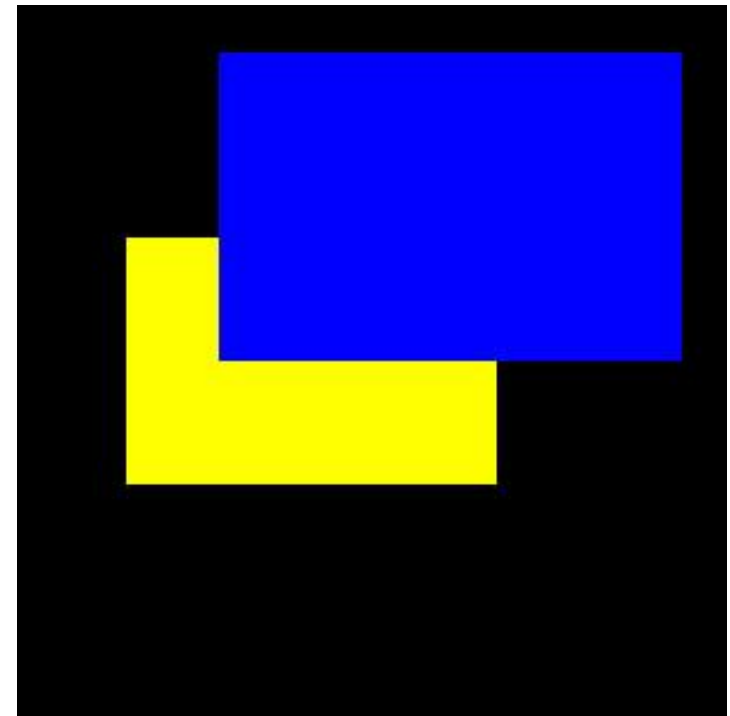
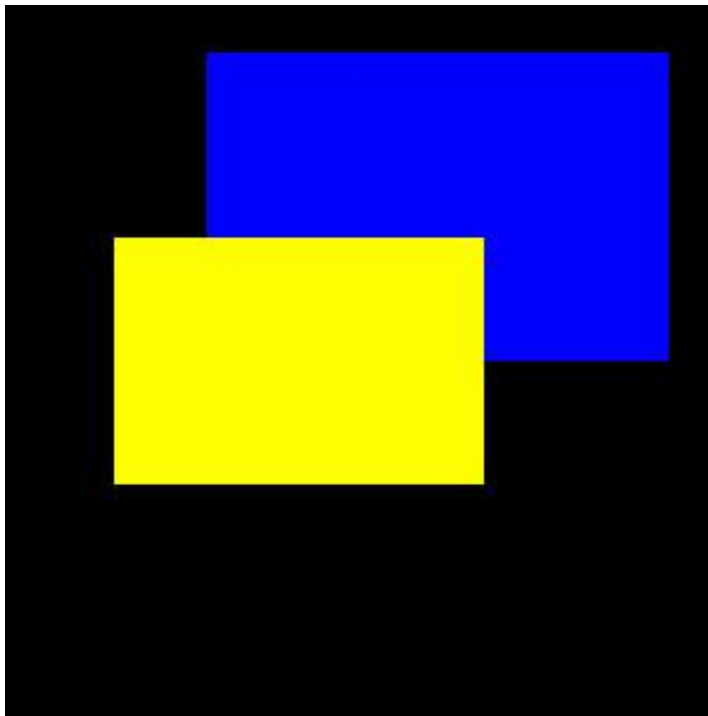


# PARTE PRÁTICA

- Criar uma janela, configurar uma câmera tridimensional e desenhar dois quadrados no plano XY, com profundidades diferentes e sem se sobreporem completamente. O plano mais próximo deve ser desenhado primeiro (*SÓLIDO4*)
- Repetir a prática anterior adicionando “*buffers* de profundidade” a cena (*SÓLIDO5*)

# Teste de profundidade

O quadrado amarelo está mais distante da câmera, mas é desenhado primeiro



**USO DO BUFFER DE PROFUNDIDADE**

# PARTE PRÁTICA

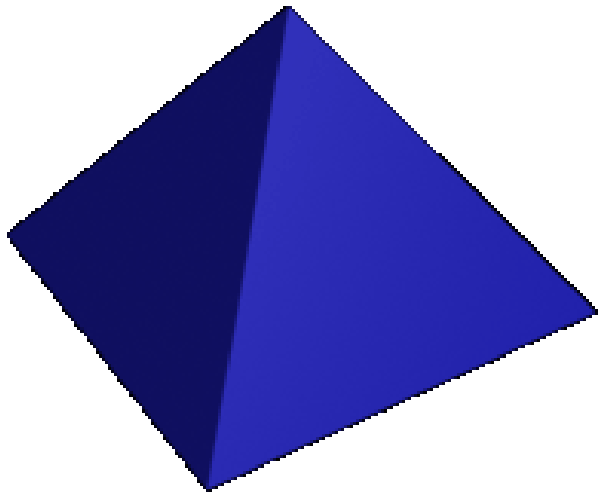
- **Criar uma janela, repetir as configurações anteriores e desenhar um sólido do tipo pirâmide (*SÓLIDO6*)**

# Seleção de faces visíveis

- A face de um polígono que têm seus vértices definidos em sentido anti-horário é denominada “face da frente”
- Este é o padrão utilizado pelo OpenGL, mas pode ser modificado
- De forma análoga, a outra face do polígono é denominada “face de trás”
- **PROBLEMA: “O OpenGL está com defeito, não desenha meu sólido corretamente!!!”**

# Seleção de faces visíveis

- Em sólidos opacos, as faces de trás (interiores ao sólido) nunca estão visíveis, mas sempre são desenhadas
- Descartar as faces ocultas pode aumentar o desempenho do aplicativo



**Pirâmide:**

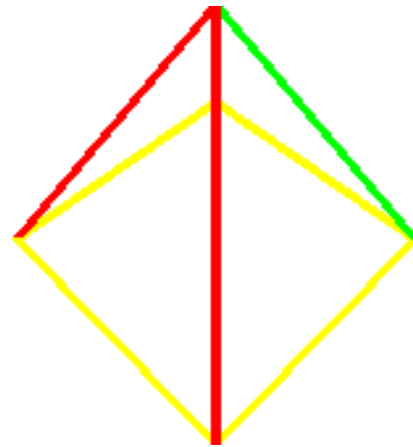
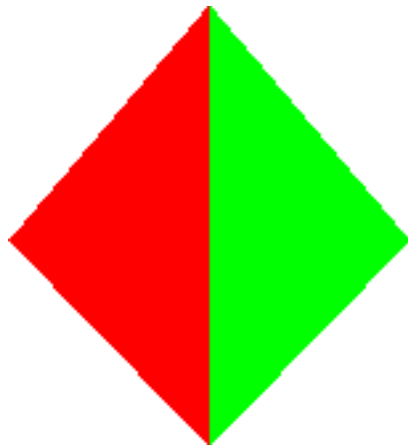
**6 faces** utilizando seleção de faces visíveis, ou **12 faces** sem utilizar seleção

# PARTE PRÁTICA

- **Repetir a prática anterior adicionando seleção de faces visíveis a cena (*SÓLIDO7*)**
- **Utilizar as funções presentes no GLUT para desenhar os seguintes sólidos: Cubo, Esfera, Chaleira e Donut (*SÓLIDO8*)**

# Modos de desenho

- As faces de um polígono podem ser desenhadas de maneira diferente
  - Pontos, Linhas ou Preenchida
- Por padrão todas as faces são desenhadas preenchidas



# Vertex Array

- **Possibilita especificar várias informações relacionadas a vértices através de arranjos**
- **Facilidade, não são necessários muitos comandos para desenhar sólidos**
- **Pode melhorar consideravelmente o desempenho**

# PARTE PRÁTICA

- **Tutorial do Nate Robins**
  - **Projection (Câmera 3D)**

# Revisão

- **GLUT – Funções utilizadas**
  - **glutSolidCube(...)**
  - **glutSolidTeapot(...)**
  - **glutSolidSphere(...)**
  - **glutSolidTorus(...)**

# Revisão

- OpenGL e GLU – Funções utilizadas
  - glMatrixMode(...)
  - glFrustum(...)
  - gluPerspective(...)
  - gluLookAt(...)
  - glEnable(...)
  - glDisable(...)     *Empírico*

# Revisão

- OpenGL e GLU – Funções utilizadas

- glClearDepth(...)      *GL\_DEPTH\_BUFFER\_BIT*

- glDepthFunc(...)

- glCullFace(...)      *GL\_CULL\_FACE*

# Exercícios

- **Desenhar um sólido do tipo cubo com cores únicas para cada vértice (sem utilizar GLUT). Usar seleção de faces para remover as faces internas do sólido e posicionar a câmera de modo que proporcione uma visão perspectiva da cena**
- **Modificar o modo de desenho do exercício anterior para que apenas as linhas que contornam o sólido sejam visíveis**

# Exercícios

- **Desenhar dois sólidos com profundidades diferentes, onde parte dos sólidos é sobreposta. Modificar o teste de profundidade para que o sólido mais distante da câmera seja desenhado na frente do mais próximo**



# **Aula 3**

**Transformações**  
**Entrada de dados**

# Transformações

- São a ferramenta básica para manipularmos geometrias
- Importante seu entendimento
- São aplicadas através de matrizes
- `glMatrixMode`, `glLoadMatrix*`
- O OpenGL possui várias funções para trabalhar com transformações

# Transformações

- **Translação**
  - **Modifica a posição**
- **Rotação**
  - **Modifica o ângulo**
- **Escala**
  - **Modifica o tamanho**
- **As transformações são aplicadas sobre o sistema de coordenadas local, afetando todos os objetos que são desenhados posteriormente**

# PARTE PRÁTICA

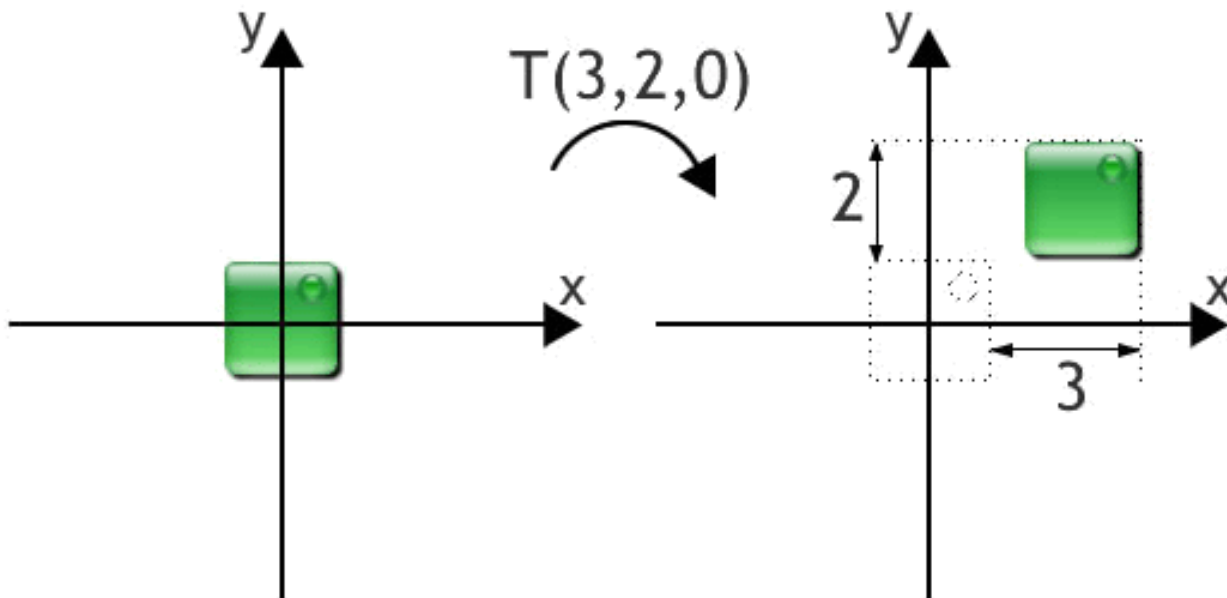
- Criar um método que desenhe um cubo ao redor do ponto  $(0, 0, 0)$ , com diferentes cores de faces. Utilizar esse método para desenhar vários cubos em diferentes posições da cena (*TRANSFORMA1*)

***OBS: Não esqueça de limpar as transformações***

- Desenhar um cubo utilizando o método anterior e tratar eventos relacionados a entrada de dados pelo teclado para rotacioná-lo em torno dos eixos X e Z (*TRANSFORMA2*)

# Translação

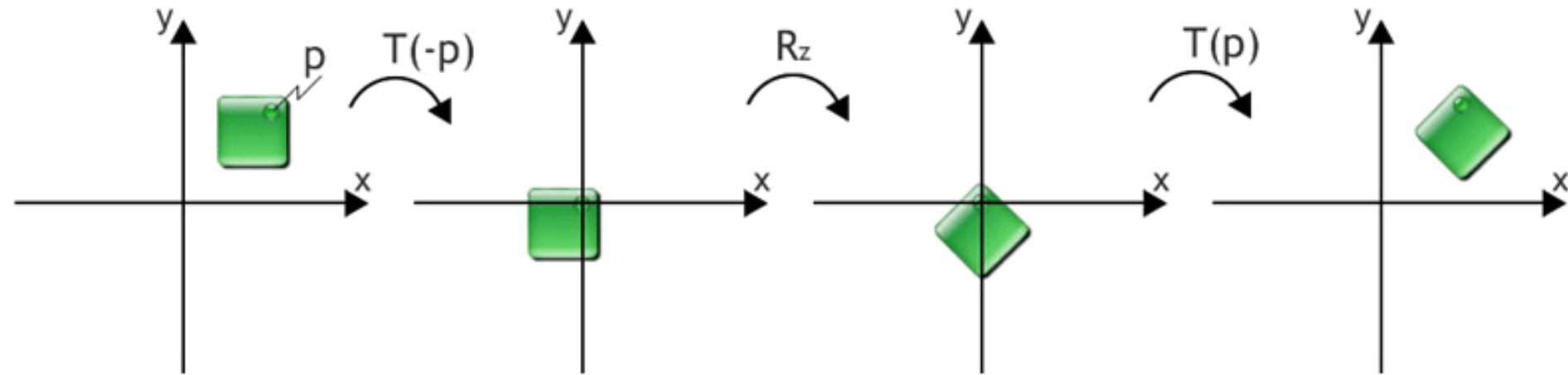
- Translada a posição do eixo em que os objetos são desenhados



**EFEITO SIMILAR A TRANSLADAR O OBJETO**

# Rotação

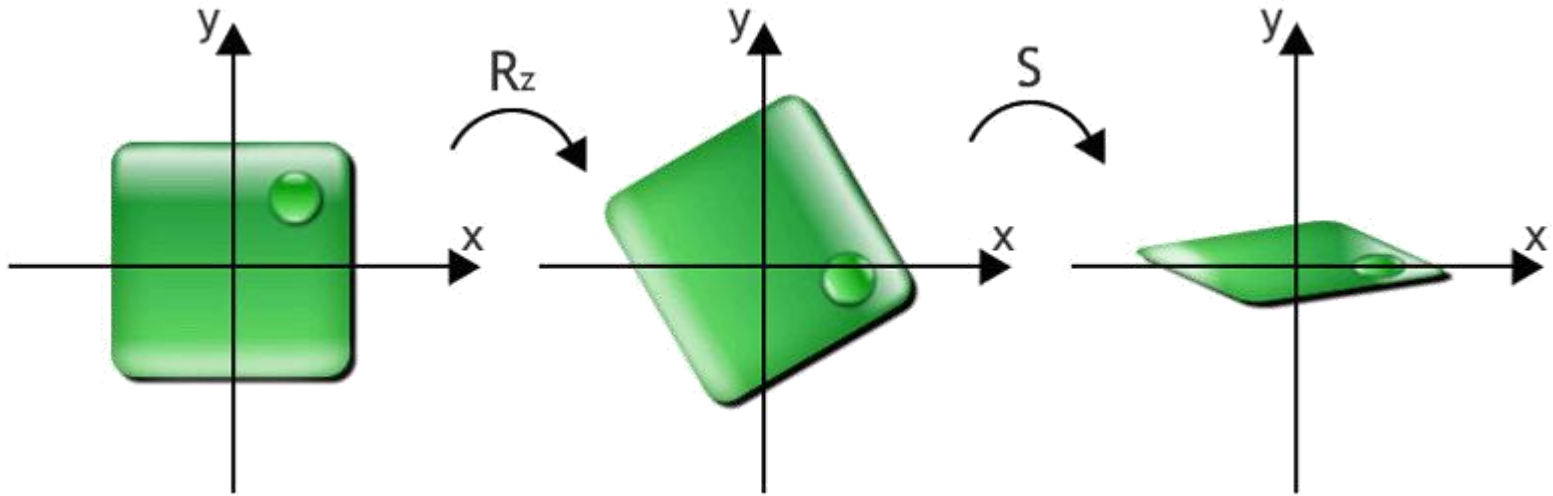
- Rotaciona o eixo em que os objetos são desenhados, em torno da posição  $(0,0,0)$



**EFEITO SIMILAR A ROTACIONAR O OBJETO**

# Escala

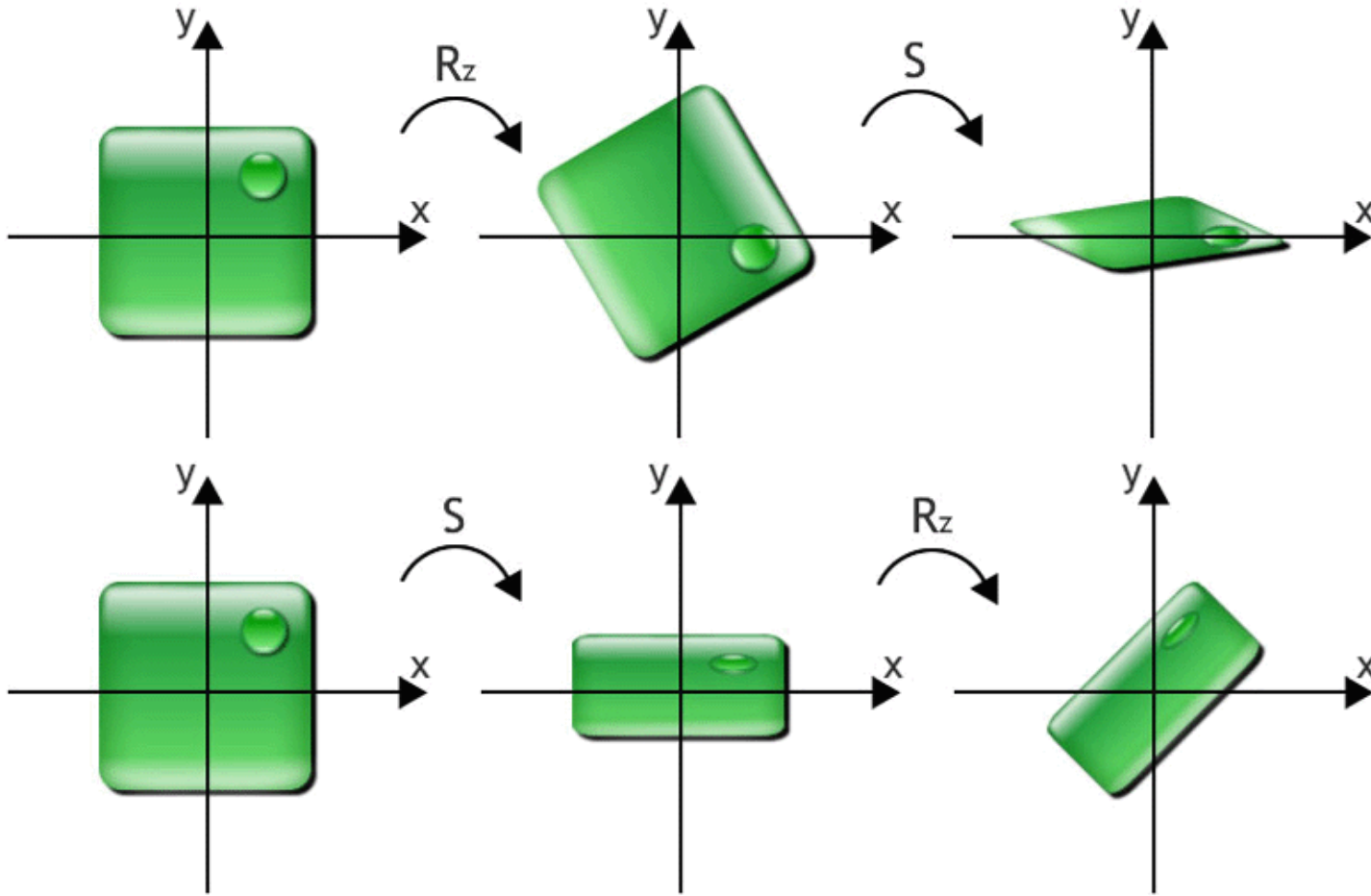
- Escala o eixo em que os objetos são desenhados, em torno da posição (0,0,0)



**EFEITO SIMILAR A MODIFICAR O TAMANHO DO OBJETO**

# Transformações

- Transformações não são comutativas



**Rotação  
+  
Escala**

**≠**

**Escala  
+  
Rotação**

# Transformações

- Para a concatenação ser aplicada corretamente ela deve seguir o modelo
  - $C = TRS$
- As transformações são aplicadas da direita para a esquerda
  - $C = T(R(S(p)))$
  - Direita para esquerda: Escala, Rotação e Translação

# PARTE PRÁTICA

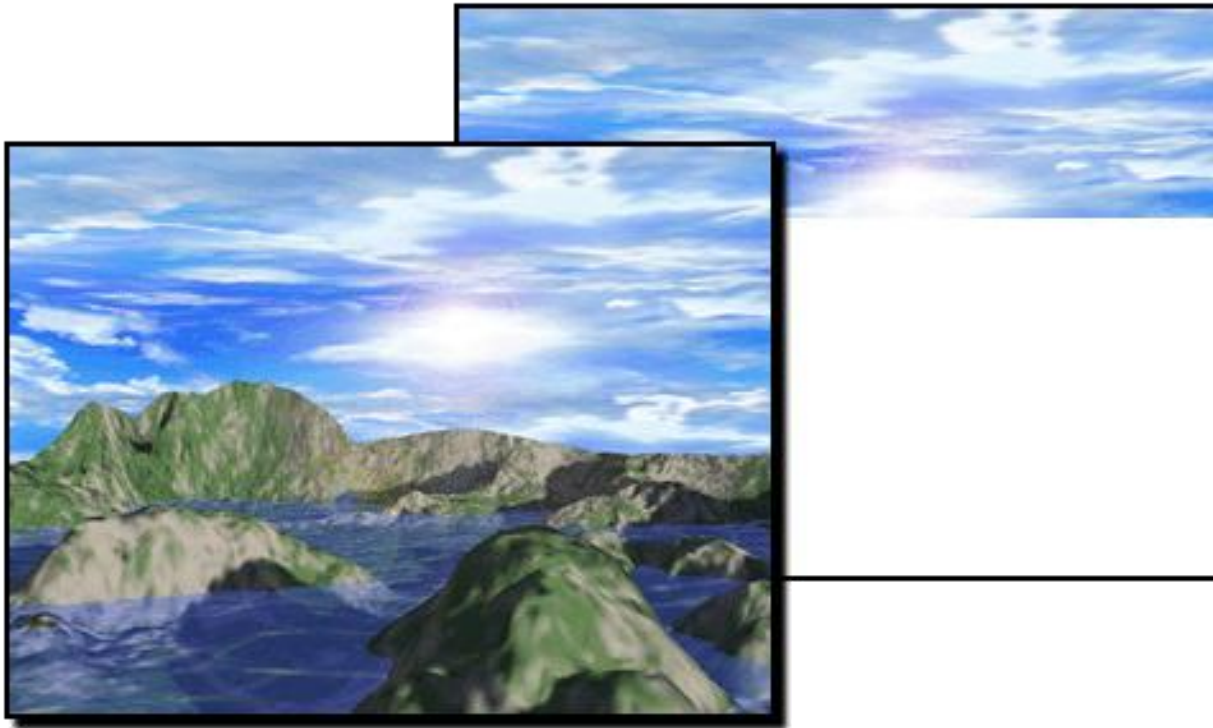
- **Adicionar a prática anterior eventos relacionados a entrada de dados pelo teclado para tratar translações nos eixos X e Z e escala uniforme (*TRANSFORMA3*)**

# Animação

- Para criar o efeito de animação é preciso modificar a cena e redesenhá-la rapidamente
- **Animação = Redesenhar + Trocar desenhos**
- Quando utiliza-se uma única área de desenho, as novas cenas são desenhadas sobre as cenas anteriores
- Pequenos defeitos na tela

# Animação

- Utilizando duas ou mais áreas (*buffers*) de desenho, pode-se eliminar esse problema
- Front Buffer (visível), Back Buffer (invisível)



# PARTE PRÁTICA

- **Adicionar a prática anterior o tratamento de evento idle, chamado quando o aplicativo está desocupado. Criar um contador de FPS e exibi-lo no console (*TRANSFORMA4*)**
- **Modificar a prática anterior para utilizar dois *buffers* para desenho (*TRANSFORMA5*)**

# Pilha de transformações

- **As transformações de mundo e projeção utilizam pilhas**
- **Pode-se facilmente salvar ou restaurar uma transformação a qualquer momento**
- **Para aplicar transformações**
  - **Salva-se o estado atual e aplica-se as transformações desejadas**
  - **Restaura o estado inicial**
- **Não é necessário se preocupar com as transformações anteriores antes de desenhar**

# Mistura de projeções

- **É possível utilizar diferentes projeções para desenhar uma cena**
- **Geralmente o desenho segue a ordem**
  - **Projeção perspectiva (câmera 3D), para desenhar a cena**
  - **Projeção paralela (câmera 2D) para desenhar interface de usuário, e escrever informações da cena**

# PARTE PRÁTICA

- Criar uma função que escreve na janela do OpenGL utilizando GLUT. Utilizar essa função para escrever o FPS na janela (*TRANSFORMA6*)

**OBS: Será necessário misturar os modos de visualização 2D e 3D**

# Escrita 2D

- O GLUT possui funções para auxiliar na escrita 2D, mas também existem outras bibliotecas para esse processo
- Geralmente são necessários vários passos para escrever manualmente
- O OpenGL possui funções que auxiliam o tratamento do espaço bidimensional
  - Similares as transformações tridimensionais
  - `glRasterPos*`

# PARTE PRÁTICA

- **Tutorial do Nate Robins**
  - **Transformation (Transformações)**

# Revisão

- **GLUT – Funções utilizadas**
  - **glutIdleFunc(...)**
  - **glutKeyboardFunc(...)**
  - **glutSpecialFunc(...)**
  - **glutSwapBuffers()**
  - **glutBitmapCharacter(...)**
  - **glutPostRedisplay()**
  - **glutBitmapCharacter(...)**

# Revisão

- **OpenGL e GLU – Funções utilizadas**
  - **glTranslatef(...)**
  - **glRotatef(...)**
  - **glScalef(...)**
  - **glPushMatrix()**
  - **glPopMatrix()**
  - **glRasterPos\*(...)**

# Exercícios

- **Desenhar um braço de robo com um ou duas junções. Adicionar eventos do teclado para tratar translação e rotação de cada parte do braço**

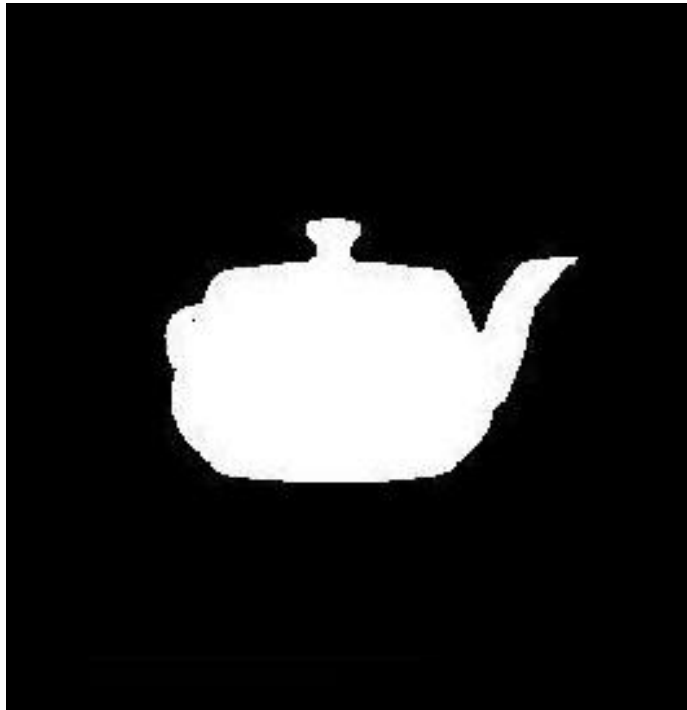


# **Aula 4**

## **Luzes e Materiais**

# Luzes

- São utilizadas para adicionar maior realismo as cenas
- Criam o efeito de profundidade da cena



# Luzes no mundo real

- **Complexa de ser descrita**
  - **Modelos de partícula e onda**
- **Modelo de partícula**
  - **A luz emite partículas em direção a uma superfície, que pode absorve-la ou refleti-la**
  - **Superfícies diferentes tem propriedades diferentes**
  - **Modelo utilizado pelo OpenGL (*Aproximado*)**

# Luzes no OpenGL

- Possui um modelo de luz local
  - Intensidade ambiente, reflexão e cor especular e cálculo de em dois lados
- Possui 8 luzes no *pipeline* padrão, que podem ser configuradas
  - Inicialmente todas estão desabilitadas
  - Possui os componentes ambiente, difuso e especular
  - Cada componente é representado por uma cor no sistema RGB

# Materiais no OpenGL

- **Permite definir um material que será utilizado no desenho dos objetos**
- **Especifica propriedades da superfície**
- **Possui os componentes ambiente, difuso, especular, emissivo e brilho (*shininess*)**
- **A cor final do objeto depende das luzes que incidem sobre ele, e do material de sua superfície**
- **Não esquecer de especificar as normais dos vértices para cada objeto**

# Passos para criação

- Criar uma janela no sistema operacional utilizado e um contexto para o OpenGL
- Configurar a área de desenho tridimensional com buffer profundidade
- Criar e habilitar uma luz
- Definir material para o objeto
- Desenhar sólido, especificando as normais para cada vértice
- Trocar os *buffers*

# PARTE PRÁTICA

- **Desenhar um sólido utilizando a biblioteca GLUT, e definir um material para o mesmo. Adicionar uma luz direcional a cena, com componentes ambiente e difusa (*LUZ1*)**
- **Modificar o exemplo anterior para criar um cubo, com normais especificadas por vértices (sem utilizar a biblioteca GLUT). Rotacionar o cubo constantemente em torno do eixo Y (*LUZ2*)**

# Cálculo de iluminação

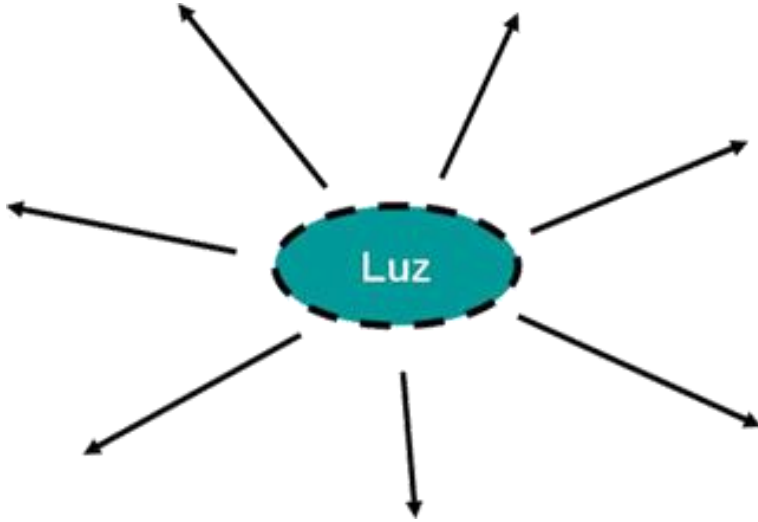
- A cor das superfícies é calculada para cada vértice e interpolada
- Qualidade da luz depende de superfícies detalhadas (muitos triângulos)
- Modelo utilizado pelo *pipeline* padrão do OpenGL, conhecido como sombreamento do Gouraud
- Muitas luzes provocam uma grande redução no desempenho do aplicativo

# Tipos de luzes

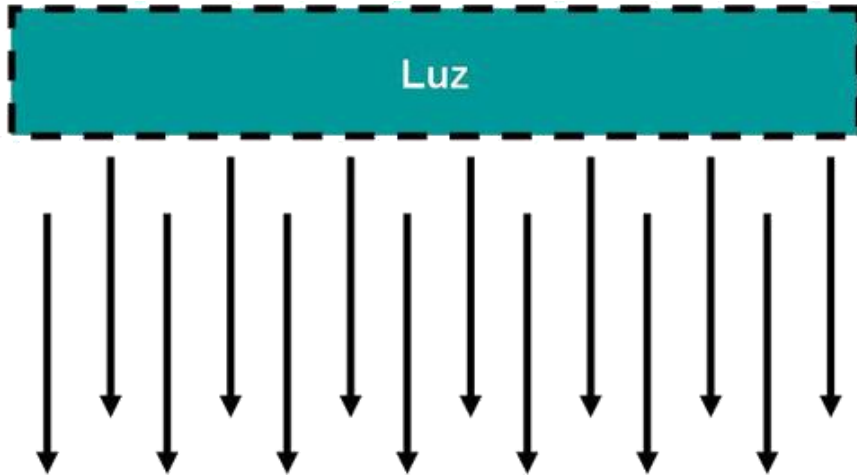
- **Luz direcional – Luz do sol**
  - **Está a uma distância infinita dos objetos, todos os raios são paralelos**
- **Luz pontual - Lâmpada**
  - **Possui uma posição na cena e emite luz para todas as direções**
- **Projektor (*Spot*) – *Holoforte em um campo***
  - **Possui uma posição na cena e emite luz na forma de um cone**

# Tipos de luzes

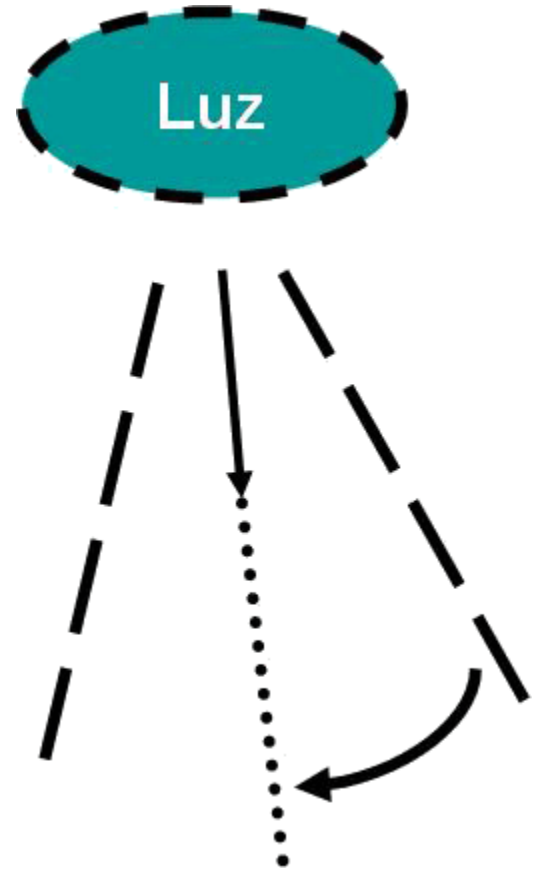
PONTUAL



DIRECCIONAL

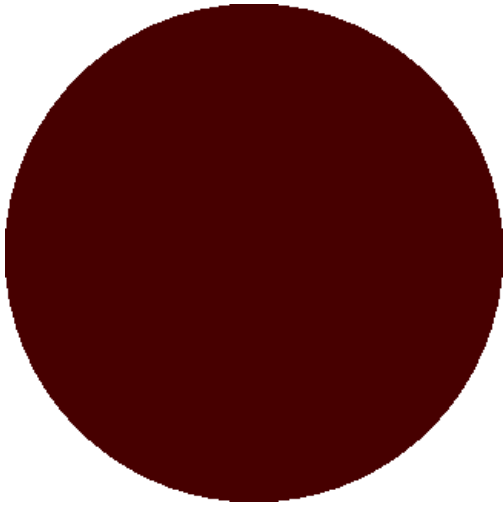


HOLOFOTE

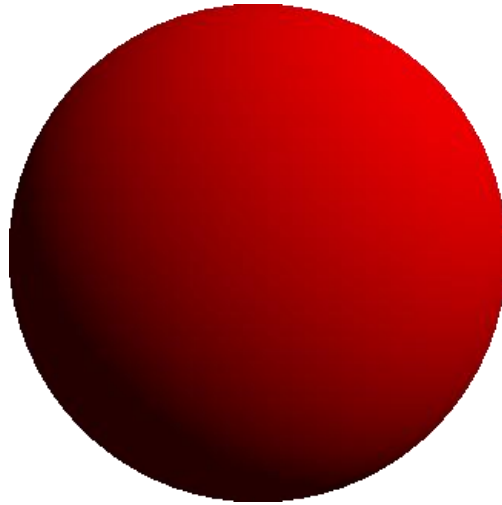


# Componentes da luz

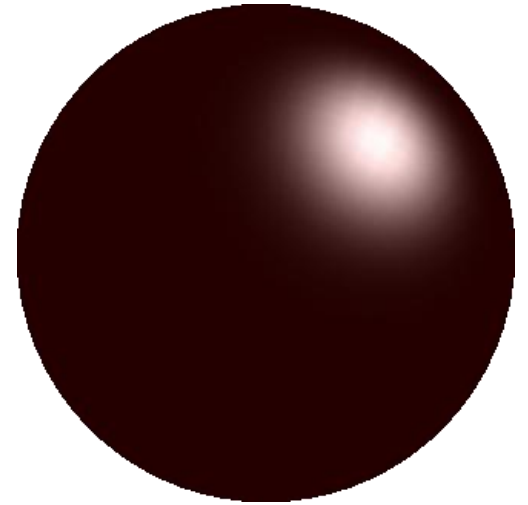
## Sombreamento de Phong



**COMPONENTE  
AMBIENTE**

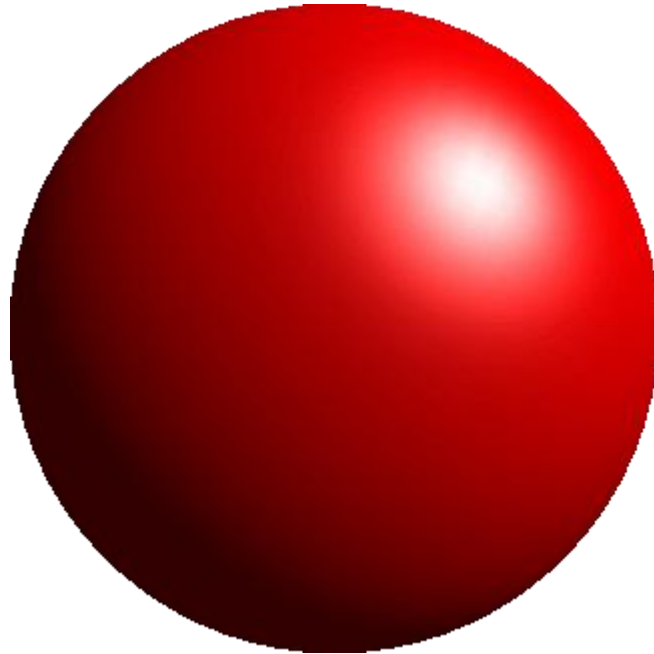


**COMPONENTE  
DIFUSA**



**COMPONENTE  
ESPECULAR**

# Cálculo da cor final



**IMAGEM FINAL**  
***ADIÇÃO DE TODAS AS COMPONENTES***

# PARTE PRÁTICA

- **Modificar o exemplo anterior para desenhar uma esfera utilizando o GLUT, e adicionar o cálculo da componente de luz especular para os objetos da cena (*LUZ3*)**
- **Modificar o número de triângulos da esfera do exemplo anterior e observar como a qualidade da luz é modificada (*LUZ4*)**

**OBS: Este exercício pode ser feito em casa**

# Atenuação

- Perda de transmissão da luz
- Geralmente devido a distância percorrida
- Tipos de atenuação
  - Constante, Linear e quadrática
- Por padrão, a atenuação constante está definida como 1
- Divide a intensidade da luz pela soma de todas as atenuações
- **Problemas com divisão por 0**
- Não se aplica a luzes direcionais

# Problemas com normais

- Quando as normais não são unitárias o cálculo de luz não é realizado corretamente
- Transformações de escala uniforme e não uniforme deformam a normal
- O OpenGL possui uma função para corrigir as normais das superfícies
- Perda de desempenho

# PARTE PRÁTICA

- **Tutorial do Nate Robins**
  - **Light position (Posição da luz)**
  - **Light material (Materiais)**

# Revisão

- OpenGL – Funções utilizadas
  - glNormal\*
  - glLight\*
  - glMaterial\*

# Exercícios

- **Desenhar um grande plano nos eixos X e Z, especificando suas normais. Adicionar duas luzes holoforte de cores diferente a cena. Configurar as luzes para iluminares partes diferentes do plano**

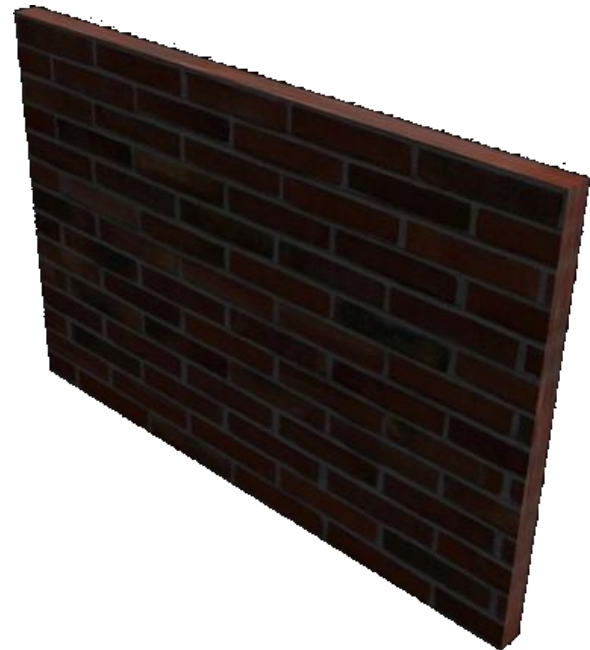
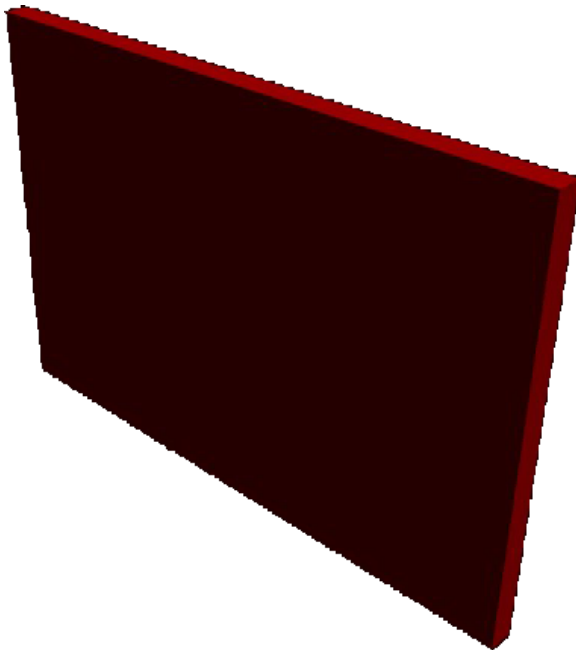


# **Aula 5**

## **Texturas**

# Texturas

- Uma imagem mapeada para a superfície de um objeto
- Adiciona maiores detalhes aos objetos sem modificar sua malha



# Texturas no OpenGL

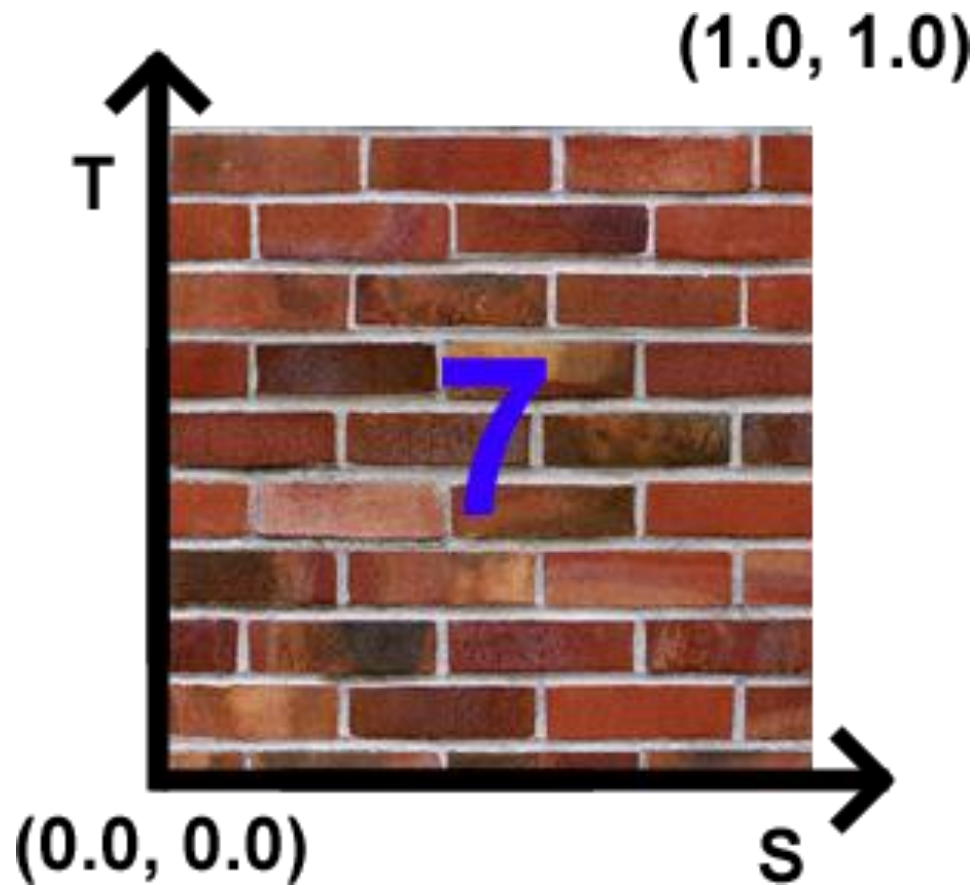
- Possui um objeto para armazenar a textura
  - Dados da imagem carregada (.bmp, .jpg)
  - Informações de como a textura será aplicada
- É preciso definir o mapeamento de textura para cada vértice do objeto
- Inicialmente o estado de mapeamento de texturas está desabilitado

# Texturas no OpenGL

- O OpenGL não possui funções para carregar os dados das imagens dos formatos existentes (bmp, jpg, png, tga)
- É necessário que as imagens tenham dimensões potência de 2
  - 16x64, 32x32, 256x256, 1024x1024

# Mapeamento de textura

- Define como a textura será mapeada para a superfície



# Passos para criação

- Criar uma janela no sistema operacional utilizado e um contexto para o OpenGL
- Configurar a área de desenho tridimensional com buffer profundidade
- **Habilitar mapeamento de texturas**
- **Criar um objeto de textura e configurá-lo**
- **Desenhar sólido, especificando as coordenadas de textura para cada vértice**
- Trocar os *buffers*

# PARTE PRÁTICA

- **Criar um plano e definir as coordenadas de texturas S,T para o mesmo. Criar manualmente os dados de uma texturas 8x8. Habilitar o mapeamento de texturas e aplicar a textura sobre o plano criado (*TEX1*)**

# DevIL

- **Biblioteca para carregar e salvar imagens**
  - **Bmp,tga, png, jpg, gif**
- **Multi-plataforma**
- **Integra-se com OpenGL**
  - **Transforma as dimensões da textura para funcionar no OpenGL (*exemplo, 640x480*)**
- **PROBLEMAS:**
  - **Documentação ruim, com poucos exemplos**
  - **Inverte as texturas no eixo Y**

# PARTE PRÁTICA

- **Criar um cubo e definir as coordenadas de texturas S,T para o mesmo. Carregar uma textura utilizando o DevIL. Habilitar o mapeamento de texturas e aplicar a textura sobre o cubo criado (*TEX2*)**

# Parâmetros da textura

- É possível especificar as coordenadas de textura fora do intervalo [0.0 ... 1.0]
- Quando isso ocorre o OpenGL pode repetir a textura, ou copiar a última linha ou coluna da textura
- É possível misturar o efeito para os eixos S e T das coordenadas de textura

# Parâmetros da textura

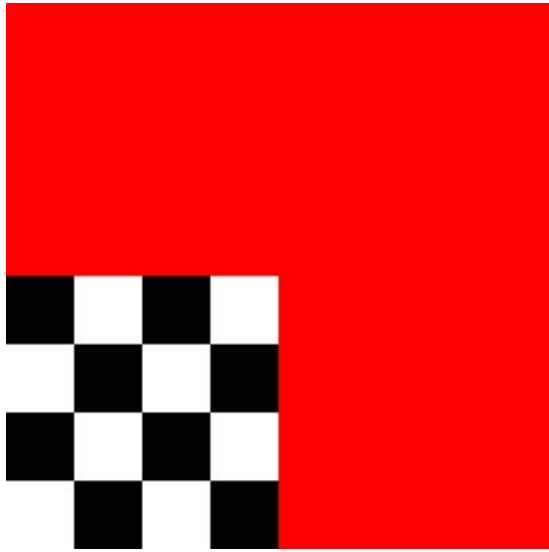
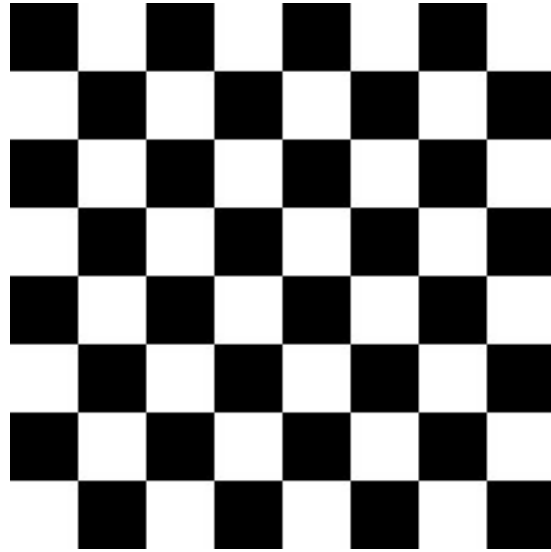
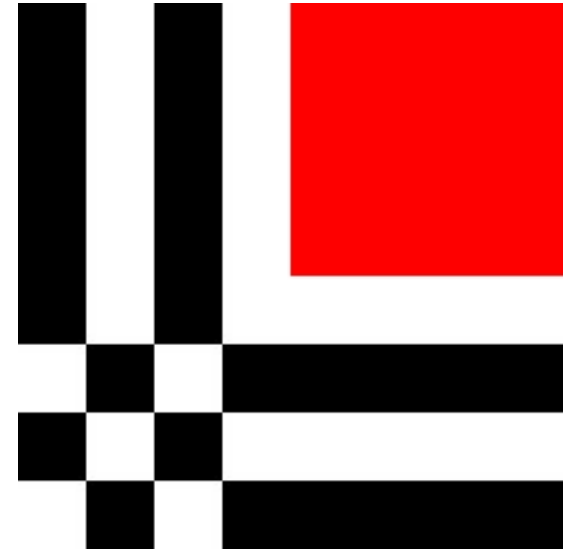


ILUSTRAÇÃO  
TEXTURA  
+  
POLÍGONO



MODO  
REPEAT

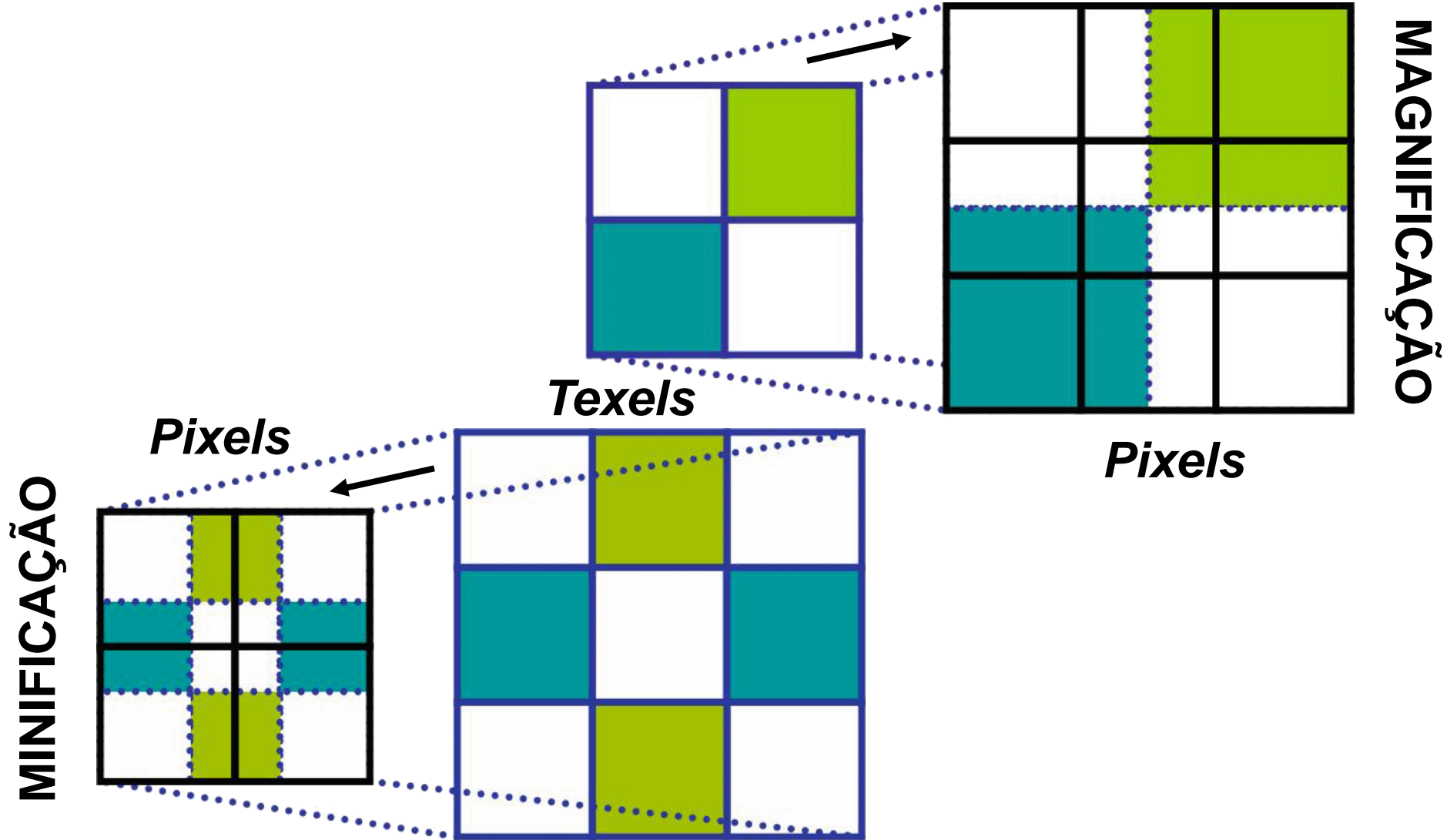


MODO  
CLAMP

# Filtros de texturas

- Existem duas situações onde é necessário que a textura seja filtrada
  - Cada *texel* da textura é mapeado para mais de um *pixel* da imagem: Magnificação
  - Cada *texel* da textura é mapeado para menos de um *pixel* da imagem: Minificação
- Nesses casos podem ser utilizados filtros para melhorar a qualidade da imagem
  - Nearest, Linear

# Filtros de textura



# PARTE PRÁTICA

- Tutorial do Nate Robins
  - Texture (Texturas)

# Tópicos avançados

- **Mip-mapping**
- **Geração automática de coordenadas de textura**
- **Matriz de textura**
- **Multi-texturização**

# Revisão

- **Informações dos vértices**
  - **Posição**
  - **Cor**
  - **Normal**
  - **Coordenada de textura**
  - **Outros**

# Revisão

- **DevIL – Funções utilizadas**
  - **ilInit()**
  - **iluInit()**
  - **ilutRenderer(ILUT\_OPENGL)**
  - **ilutGLLoadImage(...)**

# Revisão

- **OpenGL – Funções utilizadas**

- **TODO**

# Exercícios

- **Modificar o primeiro exemplo, adicionando filtros de minificação e magnificação. Tratar eventos do teclado para aumentar a distância da câmera ao plano**
- **Modificar o exemplo anterior, definindo o mapeando de textura superior ao intervalo [0.0 ... 1.0] e aplicando *clamp* para a coordenada S e *repeat* para a coordenada T**



# **Aula 6**

**Packman 3D**

**Fase 1**

# Introdução

- **Poucos desenvolvedores tem conhecimento de todas as etapas que envolvem a programação de um jogo**
- **A programação é apenas uma das partes do desenvolvimento de um jogo**
- **Objetivo do trabalho**
  - **Aprender as principais fases da programação e design de um jogo tridimensional**

# Cronograma

- **Aula 1**

- **Primeira versão do jogo – Carregar e desenhar o mapa do jogo**

- **Aula 2**

- **Adicionar outros objetos do mapa, jogador (packman), controles e sistema de colisão**

- **Aula 3**

- **Adicionar monstros (IA), “super-comida” e finalizar o jogo**

# Arquitetura utilizada

- **Orientação a objetos**
  - **Classes: Camera, LuzDirecional, Jogador, outros**
- **Bibliotecas auxiliares**
  - **OpenGL, GLU**
  - **GLUT**
  - **DevIL**
  - **Bibliotecas de matemática e colisão\***

*\* Bibliotecas simples criadas para o curso*

# Construção do jogo – Parte 1

- Criar um aplicativo utilizando GLUT
- Inserir biblioteca de matemática
- Criar classes para controle:
  - Câmera e Jogo
- Criar modelo de mapa do jogo
  - Quais objetos existem em um mapa de packman ?

# Construção do jogo – Parte 1

- Modelo de mapa proposto

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XS          XX          XX          SX
X  X  X      XXXX      X  X  X
X  X  X  XX  gggg  XX  X  X  X
X  X      XX  XXXX  XX      X  X
X      X      XXXX      X      X
XXX  XXXX      p  XXXX  XXX
X      X      XXXX      X      X
X  X      XX      XX      X  X
X  X  X  XX  XXXX  XX  X  X  X
X  X  X      XXXX      X  X  X
XS          XX          XX          SX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

# Construção do jogo – Parte 1

- **Criar classe para controle de:**
  - **Luz, material e estado do jogo**
  
- **Melhorias**
  - **Modificar o aspecto da câmera**
  - **Criar uma nova classe para desenho de cubo utilizando display list**

# **Aula 7**

**Packman 3D**

**Fase 2**

# Construção do jogo – Parte 2

- Inserir bibliotecas de colisão
- Criar classe do personagem
  - O que o personagem precisa ter ???

# **Aula 8**

**Packman 3D**

**Final**



# FUTUROS

- **Ensinar transparência ???**

# FUTUROS

- Na parte de transformação já estou ensinando escrita 2D, no desenvolvimento do jogo ensinar display list (Colocar os monstros em classes, talz)
- Já ensinado antes do jogo:
  - Desenho 3D, Cull Face, ZBuffer, Double Buffer, textura, luzes, projeções, etc
- Classes que disponibilizarei para os alunos:
  - Math(Vec3, Vec4), Colision(Sphere, AABB), DevIL, GLUT, Desenhos básicos(CUBO)