DEPARTAMENTO DE CIÊNCIAS
EXATAS E TECNOLOGIA

CURSO DE CIÊNCIA DA COMPUTAÇÃO

# TUTORIAL DE UTILIZAÇÃO DE OPENGL

**Marcionílio Barbosa Sobrinho**

## PROJETO ORIENTADO EM COMPUTAÇÃO II

ORIENTAÇÃO DA MONOGRAFIA:
PROF. MARCELO F. PORTO
COORDENAÇÃO DA DISCIPLINA:
PROF.ª MAGALI MARIA DE ARAÚJO BARROSO

**2003**
**BELO HORIZONTE**

**Marcionílio Barbosa Sobrinho**

# Tutorial de Utilização de OpenGL

Monografia apresentada como requisito parcial do Trabalho de Conclusão do Curso de Ciência da Computação do Departamento de Ciências Exatas e Tecnologia.

**Centro Universitário de Belo Horizonte**
**Belo Horizonte**
**2003**

Centro Universitário de Belo Horizonte

Departamento de Ciências Exatas e Tecnologia

Ficha de Aprovação

A monografia intitulada **"Tutorial de Utilização de OpenGL"** de autoria de **Marcionílio Barbosa Sobrinho**, aluno do Curso de Ciência da Computação - UNI-BH, apresentada em 11 de dezembro de 2003, recebeu aprovação da Banca Examinadora constituída pelos professores:

_____

Prof.ª Marcelo F. Porto, orientador

_____

Prof.ª Bráulio R. G. M. Couto

"*Não há nada melhor que alcançar uma meta. E não há nada pior que fazer só isso, não prosseguir."*

Amyr Klink

# Índice

# Índice de tabelas e figuras

# PARTE I - INTRODUÇÃO

## Introdução

O OpenGL aproveita-se dos recursos disponibilizados pelos diversos *hardwares* gráficos existentes. Todo desenvolvimento feito através desta API é independente inclusive de linguagem de programação, podendo um objeto gráfico ser desenvolvido em "OpenGL puro", ou seja, livre das bibliotecas particulares de cada linguagem de programação, devendo para tal somente seguir as regras semânticas e léxicas da linguagem escolhida, sem alteração das funções padronizadas do OpenGL.

Por se tratar de uma "biblioteca de funções", independente de plataforma e altamente utilizada tanto no meio acadêmico quanto no meio profissional, para desenvolvimento e aplicações em computação gráfica e devido à existência de poucos referenciais de pesquisa de OpenGL, disponíveis na língua portuguesa este tema foi escolhido para pesquisa e desenvolvimento.

O presente trabalho tem por objetivo o desenvolvimento de um tutorial apresentando as funcionalidades do OpenGL bem como a utilização de suas bibliotecas de desenvolvimento (GLUT: OpenGL Utility ToolKit) e funcionalidades da mesma, além da apresentação do embasamento teórico desta API. Visando a construção gradativa do conhecimento da ferramenta por parte do leitor, utilizando-se para tal, exemplos simples das funcionalidades do OpenGL.

Esta presente obra está dividida em 3 partes distintas : Introdução, Desenvolvimento e Conclusão.

Na primeira parte são apresentados os objetivos e as justificativas de escolha do tema, além de fazer um breve relato dos capítulos subseqüentes contidos no desenvolvimento da presente obra.

A parte de desenvolvimento foi subdividida em 11 capítulos onde são tratados além dos conceitos pré-liminares, conceitos mais avançados do OpenGL.

Existe uma breve introdução sobre a evolução da computação gráfica até a criação da API de desenvolvimento : OpenGL.

O capítulo 1 faz a conceituação do OpenGL além de apresentar suas principais características teóricas.

O capítulo 2 ensina como configurar os diversos ambientes de desenvolvimento para utilização das bibliotecas de desenvolvimento do OpenGL.

Na seqüência os demais capítulos irão tratar assuntos de computação gráfica com o auxílio do OpenGL e estão postados de forma a propiciar uma construção de conhecimento gradativa por parte do leitor. Os mesmos seguem a seguinte disposição de assuntos :

- Capítulo 3 - Criação de Primitivas, que descreve como construir as primitivas básicas com a utilização do OpenGL : pontos, polígonos, triângulos etc.;

- Capítulo 4 - Cores : Além de apresentar como a cor é percebida pelo olho humano, também apresenta a formação de cores através do computador e como estas cores são tratadas pelo OpenGL.

- Capítulo 5 - Transformações : Apresenta as transformações necessárias para construção de uma determinada cena, abordando : rotação, translação, escala; além das transformações de perspectiva e transformações ortográficas.

- Capítulo 6 - Formas 3D : Descreve as formas 3D suportadas e definidas no OpenGL.

- Capítulo 7 - Modelos de Iluminação : Apresenta os modelos de iluminação suportados, além de apresentar as propriedades dos materiais no OpenGL.

- Capítulo 8 - Modelagem hierárquica : Descreve as formas de modelagem hierárquicas suportadas pelo OpenGL.

- Capítulo 9 - Texturas : Descreve as várias formas de aplicação de texturas em um determinado objeto, além apresentar uma biblioteca auxiliar para mapeamento de figuras no formato JPEG.

- Capítulo 10 - Sombra Planar : Descreve os passos para criação de sombras planares de objetos, baseados em uma fonte de luz e um plano de referência.

- Capítulo 11 – Blending : Descreve a utilização do efeito “Blending” para criação de efeitos de transparência em objetos.

A terceira parte da desta obra apresenta a conclusão do trabalho.

# PARTE II - DESENVOLVIMENTO

## HISTÓRICO

A Computação Gráfica está presente em todas as áreas, desde os mais inconseqüentes joguinhos eletrônicos até o projeto dos mais modernos equipamentos para viagens espaciais, passando também pela publicidade, com as mais incríveis vinhetas eletrônicas e pela medicina onde a criação de imagens de órgãos internos do corpo humano possibilitam o diagnóstico de males que em outros tempos somente seria possível com intervenções cirúrgicas complicadas e comprometedoras.

Parece existir consenso entre os pesquisadores da história da Computação Gráfica de que o primeiro computador a possuir recursos gráficos de visualização de dados numéricos foi o **"Whirlwind I" (furacão)**, desenvolvido pelo MIT. Este equipamento foi desenvolvido, em 1950, com finalidades acadêmicas e também possivelmente militares, pois logo em seguida o comando de defesa aérea dos EUA desenvolveu um sistema de monitoramento e controle de vôos (SAGE - Semi-Automatic Ground Enviroment) que convertia as informações capturadas pelo radar em imagem em um tubo de raios catódicos (na época uma invenção recente) no qual o usuário podia apontar com uma caneta ótica. Ocorre que nesta época os computadores eram orientados para fazer cálculos pesados para físicos e projetistas de mísseis não sendo próprios para o desenvolvimento da Computação Gráfica.

Em 1962, surgiu uma das mais importantes publicações de Computação Gráfica de todos os tempos, a tese do Dr. Ivan Sutherland **("Sketchpad - A Man-Machine Graphical Communication System")**, propunha uma forma de intenção muito semelhante ao que hoje chamados de interfaces **WIMP – Window-Icon-Menu-Pointer**.

Esta publicação chamou a atenção das indústrias automobilísticas e aeroespaciais americanas. Os conceitos de estruturação de dados bem como o núcleo da noção de Computação Gráfica interativa levou a General Motors a desenvolver o precursor dos primeiros programas de C.A.D. Logo em seguida

diversas outras grandes corporações americanas seguiram este exemplo sendo que no final da década de 60 praticamente todas as indústrias automobilísticas e aeroespaciais faziam uso de *softwares* de CAD

Dois fatores, entretanto, foram fundamentais para o desenvolvimento da Computação Gráfica tal como a conhecemos hoje:

a)O desenvolvimento da tecnologia de circuitos integrados durante a década de 70 que permitiu o barateamento e a conseqüente popularização das máquinas;

b)O fim da idéia de que os fabricantes de computadores devem fornecer apenas a máquina e o sistema operacional e que os usuários devem escrever seus próprios aplicativos. A popularização dos aplicativos prontos e integrados (planilhas, editores de texto, editores gráficos, processadores de imagem, bancos de dados, etc) permitiram a popularização da Computação Gráfica na medida em que possibilitaram que o usuário comum sem conhecimento ou tempo para desenvolver aplicativos gráficos (nem sempre tão simples de serem programados) pudesse utilizar as facilidades da mesma.

Com a evolução da computação gráfica fez-se necessária a existência de bibliotecas computacionais que suportassem o desenvolvimento dos aplicativos gráficos tanto os criados pelas indústrias de softwares como os desenvolvidos pelo próprio usuário.

Foi nos anos 70 que ocorreram os primeiros pacotes gráficos e as primeiras conferências do SIGGRAPH (Special Interest Group on Graphics). Foi proposta por um comitê essencialmente Norte Americano do ACM SIGRAPH, em 1977, a primeira padronização gráfica como o “Core Graphics System”, conhecido como CORE. O objetivo era propor, para as aplicações em 2D e 3D, um padrão contendo um conjunto de funções gráficas que, na sua utilização não dependessem dos equipamentos gráficos envolvidos na aplicação. No ponto final da década de 70, foi formulado um outro padrão de gráficos, chamado GKS (Graphic Kernel System), que deveria adaptar-se melhor à grande diversidade dos

equipamentos gráficos e das aplicações potenciais, através da introdução da noção de estação de trabalho. O GKS foi adotado pela ISO (International Standards Organization) como norma internacional em 1985. A introdução da GKS como padrão internacional representou um avanço, apesar de, na sua versão atual ele não possibilitasse o aproveitamento de certos recursos disponíveis em novos equipamentos gráficos. A proposta em estudo pela ISO de um padrão GKS para aplicações gráficas em 3D, deveria contribuir para expandir a sua aceitação. Ao nível de padronização gráfica, uma proposta foi feita pela ANSI (American National Standard Institute). Tratava-se do PHIGS (Programmer's Hierarchical Interctive), que cobria os aspectos de modelagem de objetos por hierarquias (pontos não abordados pelo GKS), assim como os aspectos gráficos em 3D. A partir dos anos
80, com a chegada dos micros e dos seus aperfeiçoamentos constantes, as aplicações da Computação Gráfica deixaram de ser reservadas aos especialistas.

Padrões gráficos, como o GKS *(Graphical Kernel System*) e o PHIGS *(Programmer's Hierarchical Interactive Graphics System*), tiveram importante papel na década de 80, inclusive ajudando a estabelecer o conceito de uso de padrões mesmo fora da área gráfica, tendo sido implementados em diversas plataformas. Nenhuma destas API's (Application Programming Interface), no entanto, conseguiu ter grande aceitação (Segal, 1994). A interface destinada a aplicações gráficas 2D ou 3D deve satisfazer diversos critérios como, por exemplo, ser implementável em plataformas com capacidades distintas sem comprometer o desempenho gráfico do hardware e sem sacrificar o controle sobre as operações de hardware. (Segal, 1997)

Atualmente, a OpenGL ("GL" significa *Graphics Librar*y) é uma API e grande utilização no desenvolvimento e aplicações em computação gráfica. Este padrão é o sucessor da biblioteca gráfica conhecida como IRIS GL, desenvolvida pela *Silicon Graphics* como uma interface gráfica independente de hardware (Kilgard, 1994). A maioria das funcionalidades da IRIS GL foi removida ou reescrita na OpenGL e as rotinas e os símbolos foram renomeados para evitar

conflitos (todos os nomes começam com gl ou GL_). Na mesma época, foi formado o *OpenGL Architecture Review Boar*d, um consórcio independente que administra o uso da OpenGL, formado por diversas empresas da área. OpenGL é uma interface que disponibiliza um controle simples e direto sobre um conjunto de rotinas, permitindo ao programador especificar os objetos e as operações necessárias para a produção de imagens gráficas de alta qualidade. Para tanto, a OpenGL funciona como uma máquina de estados, onde o controle de vários atributos é realizado através de um conjunto de variáveis de estado que inicialmente possuem valores *defaul*t, podendo ser alterados caso seja necessário. Por exemplo, todo objeto será traçado com a mesma cor até que seja definido um novo valor para esta variável. Por ser um padrão destinado somente à renderização (Segal, 1997), a OpenGL pode ser utilizada em qualquer sistema de janelas (por exemplo, X Window System ou MS Windows), aproveitando-se dos recursos disponibilizados pelos diversos hardwares gráficos existentes.

# Capítulo 1 - O que é OpenGL

OpenGL é uma interface de *software* para dispositivos de *hardware*. Esta interface consiste em cerca de 150 comandos distintos usados para especificar os objetos e operações necessárias para produzir aplicativos tridimensionais interativos.

OpenGL foi desenvolvido com funcionalidades independentes de interface de *hardware* para ser implementado em múltiplas plataformas de *hardware*.

Diante das funcionalidades providas pelo OpenGL, tal biblioteca tem se tornado um padrão amplamente adotado na indústria de desenvolvimento de aplicações. Este fato tem sido encorajado também pela facilidade de aprendizado, pela estabilidade das rotinas, pela boa documentação disponível [Neider2000] e pelos resultados visuais consistentes para qualquer sistema de exibição concordante com este padrão.

As especificações do OpenGL não descrevem as interações entre OpenGL e o sistema de janelas utilizado (Windows, X Window etc). Assim, tarefas comuns em uma aplicação, tais como criar janelas gráficas, gerenciar eventos provenientes de *mouse* e teclados, e apresentação de menus ficam a cargo de bibliotecas próprias de cada sistema operacional. Neste trabalho será utilizada a biblioteca GLUT (OpenGL ToolKit) para gerenciamento de janelas.

Desde sua introdução em 1992, OpenGL transformou-se num padrão extensamente utilizado pelas indústrias. OpenGL promove a inovação e acelera o desenvolvimento de aplicações incorporando um grande conjunto de funções de *render*, de texturas, de efeitos especiais, e de outras poderosas funções de visualização.

## 1.1 - OpenGL como máquina de estados

OpenGL é uma máquina de estados. Todos os estados ou modos habilitados nas aplicações têm efeito enquanto os mesmos estiverem ligados ou forem modificados. Todas as características do OpenGL, configuráveis através de variáveis tais como : cores, posições, característica de luzes, propriedades de materiais, objetos que estão sendo desenhados, projeções e transformações.

## 1.2 - O "Pipeline" do OpenGL

A maior parte das implementações do OpenGL tem uma ordem de operações a serem executadas. Uma série de estágios de processos chamam o "*pipeline*" de renderização do OpenGL.

O diagrama seguinte mostra como o OpenGL, obtém e processa os dados.

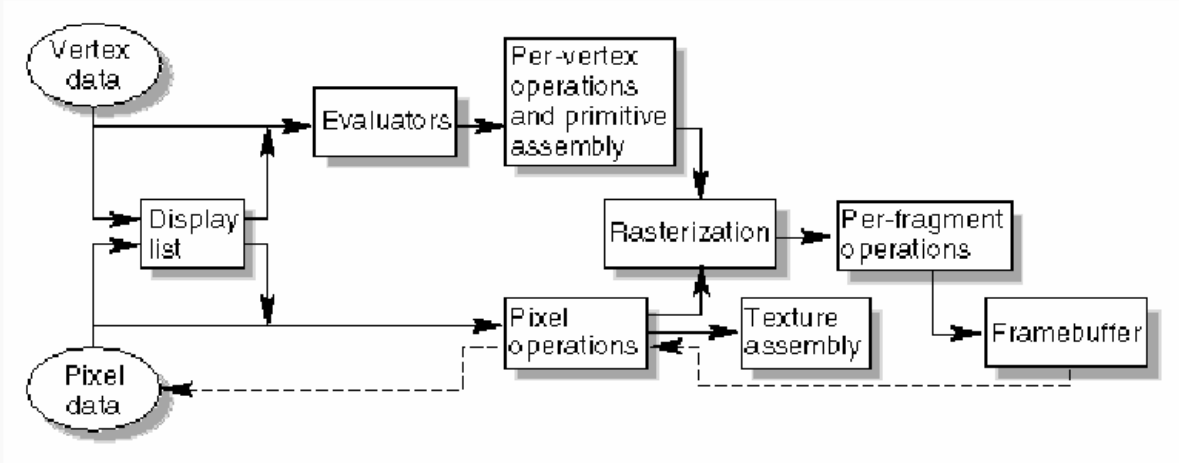



Figura 1. Processamento de dados pelo OpenGL

### 1.2.1 - Listas de Exposição

Todos os dados descrevem uma geometria ou pixels, podem ser conservados em uma lista da exposição para uso corrente ou serem usados mais tarde. (A alternativa além de manter dados em uma lista de exposição é processar os dados imediatamente (também conhecido como modo imediato))

Quando uma lista da exposição é executada, os dados retidos são enviados da lista apenas como se fossem enviados pela aplicação no modo imediato.

### 1.2.2 - Avaliadores

Todas as primitivas geométricas são eventualmente descritas por vértices. As curvas e as superfícies paramétricas podem inicialmente ser descritas pelos pontos de controle e pelas funções polinomiais chamadas funções base. Os avaliadores fornecem um método para derivar os vértices usados para representar a superfície dos pontos de controle. O método é o mapeamento polinomial, que pode produzir a normal de superfície, as coordenadas da textura, as cores, e valores de coordenadas espaciais dos pontos de controle.

### 1.2.3 - Operações por vértices

Para dados dos vértices, está seguida do "o estágio das operações por vértices", que converte os vértices em primitivas. Alguns dados do vértice (por exemplo, coordenadas espaciais) são transformados em matrizes 4 x 4 de pontos flutuantes. As coordenadas espaciais são projetadas de uma posição no mundo 3D a uma posição na tela. Se as características avançadas estiverem habilitadas permitidas, este estágio é mesmo mais ocupado. Se texturização for usada, as coordenadas da textura podem ser geradas e transformadas aqui. Se as luzes forem habilitadas, os cálculos da luz serão executados usando os vértices transformados, a normal da superfície, a posição da fonte de luz, as propriedades de materiais, e a outras informações de luzes para produzir um valor da cor.

### 1.2.4 - Montagem de Primitivas

*Clipping*, uma parte importante da montagem de primitivas, é a eliminação de partes da geometria que saem fora de uma parte do espaço, definido por um plano. O clipping do ponto simplesmente passa ou rejeita vértices; o clipping da linha ou do polígono pode adicionar novos vértices , dependendo de como a linha ou o polígono são interligados. Em alguns casos, isto é seguido pela divisão da perspectiva, a qual faz com que os objetos geométricos distantes pareçam mais

perto. Então as operações do viewport e da profundidade (coordenada de z) são aplicadas.

### 1.2.5 - Operações de Pixels

Enquanto os dados geométricos pegam um caminho através do pipeline de renderização do OpenGL., dados de pixels tomam uma rota diferente. Os dados de Pixels em uma matriz na memória do sistema são empacotados e em seguida escalados, inclinados e processados por um mapa de pixels. Os resultados são escritos na memória da textura ou emitidos à uma etapa de rasterização. Se os dados do pixel forem lidos do framebuffer, operações de transferência de pixels (escala, polarização, mapeamento, "*clamping*") são executadas. Então estes resultados são empacotados em um formato apropriado e retornados a uma matriz de memória do sistema.

### 1.2.6 - Montagem de Texturas

Uma aplicação OpenGL pode aplicar imagens de texturas em objetos geométricos, para tornar estes mais realísticos. Se diversas imagens de textura forem usadas, as mesmas deverão ser colocadas em objetos de modo que se possa facilmente comutar entre elas. Algumas exceções do OpenGL podem ter recursos especiais para acelerar o desempenho das texturas. Se existir memória especial disponível , os objetos de textura podem ter prioridade de controle do recurso limitado de memória.

### 1.2.7 - Rasterização

Rasterização é a conversão de dados geométricos e do pixel em fragmentos. Cada quadrado do fragmento corresponde a um *pixel* no *framebuffer*. Os "*stipples*" da linha e do polígono, largura da linha, tamanho do ponto, modelo de sombra, e os cálculos da cobertura para suportar o *antialiasing* são feitos considerando a conexão dos vértices em linhas ou os *pixels* internos. Os valores da cor e da profundidade são atribuídos para cada quadrado do fragmento.

### 1.2.8 - Operações fragmentadas

Antes que os valores estejam armazenados realmente no framebuffer, uma série de operações, que podem se alterar ou mesmo jogar sair fora dos fragmentos, são executadas. Todas estas operações podem ser habilitadas ou desabilitadas. A primeira operação que pode ser encontrada é o "*texturing*", onde um *texel* (elemento da textura) é gerado na memória da textura para cada fragmento e aplicado ao fragmento. Então os cálculos do "*fog*" podem ser aplicados seguidos pelo teste "*scissor*", pelo teste do alfa, pelo teste do estêncil, e pelo teste do *buffer* de profundidade (o *buffer* de profundidade é para a remoção de faces ocultas da superfície). Então, "*blending*", operação lógica de "*dithering*", e mascaramento por um *bitmask* podem ser executadas.

## 1.3 - Funções gráficas do OPENGL

***Buffer de acumulação*** : Trata-se de um *buffer* no qual múltiplos *frames* renderizados, podem ser compostos para produzir uma única imagem. Usado para efeitos tais como a profundidade de campo, "*blur*" de movimento, e de *anti-aliasing* da cena.

***Alfa Blending :*** Provê mecanismos para criar objetos transparentes. Usando a informação alfa, um objeto pode ser definido como algo totalmente transparente até algo totalmente opaco.

***Anti-aliasing :*** Um método de renderização utilizado para suavizar linhas e curvas. Esta técnica calcula a média da cor dos *pixels* junto à linha. Tem o efeito visual de suavizar a transição dos pixels na linha e daqueles junto à linha, assim fornecendo uma aparência mais suave.

***Modo "Color-Index":*** *Buffer* de Cores que armazena índices de cores das componentes vermelhas, verdes, azuis, e alfa das cores (RGBA).

***Display Lists :*** Uma lista nomeada de comandos de OpenGL. Os índices de um *Display list* podem ser pré-processados e podem conseqüentemente executar mais eficientemente do que o mesmo conjunto de comandos do OpenGL executados no modo imediato.

***Double buffering :*** Usado para fornecer uma animação suave dos objetos. Cada cena sucessiva de um objeto em movimento pode ser construída em "*background*" ou no buffer "invisível" e então apresentado. Isto permite que somente as imagens completas sejam sempre apresentadas na tela.

***FeedBack :*** Um modo onde OpenGL retornará a informação geométrica processada (cores, posições do *pixel*, e assim por diante) à aplicação.

***Gouraud Shading :*** Interpolação suave das cores através de um segmento de polígono ou de linha. As cores são atribuídas em vértices e linearmente interpoladas através da primitiva para produzir uma variação relativamente suave na cor.

***Modo Imediato :*** A execução de comandos OpenGL quando eles são chamados, possui resultado melhor do que os "*Display Lists*".

***Iluminação e sombreamento de materiais :*** A habilidade de computar exatamente a cor de algum ponto dado as propriedades materiais para a superfície.

***Operações de pixel :*** Armazena, transforma, traça e processa aumento e redução de imagens.

***Executores polinomiais :*** Para suportar as NURBS (*non-uniform rational B-splines*).

***Primitivas :*** Um ponto, uma linha, um polígono, um *bitmap*, ou uma imagem. Primitivas da rasterização : *bitmaps* e retângulos de pixels

***Modo RGBA :*** *Buffers* de cores armazenam componentes vermelhos, verdes, azuis, e alfa da cor.

***Seleção e colheita :*** Trata-se de um modo no qual o OpenGL determina se certa primitiva identificada do gráficos é renderizada em uma região no *buffer* de *frame*.

***Planos do estêncil :*** Um *buffer* que é usado para mascarar *pixels* individuais no *buffer* de *frame* de cores.

***Mapeamento de Texturas :*** O processo de aplicar uma imagem a uma primitiva gráfica. Esta técnica é usada para gerar o realismo nas imagens.

***Transformações :*** A habilidade de mudar a rotação, o tamanho, e a perspectiva de um objeto no espaço 3D coordenado.

***Z-buffering :*** O *Z-buffer* é usado para verificar se uma porção de um objeto é mais próxima da tela do que outra. É importante na remoção de superfície escondida.

# Capítulo 2 - Ambiente OpenGL para desenvolvimento

## 2.1 - Instalação do OPENGL

As bibliotecas do OpenGL são distribuídas como parte dos sistemas operacionais da Microsoft, porém as mesmas podem ser baixadas no site oficial do OpenGL : http://www.opengl.org. Estas bibliotecas também estão disponíveis em outros sistemas operacionais por padrão, tais como, MacOS e Unix Solaris, além de estarem disponíveis no Linux.

## 2.2 - Instalação do GLUT

O GLUT é um conjunto de ferramentas para escrita de programas OpenGL, independente do sistema de janelas. Ele implementa um sistema de janelas simples através de sua API, para os programas OpenGL. GLUT provê uma API portável, o que permite que programas trabalhem tanto em ambientes baseados em WIN32 quanto X11.

O GLUT suporta :

- Janelas múltiplas para renderização OpenGl
- Resposta a eventos baseados em Callback de funções
- Uma rotina "*idle*" e "*timers*"
- Criação de menus *pop-up* simples
- Suporte pra *bitmaps* e fontes
- Uma miscelânea de funções para gerenciamento de janelas.

### 2.2.1 - Instalando o GLUT no Borland C++ Builder 5 no ambiente Windows

Os arquivos de instalação do GLUT poderão ser obtidos em http://www.opengl.org/developers/documentation/glut/index.html

1 – Baixe o arquivo do *link* acima e descompacte-os em uma pasta temporária

2 – Copie os arquivos glut.dll e glut32.dll a pasta c:\windows\system32 ou uma pasta que esteja no caminho do sistema operacional, no PATH.

3 – Copie os arquivos glut*.lib para <diretório de Borland C Builder>\lib

4 – Copie o arquivo glut.h para < diretório de Borland C Builder >\include\gl

5 – Como os arquivos originais são desenvolvidos para Microsoft Visual C++ você deverá executar os seguintes comandos dentro da pasta \lib, localizada dentro do diretório de instalação do Borland C++ :

- implib glut.lib c:\windows\system32
- implib glut32.lib c:\windows\system\glut32.dll

Obs.: Normalmente poderão ocorrer problemas no momento da compilação / linkedição de um programa OpenGL, com o Borland C++, porém isto se deve ao fato da instalação do mesmo ter sido feita no caminho \Arquivos de programas\Borland\CBuilder5, existe um pequeno bug. Para resolução deste problema instale o CBuilder e um diretório tipo C:\BC5 que funcionará corretamente.

### 2.2.2 - Instalando o GLUT no MS Visual C++

Como esta API foi desenvolvida no próprio MS Visual C++, não é necessária nenhuma conversão da mesma para funcionamento. Porém é necessário a instalação da mesma. Assim ela deverá ser baixada do site http://www.opengl.org/developers/documentation/glut/index .

1 – Descompacte os arquivos em um diretório temporário.

2 – Copie os arquivos glut.dll e glut32.dll a pasta c:\windows\system32 ou uma pasta que esteja no caminho do sistema operacional, no PATH.

3 – Copie os arquivos glut*.lib para <diretório de instalação do visual c>\lib

4 – Copie o arquivo glut.h para <diretório de instalação do visual c>\include\gl

5 – Quando for compilar um programa deverão serem incluídas as seguintes bibliotecas na opção de linkedição do programa : opengl32.lib, glu32.lib, glut32.lib.

### 2.2.3 - Instalando o GLUT no DEV-C++

O Dev-C++ é um ambiente de desenvolvimento integrado para a linguagem de programação C/C++. O compilador Dev-C++, que dá suporte a compiladores baseados no GCC (GNU Compiler Collection), pode ser obtido em http://www.bloodshed.net/devcpp.html.

A versão mais atual do Dev-C++ (4.9.8 ou superior) já inclui as bibliotecas do GLUT por padrão devendo apenas ter o passo 2 executado. Para versões anteriores siga os seguintes passos :

1. Faça o download do arquivo glut-devc.zip e descompacte o mesmo;
2. Mova o arquivo glut.h para a pasta GL do DevC++ (C:\Dev-C++\Include\GL);
3. Mova os arquivos glut32.def e libglut.a para a pasta Lib do DevC++ (C:\Dev-C++\Lib);
4. Mova o arquivo glut32.dll para a mesma pasta onde se encontram os arquivos opengl32.dll e glu32.dll;

## 2.3 - Sintaxe de Comandos do OpenGL

Os comandos da API OpenGL obedecem a um padrão bem definido para especificação dos nomes de funções e constantes. Todos os comandos utilizam-se do prefixo gl em letras minúsculas. Similarmente, OpenGL define constantes com as iniciais GL_, em letras maiúsculas, e usa um "*underscore*" para separar as palavras ( Ex. GL_COLOR_BUFFER_BIT ).

Em algumas funções algumas seqüências de caracteres extras aparecem, como sufixo, no nome destas funções (Ex. glColor3f e glVertex3f). É verdade que a parte "Color" do comando glColor3f() não é suficiente para definir o comando como um conjunto de cores correntes. Particularmente, a parte "3" do sufixo

indica que três argumentos são necessários; outra versão de “Color” necessita de 4 argumentos. O “f” do sufixo indica que os argumentos são do tipo números de ponto flutuante. Através destas definições para diferentes tipos de formatos é permitido ao OpenGL aceitar dados no seu próprio formato definido.

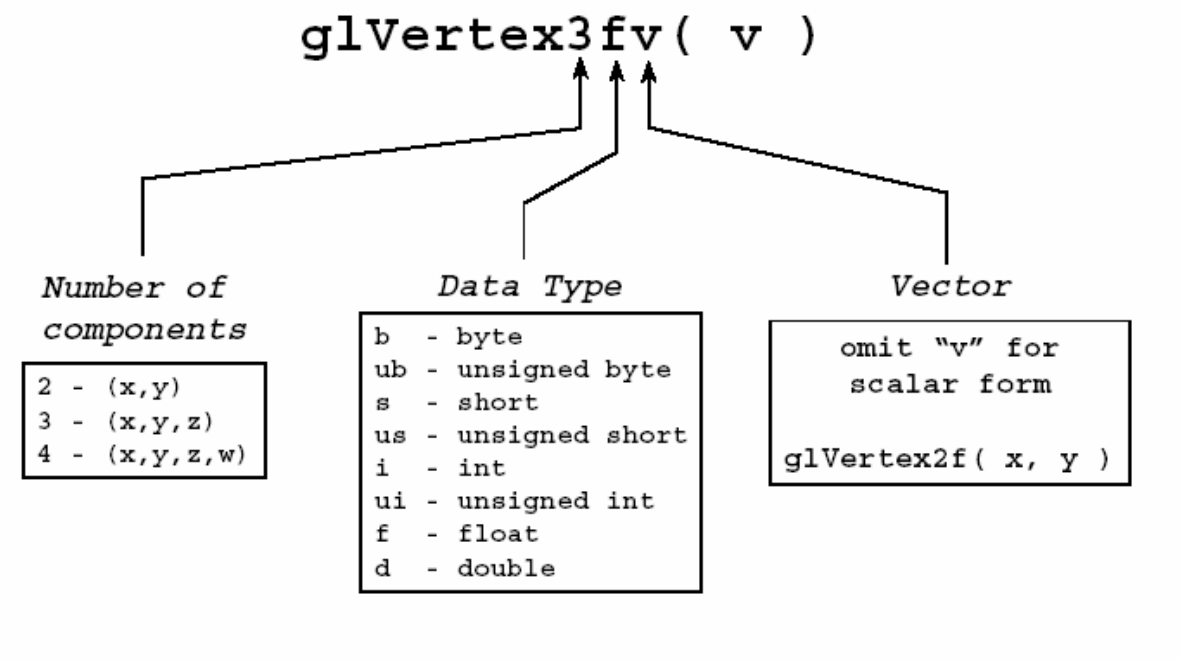



Figura 2. Sintaxe de comandos OpenGL

Alguns comandos OpenGL aceitam até 8 tipos diferentes de argumentos. As letras usadas como sufixo, como dito anteriormente, irão determinar o tipo de dados correspondente à padronização ISO para implementação em linguagem C.

Sufixos de comandos e tipos de dados para argumentos

| **Sufixo** | **Tipo de Dados** | **Tipo correspondente na linguagem C** | **Definição de tipo para OpenGL** |
|---|---|---|---|
| b | Inteiro de 8 bits | Signed char | GLbyte |
| s | Inteiro de 16 bits | Short | GLshort |
| i | Inteiro de 32 bits | Int ou long | GLint,GLsizei |
| f | Ponto flutuante de 32 bits | Float | GLfloat,GLclampf |

| | | | |
|---|---|---|---|
| d | Ponto flutuante de 64 bits | double | GLdouble,GLclampd |
| ub | Inteiro não sinalizado de 8 bits | Unsigned char | GLubyte,GLboolean |
| us | Inteiro não sinalizado de 16 bits | Unsigned short | GLushort |
| ui | Inteiro não sinalizado de 32 bits | Unsigned int ou unsigned long | GLuint,GLenum,GLbitfield |

Tabela 1. Sufixo de comandos OpenGL

## 2.4 - Estrutura Básica de Programas OpenGL

Um programa OpenGL deve conter um mínimo de requisitos para sua perfeita execução. Normalmente alguns passos devem ser seguidos para criação destes programas. Estes passos são :

- declaração dos arquivos de header para o OpenGL;
- configurar e abrir a janela;
- inicializar os estados no OpenGL;
- registrar as funções de "*callback*";
- renderização;
- redimensionamento;
- entradas : teclado, mouse, etc.;
- entrar no loop de processamento de eventos.

O programa exemplo abaixo irá ilustrar estes passos

```
#include <GL/gl.h>
#include <GL/glut.h>
void main( int argc, char** argv )
{
  int mode = GLUT_DOUBLE | GLUT_RGB;
  glutInitDisplayMode( mode );
  glutInitWindowSize(400,350);
  glutInitWindowPosition(10,10);
  glutCreateWindow( argv[0] );
  init();
  glutDisplayFunc( display );
  glutReshapeFunc( resize );
  glutKeyboardFunc( key );
```

```
  glutIdleFunc( idle );
  glutMainLoop();
}
```

- Os arquivos de *header* contém as rotinas e declarações necessárias para utilização do OpenGL/GLUT com a linguagem c/c++.

- As funções glutInitDisplayMode() e glutCreateWindow() compõem o passo de configuração da janela. O modo da janela que é argumento para a função glutInitDisplayMode(),indica a criação de uma janela double-buffered (GLUT_DOUBLE) com o modo de cores RGBA (GLUT_RGB). O primeiro significa que os comandos de desenho são executados para criar uma cena fora da tela para depois, rapidamente, colocá-la na view (ou janela de visualização). Este método é geralmente utilizado para produzir efeitos de animação. O modo de cores RGBA significa que as cores são especificadas através do fornecimento de intensidades dos componentes *red*, *green* e *blue* separadas. A função glutCreateWindow() cria a janela com base nos parâmetros definidos nas funções glutInitWindowSize (tamanho da janela em pixels) e glutInitWindowPosition (coordenadas para criação da janela). Esta janela conterá o nome especificado em seu parâmetro de entrada.

- Em seguida é chamada a rotina init(), a qual contém a primeira inicialização do programa. Neste momento serão inicializados quaisquer estados OpenGL, que serão executados na execução do programa.

- O próximo passo será o registro das funções de *callback*, que estarão sendo utilizadas na execução do programa.

- Finalmente o programa irá entrar em um processo de *loop*, o qual interpreta os eventos e chamadas das rotinas especificadas como *callback*.

### 2.4.1 - Rotinas de *Callback*

As funções de *callback* são aquelas executadas quando qualquer evento ocorre no sistema, tais como : redimensionamento de janela o desenho da mesma, entradas de usuários através de teclado, mouse, ou outro dispositivo de entrada, e ocorrência de animações. Assim o desenvolvedor pode associar uma ação especifica à ocorrência de determinado evento.

GLUT oferece suporte a muitos diferentes tipos de ações de *callback* incluído :

- glutDisplayFunc() – chamada quando um pixel na janela necessita ser atualizado.

- glutReshapeFunc() – chamado quando a janela é redimensionada.

- glutKeyboardFunc() – chamada quando uma tecla do teclado é pressionada.

- glutMouseFunc() – chamada quando o usuário pressiona um botão do mouse.

- glutMotionFunc() - chamada quando o usuário movimenta o mouse enquanto mantém um botão do mesmo pressionado.

- glutPassiveMouseFunc() – chamado quando o *mouse* é movimentado, independente do estado dos botões.

- glutIdleFunc() – uma função de *callback* chamada quando nada está acontecendo. Muito útil para animações.

## 2.4.2 - Exemplo de um programa OpenGL

Este exemplo simples, apenas desenha um quadrado na tela.

```
/* Exemplo1.c - Marcionílio Barbosa Sobrinho
* Programa simples que apresenta o desenho de um quadrado
* Objetivo : Demonstrar funções de gerenciamento de
*            janelas e funções de callback
* Referência do Código: OpenGL Programming Guide - RedBook
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

void display(void)
{
   /* Limpa o Buffer de Pixels */
   glClear (GL_COLOR_BUFFER_BIT);

   // Define a cor padrão como branco
   glColor3f (1.0, 1.0, 1.0);

   /* desenha um simples retângulo com as coordenadas
   * (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();

   /* Inicia o processo de desenho através dos
      dados bufferizados
   */
   glFlush ();
}

void init (void)
{
    /* Seleciona a cor de fundo para limpeza da tela */
    glClearColor (0.0, 0.0, 0.0, 0.0);

    /* inicializa os valores de visualização */
    glMatrixMode(GL_PROJECTION);

    /* Faz com que a matriz corrente seja inicializada com a matriz identidade
       (nenhuma transformação é acumulada)
    */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}


/*
  Cria a janela
*/
int main(int argc, char** argv)
{

   /*
     Estabelece o modo de exibição a ser utilizado pela janela a ser criada
     neste caso utiliza-se de um buffer simples, ou seja, a apresentação será imediata à
execução
     Define o modo de cores como RGBA
   */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   /*
```

```
    Determina o tamanho em pixels da
    janela a ser criada
  */
  glutInitWindowSize (250, 250);

  /*
    Estabelece a posição inicial para criação da
    janela
  */
  glutInitWindowPosition (100, 100);

  /*
    Cria uma janela com base nos parâmetros especificados
    nas funções  glutInitWindowSize e glutInitWindowPosition
    com o nome de título especificado em seu argumento
  */
  glutCreateWindow ("Exemplo 1");

  /*
     Especifica os parâmetros inicias para as variáveis
     de estado do OpenGL
  */
  init ();

  // Associa a função display como uma função de callback
  glutDisplayFunc(display);

  /*
    Inicia a execução do programa OpenGL.
    O programa irá executar num loop infinito devendo
    o desenvolvedor especificar as condições de saída do mesmo
    através de interrupções no próprio programa ou através
    de comandos de mouse ou teclado como funções de callback
  */
  glutMainLoop();
  return 0;
}
```

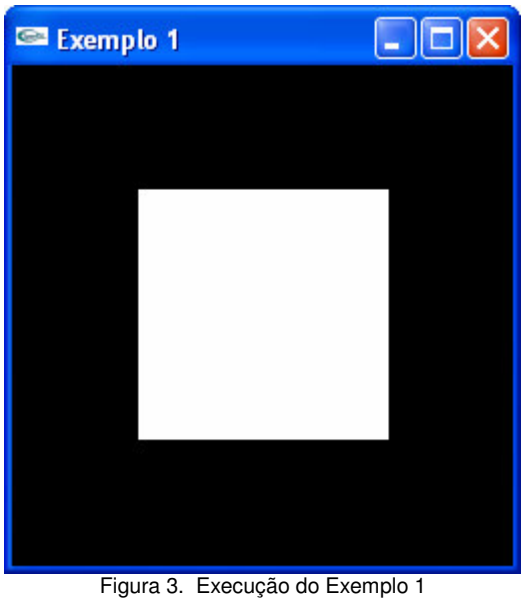



Figura 3. Execução do Exemplo 1

# Capítulo 3 - Criação de Primitivas.

Todas as primitivas geométricas, no OpenGL, são eventualmente descritas em termos de seus vértices – coordenadas que definem os pontos, pontos finais e segmentos de linhas, ou cantos de polígonos – os quais são coordenadas homogêneas [Mason Woo]. Coordenadas homogêneas são da forma ( x, y, z, w ). Dependendo de como os vértices estão organizados, OpenGL pode renderizar e mostrar qualquer primitiva.

O OpenGL apresenta apenas 10 tipos de primitivas distintas, porém com a devida organização destas primitivas é possível a criação de estruturas mais complexas.

## 3.1 - Pontos

Um ponto é representado por um conjunto de números de pontos flutuantes chamados vertex. Todos cálculos internos são feitos como se os vértices fossem tridimensionais. Vértices especificados por um usuário como bidimensional (isto é, somente com as coordenadas x e y) têm a coordenada z atribuída como 0 pelo OpenGL.

Como descrito anteriormente o OpenGL trabalha com coordenadas homogêneas de geometria projetiva tridimensiona. Então para cálculos internos, todos os vértices são representados com quatro coordenadas de ponto-flutuante (x, y, z, w). Se w é diferente de zero, estas coordenadas correspondem a pontos tridimensionais Euclidianos (x/w, y/w, z/w).[Woo 1997].

## 3.2 - Linhas

Em OpenGL, o termo linha refere-se a segmento de linha; não é, portanto, a versão matemática que estende para o infinito em ambas as direções. Existem formas fáceis para especificar uma serie conecta de segmentos de linhas, ou uma série fechada de segmentos. Em todos os casos, portanto, as linhas que constituem a série conectada, são especificadas nos termos dos seus vértices até seus pontos finais..

## 3.3 - Polígonos

Polígonos são áreas fechadas por um loop simples de segmentos de linhas, onde os segmentos de linhas são especificados por vértices e seus pontos finais. Polígonos são tipicamente desenhados com os pixels do seu interior preenchidos, mas também podem ser desenhados como não preenchidos ou como um conjunto de pontos.[Woo 1997]

No geral, polígonos podem ser complicados, então OpenGL faz algumas restrições no que constitui esta primitiva. Primeiramente, as bordas de um polígono OpenGL não podem cruzar-se (Matematicamente deve ser um *polígono simples*). Segundo, os polígonos de OpenGL devem ser convexos, significando que não podem ter recortes. Indicada precisamente, uma região é convexa se dado quaisquer dois pontos no interior, a linha segmento que os une estiver também no interior.[Woo 1997]

A razão para as limitações de OpenGL em tipos válidos de polígonos é que é mais simples fornecer rapidamente um hardware de renderização de polígono para essa classe restrita dos polígonos. Os polígonos simples podem ser renderizados rapidamente. Outros casos são difíceis de detectar rapidamente. Assim para o desempenho máximo, OpenGL supõe que os polígonos são simples.[Woo 1997]

## 3.4 - Desenhando Primitivas

No OpenGL todos objetos geométricos são descritos como um jogo ordenado de vértices. Para tal operação é utilizada a função glVertex*() .[Woo 99]. (onde * representa o parâmetro relativo ao número de componentes e tipos de dados a serem utilizados no desenho. Conforme descrito no capítulo anterior.)

Exemplo :

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);
GLdouble dvect[3] = {5.0, 9.0, 1992.0};
glVertex3dv(dvect);
```

Uma vez que os vértices estão especificados, é necessário dizer ao OpenGL que tipo de primitiva será criada. Para que isto aconteça é necessário que a especificação dos vértices estejam entre o par de comandos glBegin() e glEnd(). O parâmetro do comando glBegin() será o tipo da primitiva a ser desenhada. Os tipos possíveis para parâmetro são :

| Valor do parâmetro | Significado |
|---|---|
| GL_POINTS | Pontos Individuais |
| GL_LINES | Pares de vértices interpretados como segmentos de linha individuais |
| GL_LINE_STRIP | Série de segmentos de linha conectados |
| GL_LINE_LOOP | Como o anterior, porém com um segmento adicionado entre último e primeiro vértices |
| GL_TRIANGLES | Triplos vértices interpretados como triângulos |
| GL_TRIANGLE_STRIP | Faixas de triângulos unidas |
| GL_TRIANGLE_FAN | Leque de triângulos unidos |
| GL_QUADS | Quádruplo de vértices interpretados como polígonos de quatro lados |
| GL_QUAD_STRIP | Faixa quadrilateral unida |
| GL_POLYGON | Limite de um polígono simples, convexo |

Tabela 2. Tipos de primitivas

O comando glEnd() marca o fim de uma lista de dados de vértices.

A figura abaixo mostra o exemplo desenhado de todas as primitivas acima :

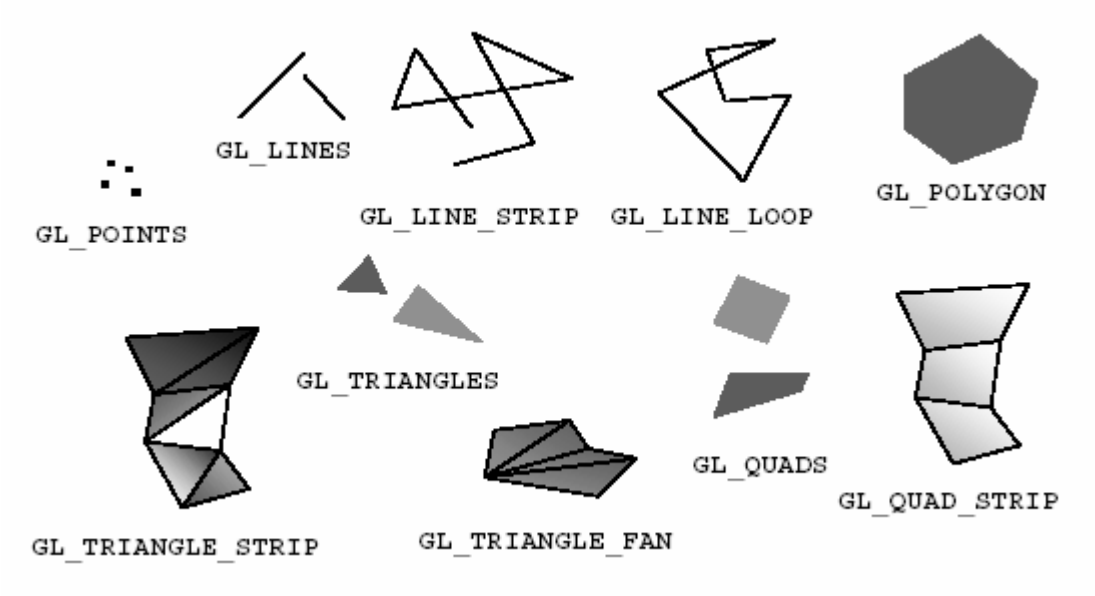



Figura 4. Tipos de primitivas

## 3.5 - Programa exemplo

Neste exemplo serão utilizados todos os tipos de primitivas suportadas pelo OpenGL. A base dos programas será o programa exemplo do tópico 2.4.2 - Exemplo de um programa OpenGL. A função display() deverá ter a porção de código relativa ao desenho da primitiva alterada para tratamento de cada tipo.

### GL_POINTS

```
glBegin(GL_POINTS);
  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();
```

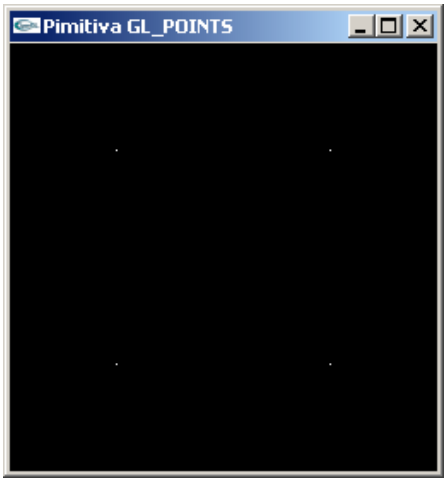



Figura 5. GL_POINTS

## GL_LINES

```
glBegin(GL_LINES);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
glEnd();
```

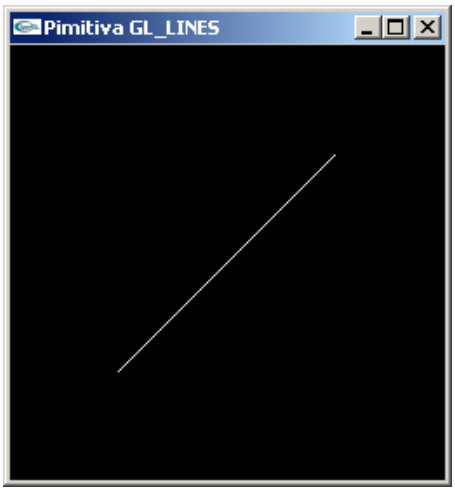



Figura 6. GL_LINES

## GL_LINE_STRIP

```
glBegin(GL_LINE_STRIP);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.30, 0.45, 0.0);
glEnd();
```

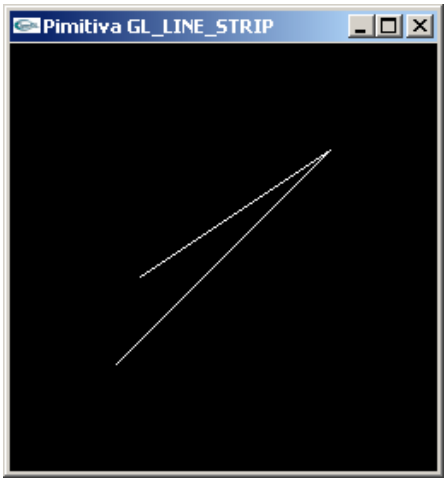



Figura 7. GL_LINE_STRIP

## GL_LINE_LOOP

```
glBegin(GL_LINE_LOOP);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.30, 0.45, 0.0);
glEnd();
```

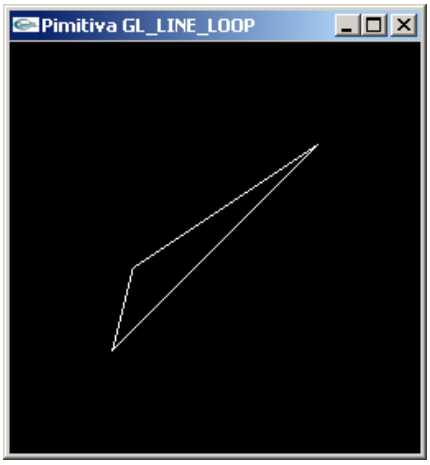



Figura 8. GL_LINE_LOOP

## GL_TRIANGLES

```
glBegin(GL_TRIANGLES);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.30, 0.45, 0.0);
glEnd();
```


Figura 9. GL_TRIANGLES

## GL_TRIANGLE_STRIP

```
glBegin(GL_TRIANGLE_STRIP);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.30, 0.45, 0.0);
  glVertex3f (0.45, 0.12, 0.0);
glEnd();
```

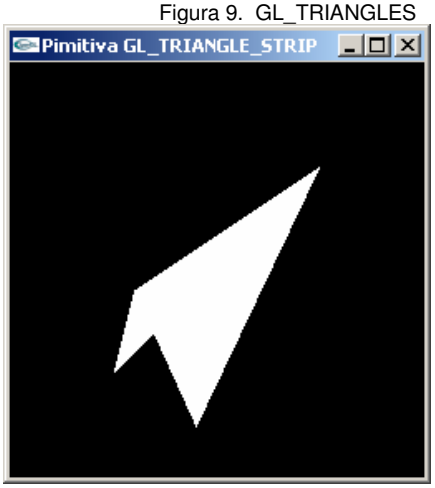



Figura 10. GL_TRIANGLE_STRIP

## GL_TRIANGLE_FAN

```
glBegin(GL_TRIANGLE_FAN);
  glVertex3f (0.25, 0.25, 0.0);
  glVertex3f (0.75, 0.75, 0.0);
  glVertex3f (0.30, 0.45, 0.0);
  glVertex3f (0.45, 0.12, 0.0);
glEnd();
```

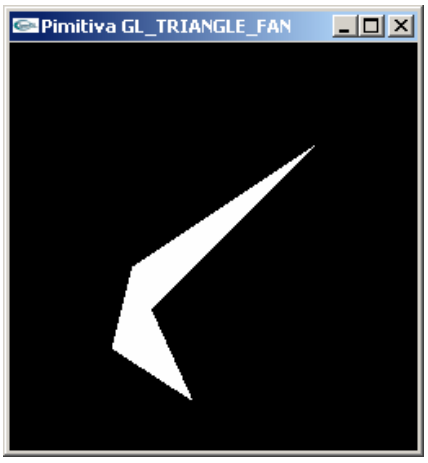



Figura 11. GL_TRIANGLE_FAN

## GL_QUADS

```
glBegin(GL_QUADS);
  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();
```

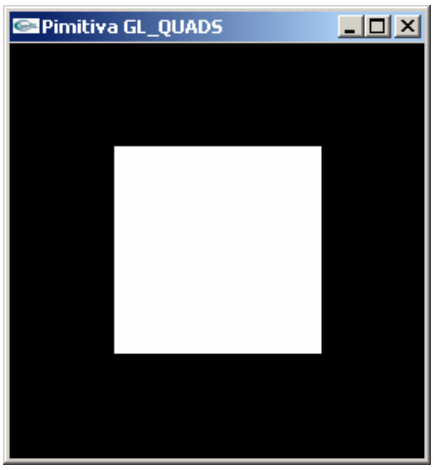



Figura 12. GL_QUADS

## GL_QUADS_STRIP

```
glBegin(GL_QUAD_STRIP);
  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();
```

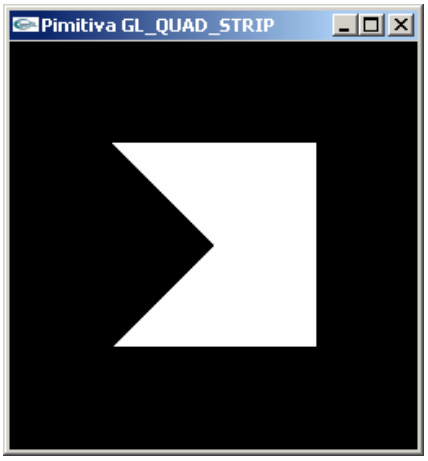



Figura 13. GL_QUADS_STRIP

## GL_POLYGON

```
glBegin(GL_POLYGON);
   glVertex3f( 0.10, 0.10 ,0.0 );
   glVertex3f( 0.10, 0.30,0.0);
   glVertex3f( 0.40, 0.30,0.0);
   glVertex3f( 0.60, 0.30,0.0);
   glVertex3f( 0.40, 0.10,0.0);
glEnd();
```

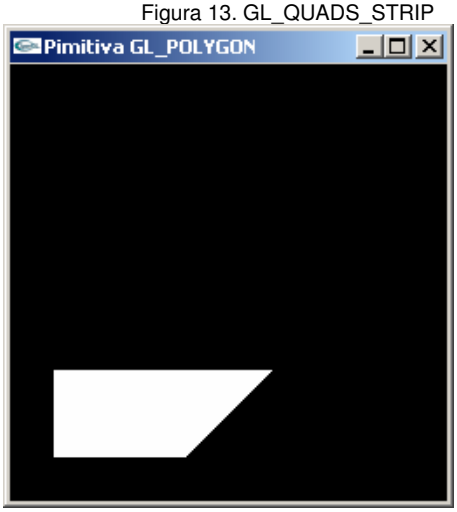



Figura 14. GL_POLYGON

# Capítulo 4 - Cores

Cor simplesmente é um comprimento de onda de luz que é visível ao olho humano.

Fisicamente a luz é composta de fótons – pequenas partículas de luz, cada uma viajando ao longo do seu próprio caminho, e cada uma vibrando em sua própria freqüência ( ou comprimento de onda, ou energia ). Um fóton é completamente caracterizado por sua posição, direção e freqüência . Fótons com comprimento de rondas que percorrem aproximadamente 390 nanômetros (nm) (violeta) e 720 nm (vermelho) cobrem o espectro de cores visíveis, formando as cores do arco-íris (violeta, índigo, azul, verde, amarelo, laranja e vermelho ). Entretanto, os olhos humanos percebem além das cores do arco-íris – rosa, dourado , por exemplo.

## 4.1 - Percepção de cores pelo olho humano

O olho humano percebe as cores quando certas células na retina (chamadas células cone, ou somente cones) se tornam excitadas depois de serem atingidas por fótons. Os três diferentes tipos de células cone respondem melhor a três comprimentos de ondas de luz diferentes : um tipo de célula cone responde melhor a luz vermelha, um tipo para o verde, e outro para o azul. Quando uma determinada mistura de fótons entra no olho, as células cone na retina registram graus diferentes de excitação que depende dos seus tipos, e se uma mistura de diferentes fótons entra, pode acontecer a excitação dos três tipos de cone em um mesmo grau, sua cor será indistinguível na primeira mistura.

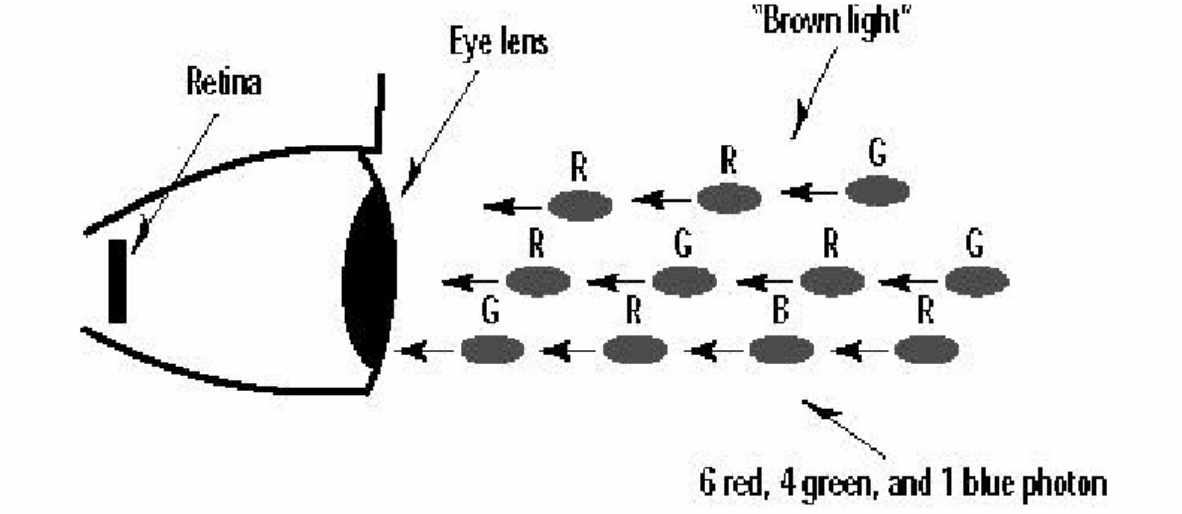



Figura 15. Percepção de cores pelo olho humano

## 4.2 - Cores no Computador

Um monitor de computador emula as cores visíveis iluminando pixels com uma combinação de vermelho, verde, e luz azul (também chamado RGB ) em proporções que excitam os cones sensíveis ao: vermelho, verde e cone azul na retina de tal modo que compatibiliza os níveis de excitação gerados pela mistura de fóton que se está tentando emular.

Para exibir uma cor especifica, o monitor envia certas quantias de luz vermelha, verde e azul para adequadamente estimular os tipos diferentes de células cone do olho humano. Um monitor colorido pode enviar diferentes proporções de vermelho, verde, e azul para cada pixel, e o olho vê milhões de cores ou assim define os pontos de luz, cada um com sua própria cor.

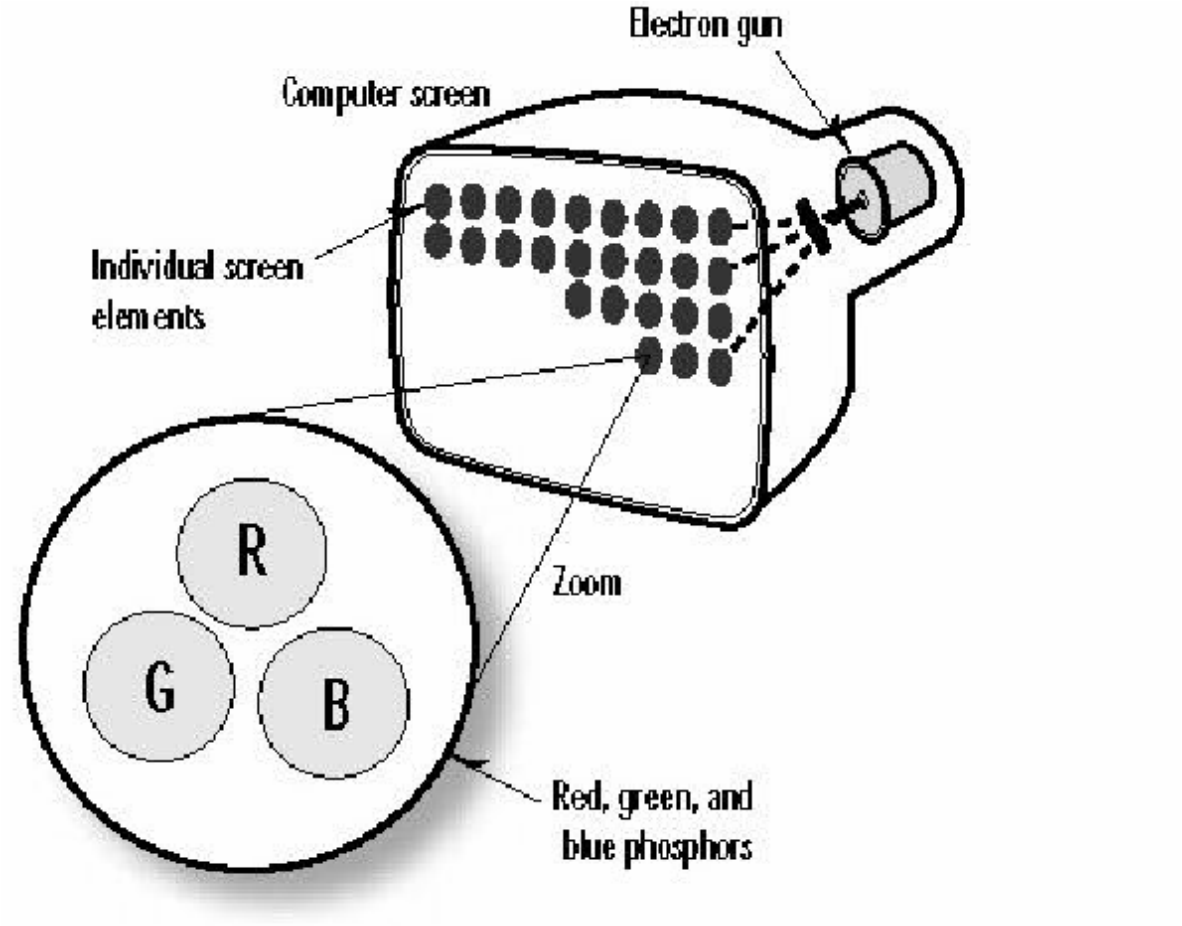



Figura 16. Simulação de cores no computador

## 4.3 - Cores no OpenGL

No OpenGL é possível trabalhar com cores de dois modos diferente : Modo RGBA e Modo de Índice de Cores.

Antes da execução de um programa o modo de exibição de cores deverá ser definido como RGBA ou Índice de Cores. Uma vez que o modo de exibição é inicializado, não poderá ser modificado. Como o programa executa uma cor (ou um índice de cores ou um valor de RGBA) é determinado em uma base de vértices para cada primitiva geométrica. Esta cor ou é uma cor especificada explicitamente para um vértice ou, se a iluminação está habilitada, é determinada pela interação das matrizes de transformação com as normais da superfície e outras propriedades dos materiais.

### 4.3.1 - Escolhendo entre RGBA e Índice de Cores

Por prover maior flexibilidade que o modo de Índice de Cores, normalmente o modo de cores utilizado é o RGBA. Porém o modo de Índice de Cores pode ser escolhido nos seguintes casos :

- No caso de se estar portando uma aplicação existente que já utilize Índice de Cores
- Se houver apenas um pequeno número de "bitplanes" disponíveis, o modo RGBA pode produzir sombras de cores. Exemplo se houver apenas 8 "bitplanes", no modo RBGA, só existirão 3 partes para o vermelho, 3 para o verde e 2 para o azul. Assim existirão 8 sombras de vermelho e verde e somente 4 sombras de azul. É provável que os gradientes entre as sombras não apresentem perfeição .
- Este modo pode ser muito útil em efeitos como, animações de mapeamento de cores e desenho de camadas.

### 4.3.2 - Definindo o Modo de cores

O comando para definição do modo de cores é :

***void glutInitDisplayMode(unsigned int mode);***

O modo deverá ser um ou mais da seguinte tabela :

| Modo | Descrição |
|---|---|
| GLUT_RGBA | Seleção do modo RGBA para a janela. Se nem o parâmetro GLUT_RGBA ou GLUT_INDEX forem definidos ele será o padrão |
| GLUT_RGB | Apelido para o GLUT_RGBA |

| | |
|---|---|
| GLUT_INDEX | Modo de Índice de Cores. A especificação deste modo anula o modo GLUT_RGBA se este também for especificado como modo |
| GLUT_SINGLE | Seleciona apenas um buffer para janela. Este é o valor padrão se nem GLUT_DOUBLE ou GLUT_SINGLE forem especificados. |
| GLUT_DOUBLE | Seleciona dois buffers para a janela. Sobrepõe o GLIT_SINGLE se especificado |
| GLUT_ACCUM | Ativa o modo de acumulação de buffer para a janela |
| GLUT_ALPHA | Ativa a componente ALFA de cor para a janela |
| GLUT_DEPTH | Ativa o buffer de profundidade para a janela |
| GLUT_STENCIL | Ativa o buffer de estêncil |
| GLUT_MULTISAMPLE | Selecione uma janela com opção de “multisampling”. Se “multisampling” não está disponível, uma janela de “non-multisampling” será escolhida automaticamente |
| GLUT_STEREO | Seleção de uma janela no modo estéreo |
| GLUT_LUMINANCE | Selecione uma janela com um modelo de cores de Luminancia. Este modo provê a funcionalidade do RGBA de OpenGL, mas os componentes verdes e azuis não são mantidos no frame buffer. Ao invés disso o componente vermelho de cada pixel é convertido para um índice entre zero e glutGet (GLUT_WINDOW_COLORMAP_SIZE)-1 e é observado em um mapa de cores por janela para determinar a cor de pixels dentro da janela. O colormap inicial de janelas de GLUT_LUMINANCE é inicializado para |

| | |
|---|---|
| | ser uma escala de cinza linear, mas pode ser modificado com as rotinas de colormap da GLUT |

Tabela 3. Modos de cores

Para especificar a cor em um determinado vértice o comando a ser utilizado é o seguinte :

**glColor*(),** onde o * representa o sufixo do número de coordenadas, o tipo de dados e o vetor. (Os parâmetros poderão variar de acordo com a quantidade de coordenadas, e o tipo dos mesmos de acordo com o tipo especificado no sufixo.)

A faixa de valores de cores é representada por valores de ponto flutuante que variam de 0 a 1, e esta é a faixa de valores que pode ser armazenada no framebuffer.

O OpenGL faz a conversão dos tipos de dados para valores de ponto flutuante.A tabela abaixo apresenta as faixas de valores para os parâmetros das cores, para cada tipo de representação de dados dos parâmetros, e o valor máximo e mínimo da conversão interna do OpenGL :

| **Sufixo** | **Tipo de Dado** | **Valor Mínimo** | **Valor mínimo para mapeamento** | **Valor Máximo** | **Valor máximo para mapeamento** |
|---|---|---|---|---|---|
| B | Inteiro de 1 byte | -128 | -1.0 | 127 | 1.0 |
| s | Inteiro de 2 bytes | -32768 | -1.0 | 32767 | 1.0 |
| i | Inteiro de 4 bytes | -2147483648 | -1.0 | 2148483647 | 1.0 |
| ub | Inteiro não sinalizado de 1 byte | 0 | 0.0 | 255 | 1.0 |
| us | Inteiro não sinalizado de 2 bytes | 0 | 0.0 | 65535 | 1.0 |
| Ui | Inteiro não sinalizado de | 0 | 0.0 | 4294967295 | 1.0 |

| | | | | | |
|---|---|---|---|---|---|
| | 4 bytes | | | | |

Tabela 4. Faixa de valores para conversão de cores

## 4.3.3 - Exemplo da utilização de cores

```
/* Exemplo2.c – Marcionílio Barbosa Sobrinho
* Programa simples que apresenta o desenho de um quadrado com variação de cores nos vertices
* Objetivo : Demonstrar a utilização de cores nos objetos
* Referência do Código: OpenGL Programming Guide – RedBook
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

GLfloat vermelho;
GLfloat verde;
GLfloat azul;

void display(void)
{
   /* Limpa o Buffer de Pixels */
   glClear (GL_COLOR_BUFFER_BIT);

   // Define a cor padrão com base nos parametros
   glColor3f (vermelho, verde, azul);

   /* desenha um simples retângulo com as coordenadas
   * (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();

   /* Inicia o processo de desenho através dos
      dados bufferizados
   */
   glFlush ();
}

void init (void)
{

     /* Define os parâmetros de cores para obter a cor branca */
     vermelho = 1.0;
     verde = 1.0;
     azul = 1.0;

    /* Seleciona a cor de fundo para limpeza da tela */
    glClearColor (0.0, 0.0, 0.0, 0.0);

    /* inicializa os valores de visualização */
    glMatrixMode(GL_PROJECTION);

    /* Faz com que a matriz corrente seja inicializada com a matriz identidade
       (nenhuma transformação é acumulada)
    */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}
```

```
/*
       Função responsável pelo controle do teclado. Dependendo da tecla pressionada :
       R,G,B, adiciona uma constante ao valor da mesma e redesenha novamente a cena com a
       Nova cor obtida destes pâmetros.
       Se a telca 'O' for pressionada volta o estado original da imagem.

       O operador ternário está comentado por apresentar incompatibilidades
       Com o compilador Dev-C++, porém não existem incompatibilidades com o Borland C++
       Builder nem com MS Visual C++

*/

void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'R':
            case 'r':// Incrementa o valor do parâmetro da cor Vermelho
                     // (vermelho - 0.1) < 0 ? vermelho = 1 : vermelho -= 0.1;
                     vermelho = vermelho - 0.1;
                     if (vermelho < 0)
                        vermelho = 1;

                     break;
            case 'G':
            case 'g':// Incrementa o valor do parâmetro da cor Verde
                     // (verde - 0.1 ) < 0 ? verde = 1 : verde -= 0.1;
                     verde = verde - 0.1;
                     if (verde < 0)
                        verde = 1;

                     break;
            case 'B':
            case 'b':// Incrementa o valor do parâmetro da cor Azul
                     // (azul - 0.1) < 0 ? azul= 1 : azul -= 0.1;
                     azul = azul - 0.1;
                     if (azul < 0)
                        azul = 1;

                     break;
                case 'O':
            case 'o':
                vermelho = 1.0;
                       verde = 1.0;
                azul = 1.0;
           break;

    }
    glutPostRedisplay();
}

/*
    Funcão principal do programa.
*/
int main(int argc, char** argv)
{

   /*
     Estabelece o modo de exibição a ser utilizado pela janela a ser criada
     neste caso utiliza-se de um buffer simples, ou seja, a apresentação será imediata à
execução
     Define o modo de cores como RGBA
   */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   /*
     Determina o tamanho em pixels da
     janela a ser criada
   */
   glutInitWindowSize (250, 250);

   /*
     Estabelece a posição inicial para criação da
     janela
```

```
   */
   glutInitWindowPosition (100, 100);

   /*
     Cria uma janela com base nos parâmetros especificados
     nas funções  glutInitWindowSize e glutInitWindowPosition
     com o nome de título especificado em seu argumento
   */
   glutCreateWindow ("Exemplo 2");

   /*
     Habilita a captura dos eventos de teclado

    */
   glutKeyboardFunc(teclado);


   /*
      Especifica os parâmetros inicias para as variáveis
      de estado do OpenGL
   */
   init ();

   // Associa a função display como uma função de callback
   glutDisplayFunc(display);

   /*
     Inicia a execução do programa OpenGL.
     O programa irá executar num loop infinito devendo
     o desenvolvedor especificar as condições de saída do mesmo
     através de interrupções no próprio programa ou através
     de comandos de mouse ou teclado como funções de callback
   */
   glutMainLoop();
   return 0;
}
```

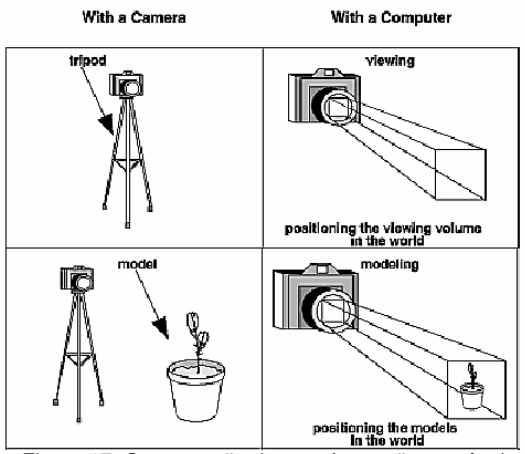
# Capítulo 5 - Transformações

O processo de transformação para produção de uma determinada cena é análogo à uma fotografia tirada com uma máquina fotográfica. [ Woo 1997]

Os passos para a obtenção da foto tanto com uma câmera quanto com o computador talvez sejam os seguintes :

1. Montar o tripé e apontar a máquina fotográfica à cena (transformação de visualização).
2. Organizar a cena a ser fotografada na composição desejada (modelagem de transformação).
3. Escolha de uma lente de máquina fotográfica ou ajuste o zoom (transformação de projeção).
4. Determinar quão grande será a fotografia final - por exemplo, no caso de uma fotografia maior que a cena original (transformação de viewport).

Após estes passos serem executados, a foto poderá ser feita ou a cena poderá ser desenhada [ Woo 1997 ].



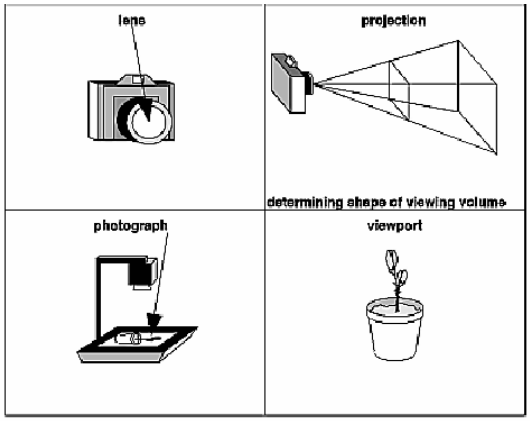




Figura 17. Comparação de transformações : máquina fotográfica x computdador

## 5.1 - Transformação de Objetos

As três rotinas do OpenGL para modelar transformações são: **glTranslate * (), glRotate * ()**, e **glScale * ()**. Estas rotinas transformam um objeto (ou sistema de coordenadas) movendo, girando, estirando, encolhendo, ou refletindo.

A biblioteca gráfica OpenGL é capaz de executar transformações de translação, escala e rotação através de uma multiplicação de matrizes. A idéia central destas transformações em OpenGL é que elas podem ser combinadas em uma única matriz, de tal maneira que várias transformações geométricas possam ser aplicadas através de uma única operação.

**Rotação** :

A rotação é feita através da função glRotatef(Ângulo, x, y, z), que pode receber quatro números float ou double (glRotated) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de rotação de "Ângulo" graus ao redor do eixo definido pelo vetor "x,y,z" no sentido anti-horário

Ex : **glRotatef** (45.0, 0.0, 0.0, 1.0), Rotaciona um objeto num ângulo de 45º

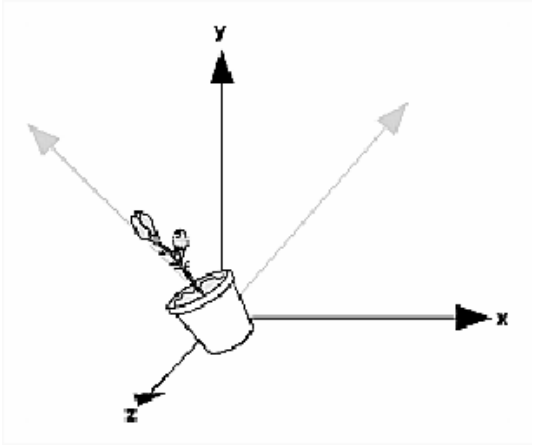



Figura 18. Rotação

**Translação :**

A translação é feita através da função glTranslatef(Tx, Ty, Tz), que pode receber três números float ou double (glTranslated) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de translação baseada nos valores dados.

Ex.:

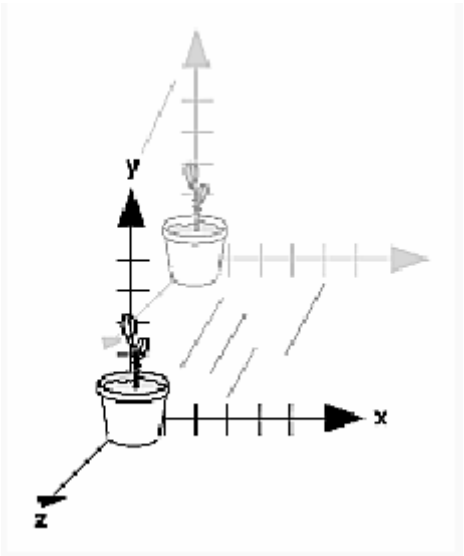



Figura 19. Translação

**Escala :**

A escala é feita através da função glScalef(Ex, Ey, Ez), que pode receber três números float ou double (glScaled) como parâmetro. Neste caso, a matriz atual é multiplicada por uma matriz de escala baseada nos valores dados.

Ex.: Efeito de glScalef(2.0, -0.5, 1.0)

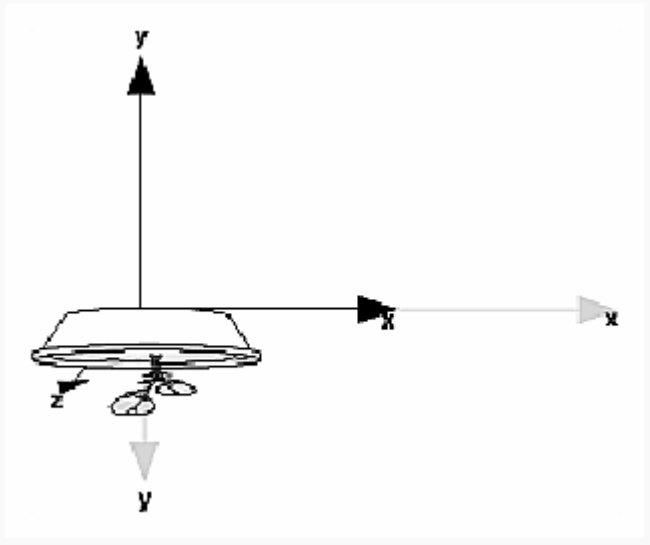



Figura 20. Escala

## 5.1.1 - Exemplo de Transformação de Objetos

```
/* Exemplo3.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta as transformações aplicadas a uma primitiva
* Objetivo : Demonstrar a utilização de transformação de objetos
* Referência do Código: OpenGL Programming Guide - RedBook
*/
#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

GLfloat escala;
GLfloat translada;
GLfloat rotaciona;

void display(void)
{

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   /* Limpa o Buffer de Pixels */
   glClear(GL_COLOR_BUFFER_BIT);
   /* Estabelece a cor da primitiva */
   glColor3f (1.0f,1.0f,1.0f);
   /* Efetua a operação de translação */
   glTranslatef(translada, 0.0f, 0.0f);
   /* Efetua a operação de escala em Y */
   glScalef(1.0f, escala, 1.0f);
   /* Efetua a operação de rotação em Z */
   glRotatef(rotaciona, 0.0f, 0.0f, 1.0f);


   /* desenha um simples retângulo  */

   glBegin(GL_QUADS);
     glVertex3f (0.025, 0.025, 0.0);
     glVertex3f (0.075, 0.025, 0.0);
     glVertex3f (0.075, 0.075, 0.0);
     glVertex3f (0.025, 0.075, 0.0);
   glEnd();


   /* Inicia o processo de desenho através dos
      dados bufferizados
   */
   glFlush ();
}

void init (void)
{

     /* Define os parâmetros de cores para obter a cor branca */
     escala = 1;
     translada = 0;
     rotaciona = 0;

    /* Seleciona a cor de fundo para limpeza da tela */
    glClearColor (0.0, 0.0, 0.0, 0.0);

    /* inicializa os valores de visualização */
    glMatrixMode(GL_PROJECTION);

    /* Faz com que a matriz corrente seja inicializada com a matriz identidade
       (nenhuma transformação é acumulada)
    */
    glLoadIdentity();
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

/*
       Função responsável pelo controle do teclado. Dependendo da tecla pressionada :
       R,S,T, irá efetuar respectivamente as operações de
```

```
        Rotação, Escala e Translação
*/

void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'S':
            case 's':// Incrementa o valor do parâmetro de escala
                     escala = escala + 0.5;
                     break;
            case 'T':
            case 't':// Incrementa o valor do parâmetro de translacao
                     translada = translada + 0.05;

                     break;
            case 'R':
            case 'r':// Incrementa o valor do ângulo de rotação
                     rotaciona = rotaciona - 5.0;
                     break;
                case 'O':
            case 'o':
                translada = 0.0;
                escala = 1.0;
                rotaciona = 0;
           break;

    }
    glutPostRedisplay();
}

/*
    Funcão principal do programa.
*/
int main(int argc, char** argv)
{

   /*
     Estabelece o modo de exibição a ser utilizado pela janela a ser criada
     neste caso utiliza-se de um buffer simples, ou seja, a apresentação será imediata à
execução
     Define o modo de cores como RGBA
   */
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   /*
     Determina o tamanho em pixels da
     janela a ser criada
   */
   glutInitWindowSize (500, 500);

   /*
     Estabelece a posição inicial para criação da
     janela
   */
   glutInitWindowPosition (100, 100);

   /*
     Cria uma janela com base nos parâmetros especificados
     nas funções  glutInitWindowSize e glutInitWindowPosition
     com o nome de título especificado em seu argumento
   */
   glutCreateWindow ("Exemplo 3 - Transformações");

   /*
     Habilita a captura dos eventos de teclado

    */
   glutKeyboardFunc(teclado);


   /*
      Especifica os parâmetros inicias para as variáveis
      de estado do OpenGL
   */
```

```
   init ();

   // Associa a função display como uma função de callback
   glutDisplayFunc(display);

   /*
     Inicia a execução do programa OpenGL.
     O programa irá executar num loop infinito devendo
     o desenvolvedor especificar as condições de saída do mesmo
     através de interrupções no próprio programa ou através
     de comandos de mouse ou teclado como funções de callback
   */
   glutMainLoop();
   return 0;
}
```

## 5.2 - Transformação de Visualização

Uma transformação de visualização muda a posição e orientação do ponto de vista. Como exemplificado pela analogia com a máquina fotográfica, a transformação de visualização posiciona o tripé de máquina fotográfica e aponta a máquina fotográfica para o modelo.

Da mesma maneira que uma máquina fotográfica é movida para alguma posição e é girada até que aponte na direção desejada., transformações estão geralmente compostas de translações e rotações. Para alcançar uma certa composição de cena na imagem final ou fotográfica ou o objeto pode ser movimentado ou a máquina fotográfica, na direção oposta. Assim, uma transformação que gira um objeto à esquerda é equivalente a uma transformação de visão que gira a máquina fotográfica à direita, por exemplo. Os comandos de transformação de visão devem ser chamados antes de qualquer execução de transformações de modelagem.

O comando responsável por este tipo de transformações no OpenGL é :

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

Este comando define uma matriz de visão e a multiplica pela matriz atual. O ponto de vista desejado é especificado por eyex, eyey, e eyez. O centerx, centery, e argumentos de centerz especificam qualquer ponto ao longo da linha desejada de visão, mas tipicamente eles são algum ponto no centro da cena a ser

visualizada  O upx, upy, e argumentos de upz indicam qual direção é para cima (quer dizer, a direção  do fundo para o topo do volume de visualização).

## 5.2.1 - Exemplo de Transformação de Visualização

```
/* Exemplo4.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta as transformações de visualização em uma cena
* Objetivo:Demonstrar a utilização de transformação de visualização com o comando glLookAt()
* Referência do Código: OpenGL Programming Guide - RedBook
*                       Example 3-1 : Transformed Cube: cube.c
*/
#include <windows.h>
#include <GL/glu.h>
#include <GL/glut.h>

GLfloat eyex,eyey,eyez,centrox, centroy, centroz;

void init(void)
{
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
   eyex = 0.0;
   eyey = 0.0;
   eyez = 5.0;
   centrox=0.0;
   centroy=0.0;
   centroz=0.0;
}
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glColor3f (1.0, 1.0, 1.0);
   glLoadIdentity (); /* clear the matrix */

   /* viewing transformation */
   gluLookAt (eyex, eyey, eyez, centrox, centroy, centroz, 0.0, 1.0, 0.0);
   glutWireCube (1.0);
   glFlush ();
}

void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glFrustum (-1.0, 1.0, -1.0, 1.0, 1.5, 20.0);
   glMatrixMode (GL_MODELVIEW);
}

/* Controla os eventos de teclado
   ao se pressionar as teclas  x, y ou z
   os valores do parametro relativo a eye da
   função glLookAt serão modificados se
   as teclas forem x, y ou z entao o valor do centro é modificado
*/

void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'x':// Incrementa o valor de eyex
                     eyex = eyex + 0.5;
                     break;
            case 'y':// Incrementa o valor de eyey
                     eyey = eyey + 0.5;
                     break;
            case 'z':// Incrementa o valor de eyez
```

```
                    eyez = eyez + 0.5;
                    break;
             case 'X':// Incrementa o valor de centrox
                    centrox = centrox + 0.5;
                    break;
             case 'Y':// Incrementa o valor de centroy
                    centroy = centroy + 0.5;
                    break;
             case 'Z':// Incrementa o valor de centroz
                    centroz = centroz + 0.5;
                    break;
            case 'O':
             case 'o':
                 eyex = 0.0;
                 eyey = 0.0;
                 eyez = 5.0;
                 centrox=0.0;
                 centroy=0.0;
                 centroz=0.0;

            break;

    }
    glutPostRedisplay();
}

/*Programa principal */

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 4 - Visualização de Transformação");
  glutKeyboardFunc(teclado);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

## 5.3 - Transformação de Projeção

O propósito da transformação de projeção é definir um volume de visualização, que é usado de dois modos. O volume de visualização determina como um objeto é projetado sobre a tela (quer dizer, usando perspectiva ou projeção ortográfica), e define quais são os objetos ou porções de objetos que serão cortados da imagem final.

### 5.3.1 - Projeção Perspectiva

A característica principal da projeção de perspectiva é a seguinte : quanto mais longe um objeto está da máquina fotográfica, menor aparece na imagem final. Isto acontece porque o volume de visualização para a projeção de

perspectiva é um frustum de uma pirâmide (uma pirâmide sem o topo ). Objetos que são mais próximos do ponto de vista aparecem maiores porque eles ocupam proporcionalmente quantia maior do volume de visão do que os que estão mais distantes. Este método de projeção é comumente usado para animação visual, simulação, e qualquer outra aplicação que exige algum grau de realismo porque é semelhante à forma de visualização do olho humano (ou uma máquina fotográfica)
.

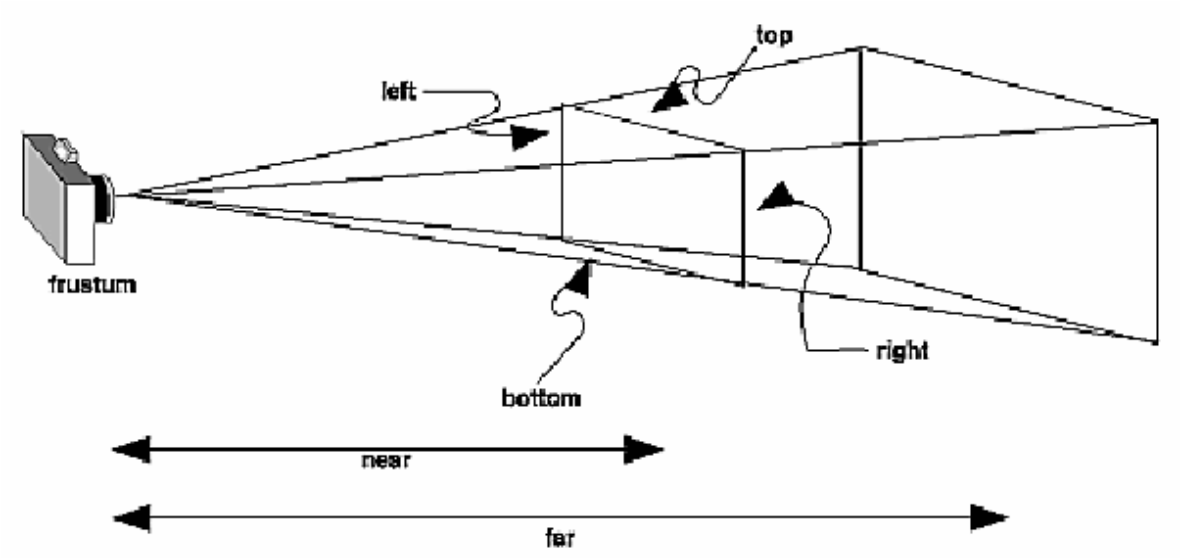


Figura 21. Frustrum

O comando para definir um frustum é:

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

Este comando cria uma matriz para um frustum de visão perspectiva e multiplica pela matriz atual. O frustum do volume de visão é definido pelos parâmetros: (left, botton, -near) e (right, top, -near) pelas coordenadas (x, y, z) específica do canto inferior esquerdo e canto de superior-direito do próximo plano de recorte ; near e far dão as distâncias do ponto de vista para o near e far do plano de recorte. Eles sempre devem ser positivos.

Embora seja fácil entender conceitualmente, glFrustum () não é intuitivo para seu uso. Normalmente em OpenGL a função gluPerspective () é utilizada. Esta rotina cria um volume de visão da mesma forma que glFrustum () faz, mas especificação é feita de um modo diferente.

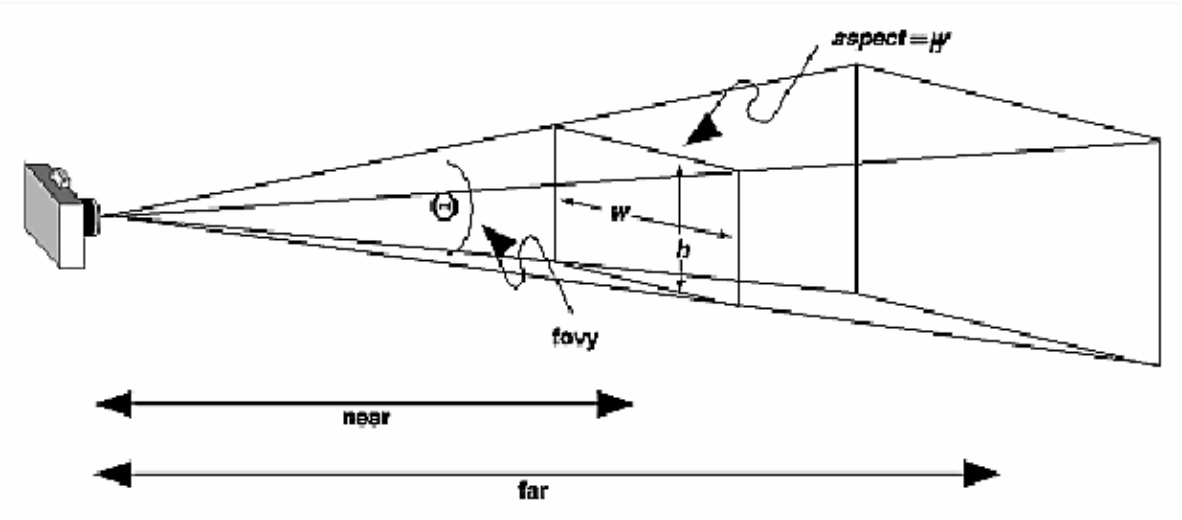



Figura 22. Projeção perspectiva

A sintaxe do comando é :

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

Este comando cria uma matriz para um frustum de visão perspectiva simétrica e multiplica pela matriz atual. fovy é o ângulo do campo de visão no plano de x-z; seu valor deve estar entre [0.0,180.0]. aspecto é a relação de aspecto do frustum, sua largura dividida por sua altura. near e far são distâncias entre o ponto de vista e os planos de corte, ao longo do z-eixo negativo. Eles devem sempre ser positivos.

## 5.3.2 - Projeção Ortográfica

Na projeção ortográfica, o volume de visão é um paralelepípedo retangular, ou mais informalmente, uma caixa. Diferentemente da projeção de perspectiva, o tamanho do volume de visão não, assim a distância da máquina fotográfica não afeta como o tamanho do objeto. Este tipo de projeção é usado para aplicações como criar plantas arquitetônicas e projetos CAD , onde é crucial manter os tamanhos atuais de objetos e ângulos entre eles da mesma forma como eles serão projetados.

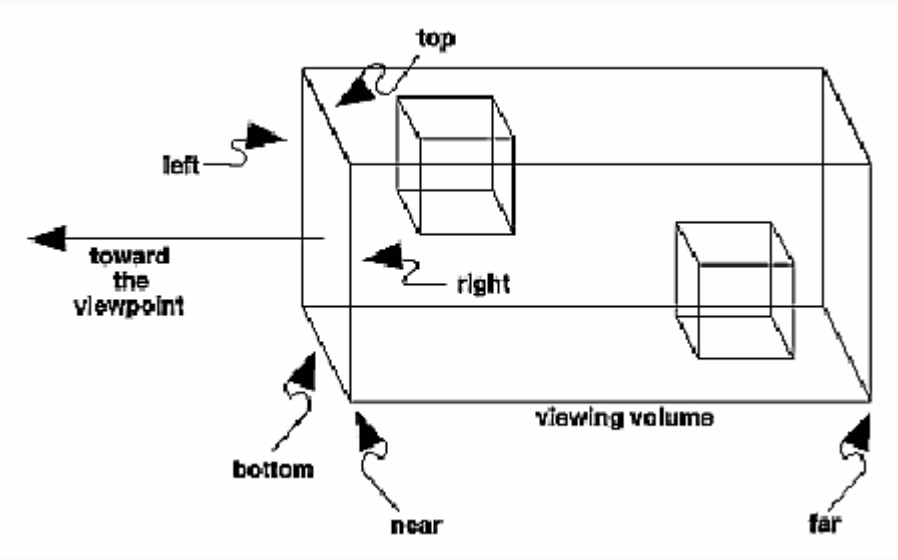



Figura 23. Projeção ortográfica

O comando para criar uma visão ortografica paralela é :

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

Este comando cria uma matriz para um volume de visão paralelo ortográfico e multiplica a pela matriz atual. (left, botton, -near) e (right, top, -near) são os pontos próximos ao plano de corte que é traçado no canto inferior esquerdo e canto superior direito da janela de visualização , respectivamente. Ambos near e far podem ser positivos ou negativos.

O exemplo 5.1.1 apresenta este tipo de projeção.

## 5.4 - Transformação de Viewport

Recordando a analogia feita com a máquina fotográfica, a transformação de viewport corresponde à fase onde o tamanho da fotografia desenvolvida é escolhido. O viewport é a região retangular da janela onde a imagem é desenhada. O viewport é medido em coordenadas de janela que refletem a posição relativa de pixels na tela do canto inferior esquerdo da janela.

O sistema de janela, não OpenGL, é responsável por abrir uma janela na tela. Porém, por deixar de existir o viewport é fixado ao retângulo de pixel inteiro da janela que é aberta. O comando glViewport () é usado para escolher uma região de desenho menor; por exemplo, pode subdividir a janela ,criar um efeito de dividir-tela para visões múltiplas na mesma janela.

Sintaxe do comando :

void **glViewport**(GLint x, GLint y, GLsizei width, GLsizei height);

Define um retângulo de pixel na janela na qual a imagem final é desenhada. O parâmetro (x, y) especifica o canto inferior-esquerdo do viewport, e largura e altura são o tamanho do retângulo do viewport .Os valores de viewport iniciais por padrão são (0, 0, winWidth, winHeight), onde winWidth e winHeight são o tamanho da janela.. (Caso não seja especificado o viewport).

# Capítulo 6 - Formas 3D

Formas 3D no computador são imagens realmente bi-dimensionais, apresentadas em uma tela de computador plana apresentando uma ilusão de profundidade, ou uma tri “dimensão.” Para verdadeiramente se ver em 3D, é necessário que de fato se veja o objeto com ambos os olhos.

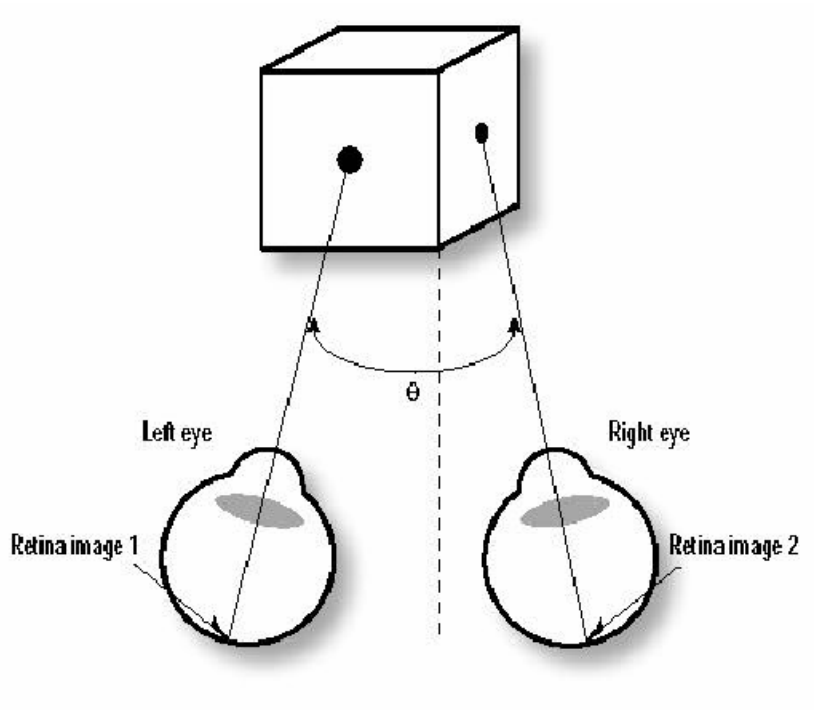



Figura 24. Percepção tri-dimensional pelo olho humano

Cada olho recebe uma imagem bi-dimensional que é como se fosse uma fotografia temporária na retina (a parte de trás do olho). Estas duas imagens são ligeiramente diferentes porque elas são recebidas em dois ângulos diferentes (os olhos são espaçados separadamente de propósito). O cérebro combina estes, então imagens ligeiramente diferentes, para produzir um único quadro composto 3D. [Richard Wright 1999]

Essencial para a visualização e modelagem tri-dimensional com o OpenGL, as transformações têm papel fundamental neste tipo de visualização. Assim conforme descrito no Capítulo 5 as transformações necessárias para modelagem e visualização 3D seguem a analogia à máquina fotográfica.

As bibliotecas GLU e GLUT possuem uma série de funções para desenhar primitivas 3D, tais como esferas, cones, cilindros e teapot, além de permitir a criação de outras formas através do conjunto de primitivas disponíveis no OpenGL.

## 6.1 - Exemplo de Forma 3D

```
/* Exemplo5.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta uma forma 3D em uma cena
* Objetivo:Demonstrar a utilização de objetos 3D
* Referência do Código: OpenGL Super Bible
*                       Program - teapot.c
*/
#include <window.h>
#include <GL/glu.h>
#include <GL/glut.h>

GLfloat angle, fAspect,eyex, eyey, eyez;


void init(void)
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    angle=45;
    eyex = 0.0;
    eyey = 80.0;
    eyez = 200.0;


}

void visao(void)
{
 // Especifica sistema de coordenadas de projeção
 glMatrixMode(GL_PROJECTION);

 // Inicializa sistema de coordenadas de projeção
 glLoadIdentity();

 // Especifica a projeção perspectiva
 gluPerspective(angle,fAspect,0.1,500);

 // Especifica sistema de coordenadas do modelo
 glMatrixMode(GL_MODELVIEW);

 // Inicializa sistema de coordenadas do modelo
 glLoadIdentity();

 // Especifica posição do observador e do alvo
 gluLookAt(eyex,eyey,eyez, 0,0,0, 0,1,0);


}


void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(1.0f, 0.0f, 0.0f);

   // Desenha a esfera com a cor corrente (wire-frame)
   glutWireSphere  ( 20 , 25 , 25 );

   // Executa os comandos OpenGL
```

```
   glutSwapBuffers();

}

void reshape (GLsizei w, GLsizei h)
{
    // Para previnir uma divisão por zero
    if ( h == 0 ) h = 1;
        // Especifica o tamanho da viewport
        glViewport(0, 0, w, h);

    // Calcula a correção de aspecto
    fAspect = (GLfloat)w/(GLfloat)h;

    visao();
}

/* Controla os eventos de teclado
   ao se pressionar as teclas  X, Y ou Z
   os valores do parametro relativo a eye da
   função glLookAt serão modificados
*/

void teclado(unsigned char tecla, int x, int y)
{
    switch (tecla) {
            case 'x':
            case 'X':// Incrementa o valor de eyex
                     eyex = eyex + 5;
                     break;
            case 'y':
            case 'Y':// Incrementa o valor de eyey
                     eyey = eyey + 5;

                     break;
            case 'Z':
            case 'z':// Incrementa o valor de eyez
                     eyez = eyez + 5;
                     break;
           case 'O':
            case 'o':
                     eyex = 0.0;
                     eyey = 80.0;
                     eyez = 200.0;
          break;

    }
    visao();
    glutPostRedisplay();
}

void GMouse(int button, int state, int x, int y)
{
       if (button == GLUT_LEFT_BUTTON)
              if (state == GLUT_DOWN) {  // Zoom-in
                     if (angle >= 10) angle -= 5;
              }
       if (button == GLUT_RIGHT_BUTTON)
              if (state == GLUT_DOWN) {  // Zoom-out
                     if (angle <= 130) angle += 5;
              }
       visao();
       glutPostRedisplay();
}


/*Programa principal */

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
```

```
  glutCreateWindow ("Exemplo 5 - Visualização 3D Wireframe");
  glutKeyboardFunc(teclado);
  glutMouseFunc(GMouse);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
  return 0;
}
```

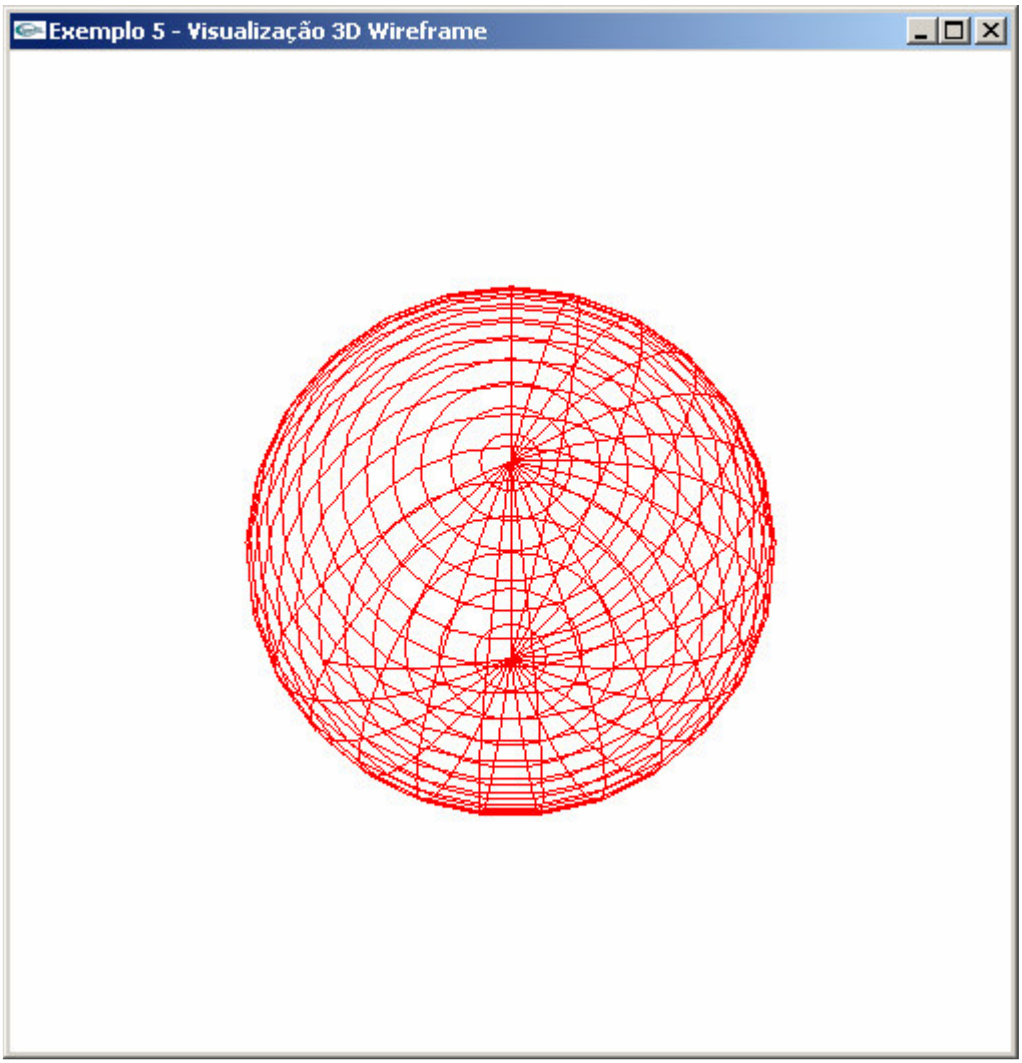



Figura 25. Esfera wire-frame

## 6.2 - Formas 3D pré-definidas no GLUT

O GLUT tem as seguintes formas geométricas pré-definidas :

- Esfera;
- Cubo;
- Cone;
- Toróide;

- Dodecaedro;
- Octaedro;
- Tetraedro;
- Icosaedro;
- Teapot

### 6.2.1 - Esfera :

void **glutSolidSphere**(GLdouble radius, GLdouble slices, GLdouble stack )
void **glutWireSphere**(GLdouble radius, GLdouble slices, GLdouble stack)

Parâmetros :

radius :Raio da Esfera
slices : Número de subdivisões ao redor do eixo Z (linhas de longitude)
stack : Número de subdivisões ao longo do eixo Z (linhas de latitude).

### 6.2.2 – Cubo

void **glutSolidCube** (GLdouble size )
void **glutWireCube** (GLdouble size)

Parâmetros :

size :Tamanho do cubo

Exemplo : glutWireCube(20);

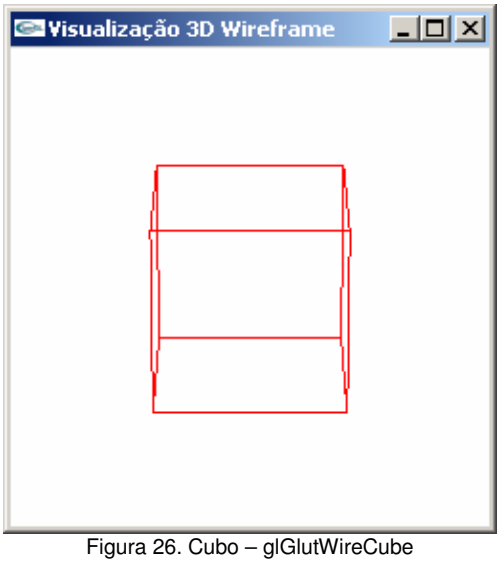


Figura 26. Cubo – glGlutWireCube

### 6.2.3 – Cone

void **glutSolidCone**(GLdouble base ,GLdouble height,Glint slices,Glint stacks)
void **glutWireCone**(GLdouble base ,GLdouble height,Glint slices,Glint stacks)

Parâmetros :

base:Raio da base do cone
height: A altura do cone
slices : Número de subdivisões ao redor do eixo Z (linhas de longitude)
stack : Número de subdivisões ao longo do eixo Z (a linhas de latitude).

Exemplo : glutWireCone(24,40,55,25);

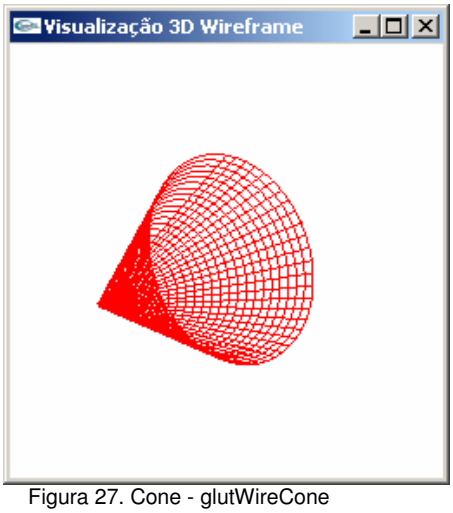


Figura 27. Cone - glutWireCone

### 6.2.4 - Toroide

void **glutSolidTorus**(GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)
void **glutWireTorus**(GLdouble innerRadius,GLdouble outerRadius,Glint nsides,Glint rings)

Parâmetros :

innerRadius:Raio interno do toróide.
outerRadius: Raio externo do toróide
nsides: Número de lados para cada seção radial.
rings: Número de subdivisões radiais do toróide.

Exemplo : glutWireTorus(15,25,30,35);

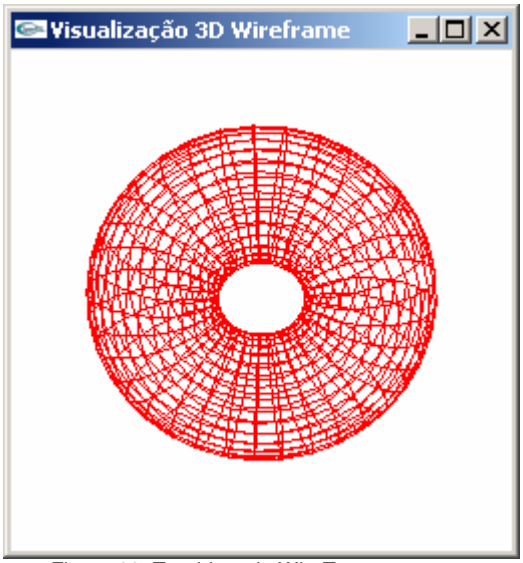



Figura 28. Toroide - glutWireTorus

## 6.2.5 - Dodecaedro

void **glutSolidDodecahedron** ()
void **glutWireDodecahedron** ()

Exemplo : glutWireDecahedron ();

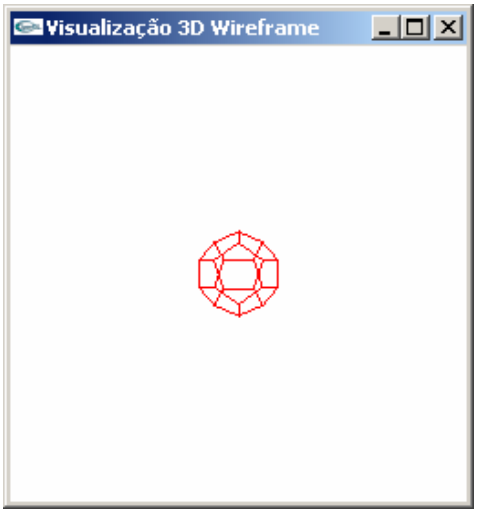



Figura 29. Dodecaedro – glutWireDecahedron

## 6.2.6 - Octaedro

void **glutSolidOctahedron**()
void **glutWireOctahedron**()

Exemplo : glutWireOctahedron()

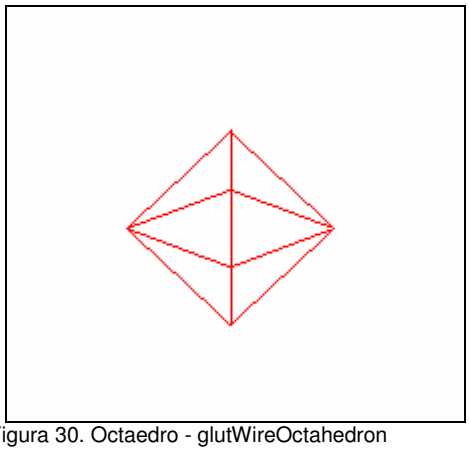


Figura 30. Octaedro - glutWireOctahedron

## 6.2.7 - Tetraedro

void **glutSolidTetrahedron**()
void **glutWireTetrahedron**()

Exemplo : glutWireTetrahedron()

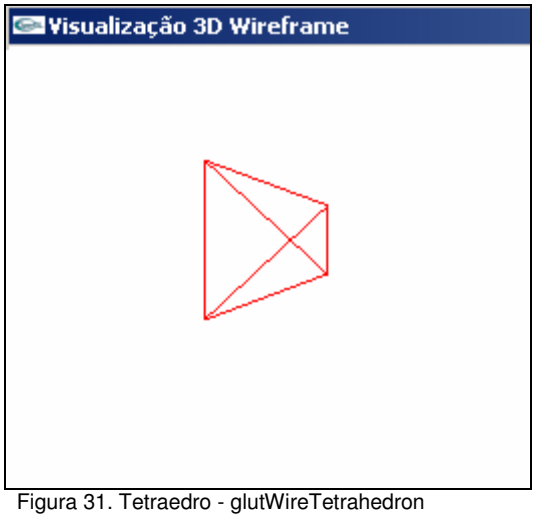



Figura 31. Tetraedro - glutWireTetrahedron

## 6.2.8 - Icosaedro

void **glutSolidIcosahedron**(),
void **glutWireIcosahedron**()

Exemplo : glutWireIcosahedron();

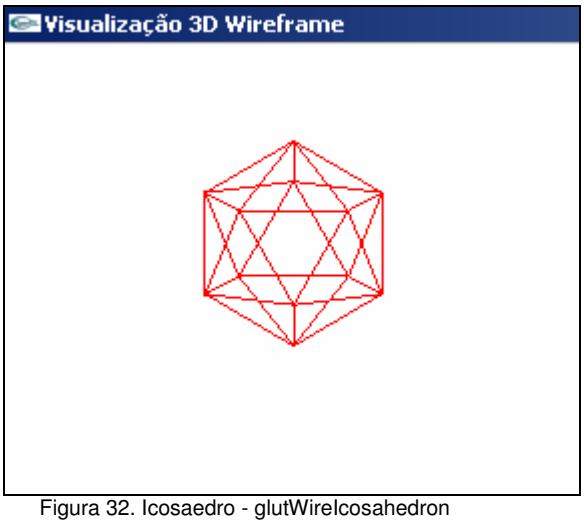



Figura 32. Icosaedro - glutWireIcosahedron

### 6.2.9 - Teapot

void **glutSolidTeapot** (GLdouble size);
void **glutWireTeapot** (GLdouble size);

Parâmetro :

size : Tamanho do “teapot”.

Exemplo : glutWireTeapot(50.0);

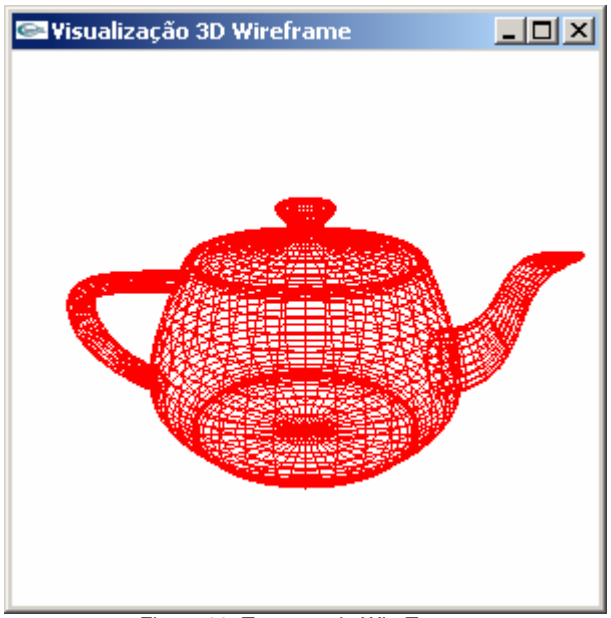


Figura 33. Teapot - glutWireTeapot

## 6.3 - Exemplo de Formas 3D, Transformações e Animação no OpenGL

```
/* Exemplo6.c - Marcionílio Barbosa Sobrinho
* Programa que simula a rotação de um planeta em torno de um astro maior
*          Trata os conceitos de transformações além de apresentar
*          animação através do OpenGL
* Referência do Código: OpenGL Programming Guide - RedBook
*                       planet.c, double.c
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
static int year = 0, day = 0, wire = 0;

/* Define o modelo de cores a ser utilizado
   GL_FLAT : a cor não varia na primitiva que é desenhada */

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

/*
  Função responsável pelo desenho das esferas.
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado.
*/

void display(void)
{

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);

  /* Verifica se o grau de rotacao do ano é maior que 245
```

```
     esta verificação é necessária para que, quando a esfera menor
     estiver atras da maior a mesma não sobreponha o desenho desta, o else
     deste if inverte a ordem de apresentação das esferas*/
  if (year < 245 )
  {

    /* Armazena a situação atual da pilha de matrizes */
    glPushMatrix();
      glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
      glTranslatef (2.0, 0.0, 0.0);
      glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
      glColor3f (0.0, 0.0, 1.0);

      /* Se a tecla w for pressionada irá fazer o controle
         do tipo de apresentação de sólido ou Wire (aramado) */

      if (wire == 0)
          glutSolidSphere(0.2, 10, 8);
      else
          glutWireSphere(0.2,10, 8);

    /*Descarrega a pilha de matrizes até o último glPushMatrix */
    glPopMatrix();

    glPushMatrix();
      glColor3f (0.89, 0.79, 0.09);

      if (wire == 0)
         glutSolidSphere(1.0, 20, 16);
      else
         glutWireSphere(1.0, 20, 16);

    glPopMatrix();

  }
  else
  {
    glPushMatrix();
      glColor3f (0.89, 0.79, 0.09);

      if (wire == 0)
         glutSolidSphere(1.0, 20, 16);
      else
         glutWireSphere(1.0, 20, 16);

    glPopMatrix();

    glPushMatrix();
      glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
      glTranslatef (2.0, 0.0, 0.0);
      glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
      glColor3f (0.0, 0.0, 1.0);

      if (wire == 0)
          glutSolidSphere(0.2, 10, 8);
      else
          glutWireSphere(0.2,10, 8);

    glPopMatrix();

  }

  // Executa os comandos
  glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
```

```
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla d, será executada uma rotação no
   próprio eixo da esfera menor. Quando pressionada a tecla y
   a esfera menor irá rotacionar em torno da esfera maior, em uma
   órbida determinada na translação na função display()
   A tecla w é responsável por determinar se as esferas serão sólidas
   ou aramadas (wire)
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 'd':
      case 'D':
           day = (day + 10) % 360;
           glutPostRedisplay();
           break;
      case 'Y':
      case 'y':
           year = (year + 5) % 360;
           glutPostRedisplay();
           break;
      case 'w' :
      case 'W' :
           wire = wire == 1 ? 0 : 1;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
*/
void spinDisplay(void)
{
  year = (year + 1) % 360;
  day  = (day + 2 ) % 360;
  glutPostRedisplay();
}


/*
  Esta função irá controlar os botões do mouse.
  Se pressionado o botão da esquerda ela define
  a função spinDisplay como a função de "idle" do GLUT
  o comando glutIdelFunc, executa uma determinada função quando
  nenhum evento estiver ocorrendo. (pressionamento de botões etc.)
  Quando o botão do meio é pressionado a função de Idle recebe NULL
  desabilitando a animação
*/
void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
```

```
               glutIdleFunc(spinDisplay);
            break;
      case GLUT_MIDDLE_BUTTON:
            if (state == GLUT_DOWN)
               glutIdleFunc(NULL);
            break;
      default:
            break;
   }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 6 - Animação no OpenGL");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

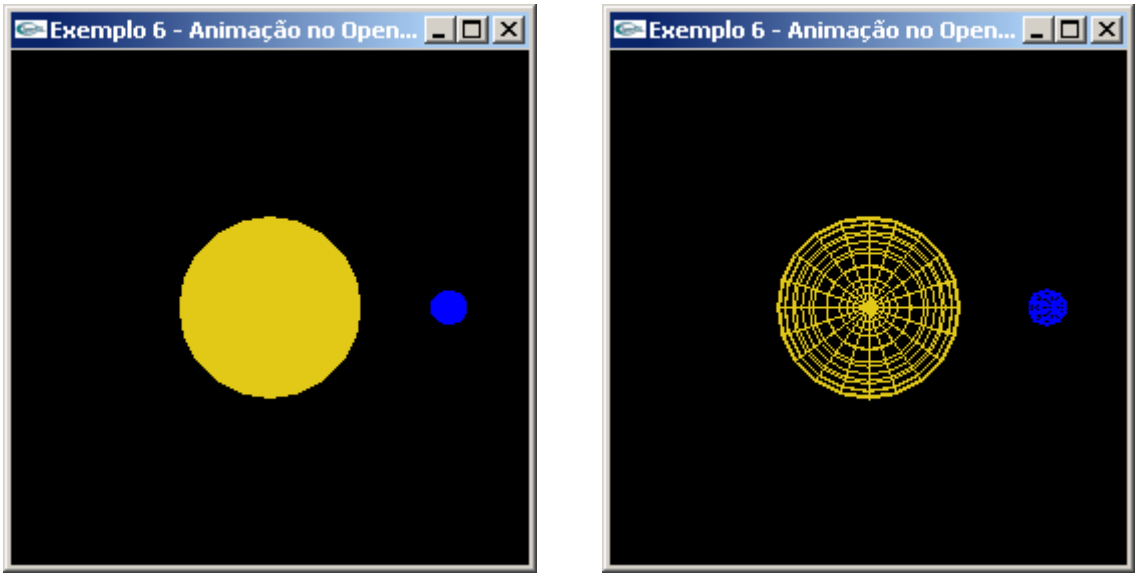



Figura 34. Imagem gerada pelo Exemplo 6 – Animação e Formas 3D

# Capítulo 7 - Modelos de Iluminação

A iluminação é uma técnica importante em objetos gráficos criados através do computador. Sem iluminação os objetos tendem a não apresentar características realistas.

O princípio da iluminação consiste em simular como os objetos refletem as luzes.

OpenGL divide iluminação em três partes:

- propriedades de materiais,
- propriedades de luzes,
- parâmetros globais de iluminação.

A iluminação está disponível em ambos os modos, RGBA e modo de índice de cores. RGBA é mais flexível e menos restritivo que a iluminação no modo de índice de cores. [Dave Shreiner 97]

## 7.1 - Como o OpenGL simula as Luzes

A iluminação no OpenGL é baseada no modelo de iluminação de Phong. A cada vértice da primitiva, uma cor é calculada usando as propriedades dos materiais junto com suas propriedades de luzes. A cor para o vértice é computada pela soma das quatro cores calculadas para a cor final do vértice. As quatro componentes que contribuem para a cor dos vértice são: Iluminação Ambiente,Componente Difusa, Luz Especular e Emissão.

Iluminação Ambiente : é uma luz que se difundiu tanto pelo ambiente que sua direção é impossível determinar - parece vir de todas as direções.

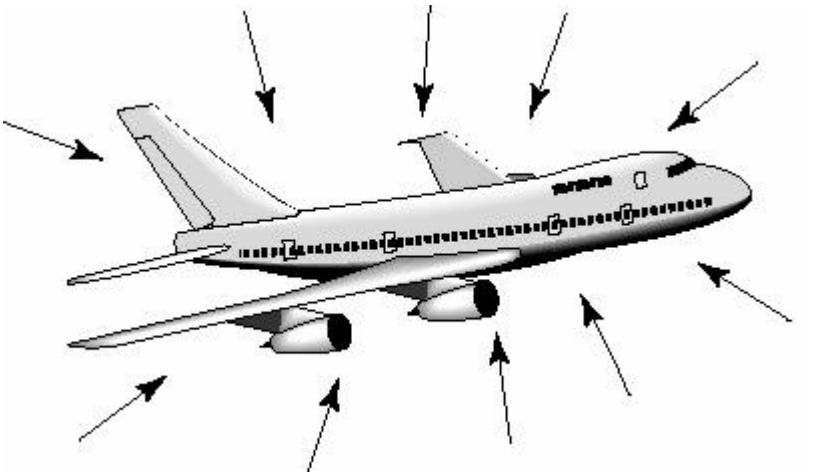


Figura 35. Iluminação ambiente

A componente difusa é a luz que vem de uma direção. Uma vez que esta luz encontre uma superfície, porém, é difundida igualmente em todas as direções, assim a superfície aparece igualmente luminosa, não importando de onde é visualizada. Qualquer luz vinda de uma posição particular ou direção tem provavelmente uma componente difusa.

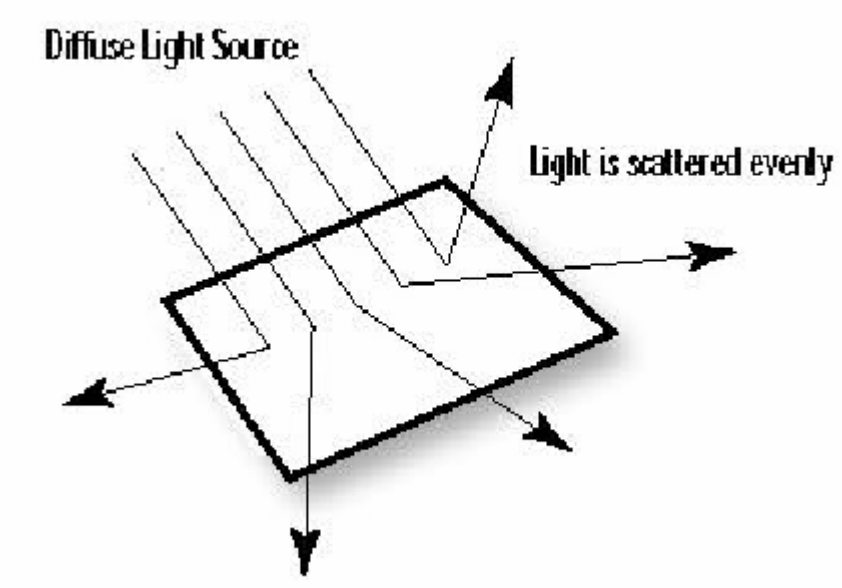



Figura 36. Fonte de luz difusa

Especular: luz que vem de uma direção e tende a ser refletida numa única direção;

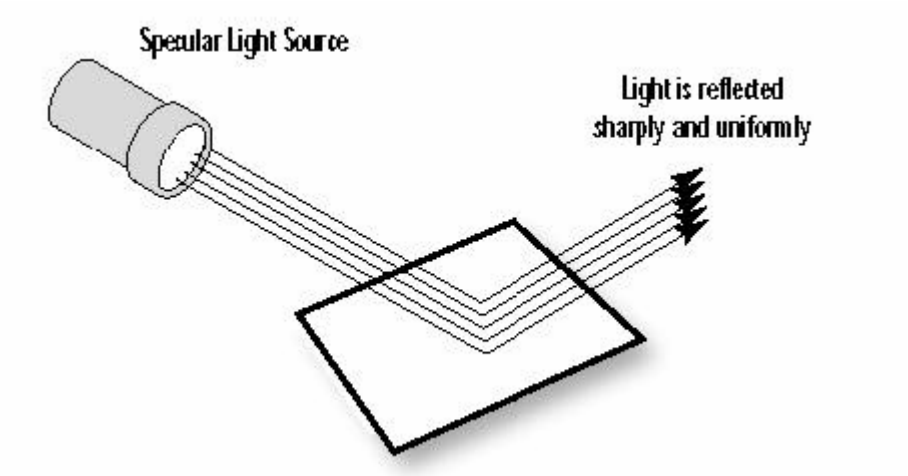



Figura 37. Luz especular

Emissiva: simula a luz que se origina de um objeto; a cor emissiva de uma superfície adiciona intensidade ao objeto, mas não é afetada por qualquer fonte de luz; ela também não introduz luz adicional da cena.

## 7.2 - Cores de Materiais

O modelo de iluminação do OpenGL faz a aproximação da cor de um material dependendo dos percentuais de luz vermelha, verde, e azul entrantes. Por exemplo, uma bola perfeitamente vermelha reflete toda a luz vermelha entrante e absorve toda a luz verde e azul que incide sobre ela.

Como luzes, materiais têm diferentes cores: ambiente, difusa e especular que determina a ambiente, difusa e especular e refletâncias de material. A refletância ambiente de um material é combinada com a componente ambiente de cada fonte de luz entrante, a refletância com a componente de luz difusa, e semelhantemente para o refletâncias da componente especular. As refletâncias ambiente e difusa definem a cor do material. Refletância especular normalmente é branca ou fica aparentemente acinzentada, de forma. Exemplo : se uma luz branca que reflete em uma esfera de plástico vermelha brilhante, a maioria da esfera aparece vermelha, porém, o destaque brilhante é branco.

## 7.3 - Adicionando luzes a uma cena

Para adicionar luz a uma cena são necessários os seguintes passos :.

1 – Definir os vetores normais para cada vértice de todos os objetos. Estes vetores normais determinam a orientação do objeto relativo às fontes de luz claras.

2 – Criar, selecionar, e posicionar uma ou mais fontes de luzes.

3 – Criar e selecionar m modelo de iluminação que define o nível de luz ambiente global e a localização efetiva do ponto de visão (para os cálculos de iluminação).

4 – Definir as propriedades de materiais para os objetos na cena

### 7.3.1 - Definição dos vetores normais

A normal de iluminação diz para o OpenGL como o objeto reflete luz ao redor de um vértice. Imaginando que exista um espelho pequeno no vértice, a normal descreve como o espelho é orientado, e por conseguinte como a luz é refletida. glNormal * () configura a normal atual que é usada no cálculo de iluminação para todos os vértices até uma nova normal ser configurada. O comando glScale * () afeta normal e também os vértices o que pode mudar o tamanho da normal, desnormalizando. OpenGL pode automaticamente normalizar a normal, através da habilitação pelo comando glEnable(GL_NORMALIZE). ou glEnable(GL_RESCALE_NORMAL). GL_RESCALE_NORMAL é um modo especial para a normal ser escalada uniformemente. Se não, deve ser utilizado GL_NORMALIZE o qual controla todas as situações de normalização, mas requer o cálculo de raízes quadradas, o que pode potencialmente afetar o desempenho.

### 7.3.2 - Criação, seleção e posicionamento de luzes

A chamada da função glLight () é usada para fixar os parâmetros para uma luz. O OpenGL permite a implementação de até oito luzes que são nomeadas

como GL_LIGHT0 até GL_LIGHT*n* onde *n* é um menos o número máximo suportado.

Luzes em OpenGL têm várias características as quais podem ser modificadas. Propriedades de cores permitem interações separadas com as diferentes propriedades dos materiais. Propriedades de posicionamento controlam a localização e o tipo da luz e a atenuação controla a tendência natural da luz deteriorar-se sobre a distância.

Especificação do comando :

void **glLight**{if}(GLenum light, GLenum pname, TYPEparam);
void **glLight**{if}v(GLenum light, GLenum pname, TYPE *param);

Cria uma luz especificada por LIGHT que pode ser GL_LIGHT0 GL_LIGHT1,..., ou GL_LIGHT7. A característica da luz que está sendo configurada é definida por *pname* que especifica um parâmetro nomeado conforme tabela abaixo. *param* indica os valores para os quais a característica de *pname* é fixada; é um ponteiro para um grupo de valores se a versão de vetor é usada, ou o próprio valor se a versão não vetorial for usada.

| Nome do Parâmetro | Valor Padrao | Significado |
|---|---|---|
| GL_AMBIENT | (0.0, 0.0, 0.0, 1.0) | Intensidade RGBA da luz ambiente |
| GL_DIFFUSE | (1.0, 1.0, 1.0, 1.0) | Intensidade RGBA da luz difusa |
| GL_SPECULAR | (1.0, 1.0, 1.0, 1.0) | Intensidade RGBA da luz especular |
| GL_POSITION | (0.0, 0.0, 1.0, 0.0) | Posição da luz<br><br>(x, y, z, w) |
| GL_SPOT_DIRECTION | (0.0, 0.0, -1.0) | Direção da luz spotlight (x, y, z) |
| GL_SPOT_EXPONENT | 0.0 | Expoente spotlight |

| | | |
|---|---|---|
| GL_SPOT_CUTOFF | 180.0 | Ângulo do spotlight |
| GL_CONSTANT_ATTENUATION | 1.0 | Fator de constante de atenuação |
| GL_LINEAR_ATTENUATION | 1.0 | Fator de atenuação linear |
| GL_QUADRATIC_ATTENUATION | 0.0 | Fator de atenuação quadrática |

Tabela 5. Característica da luz para a função glLightfv

Exemplo :

```
GLfloat luz_ambiente[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat luz_difusa[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat luz_epecular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat luz_posicao[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_AMBIENT, luz_ambiente);
glLightfv(GL_LIGHT0, GL_DIFFUSE, luz_difusa);
glLightfv(GL_LIGHT0, GL_SPECULAR, luz_epecular);
glLightfv(GL_LIGHT0, GL_POSITION, luz_posicao);
```

## 7.3.3 - Exemplo de criação e posicionamento de luzes

```
/*
  Programa : movelight.c
  Exemplo integralmente retirado do livro : OpenGL Programming Guide - Red Book
  Descrição : Apresenta um toroide ilumindado. A posição da luz pode ser modificada bastando
              para tal o botão esquerdo do mouse ser pressionado.
  Objetivo : Apresentação de criação e posicionamento de luzes.

*/

#include <windows.h>
#include <GL/glu.h>
#include <GL/glut.h>

static int spin = 0;
void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_SMOOTH);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}
/* Here is where the light position is reset after the modeling
* transformation (glRotated) is called. This places the
* light at a new position in world coordinates. The cube
* represents the position of the light.
*/
```

```
void display(void)
{
   GLfloat position[] = { 0.0, 0.0, 1.5, 1.0 };
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glPushMatrix ();
     glTranslatef (0.0, 0.0, -5.0);
     glPushMatrix ();
     glRotated ((GLdouble) spin, 1.0, 0.0, 0.0);
     glLightfv (GL_LIGHT0, GL_POSITION, position);
     glTranslated (0.0, 0.0, 1.5);
     glDisable (GL_LIGHTING);
     glColor3f (0.0, 1.0, 1.0);
     glutWireCube (0.1);
     glEnable (GL_LIGHTING);
   glPopMatrix ();
   glutSolidTorus (0.275, 0.85, 8, 15);
   glPopMatrix ();
   glFlush ();
}
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void mouse(int button, int state, int x, int y)
{
  switch (button) {
       case GLUT_LEFT_BUTTON:
            if (state == GLUT_DOWN) {
               spin = (spin + 30) % 360;
               glutPostRedisplay();
            }
            break;
       default:
           break;
  }
}

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow (“Movimentação de luzes”);
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutMainLoop();
  return 0;
}
```

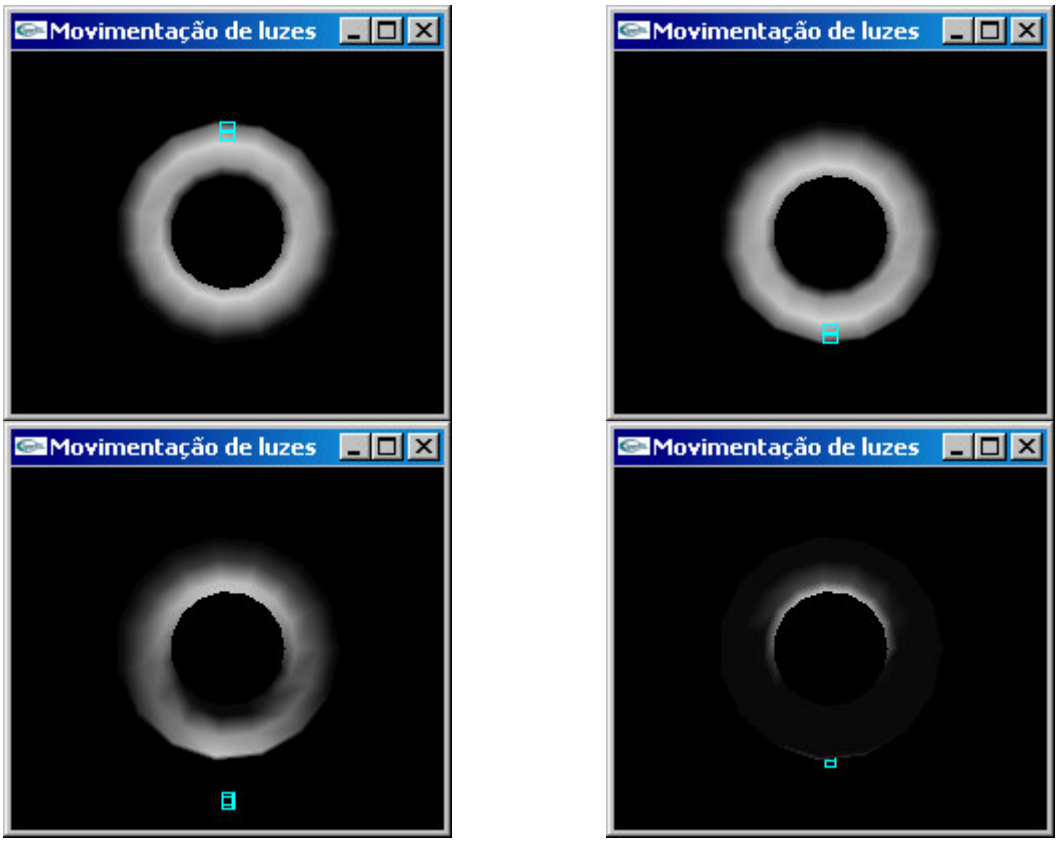



Figura 38. Movimentação de luzes

### 7.3.4 - Criação e seleção de modelo de iluminação

Propriedades as quais não são diretamente conectadas com materiais ou luzes são agrupadas dentro de modelos de propriedades. Existem três tipos de propriedades associadas com modelos de iluminação :

1 – Iluminação de dois lados usam a frente e as costas das propriedades do material para iluminação de uma primitiva.

2 - Cor ambiente global, que inicializa a contribuição global do ambiente na equação de iluminação.

3 – Modo de visualização local, que desabilita a otimização mas que prove mais rapidamente os cálculos de iluminação.

O comando usado para especificar todas as propriedades do modelo de iluminação é glLightModel * (). glLightModel * () tem dois argumentos: a propriedade do modelo de iluminação e o valor desejado para aquela propriedade.

*void **glLightModel**{if}(GLenum pname, TYPEparam);*
*void **glLightModel**{if}**v**(GLenum pname, TYPE *param);*

Especifica as propriedades do modelo de iluminação . A característica do modelo de iluminação a ser espeficado é definido por *pname* que especifica um parâmetro nomeado na tabela abaixo. *param* indica os valores para o qual a característica de pname é definida; é um ponteiro a um grupo de valores se a versão de vetor for usada, ou o próprio valor se a versão não vetorial for utilizada. A versão não vetorial pode ser usada para fixar uma única característica do modelo de iluminação, não para GL_LIGHT_MODEL_AMBIENT.

| Nome do Parâmetro | Valor Padrão | Significado |
|---|---|---|
| GL_LIGHT_MODEL_AMBIENT | (0.2, 0.2, 0.2, 1.0) | Intensidade RGBA ambiente de toda a cena |
| GL_LIGHT_MODEL_LOCAL_VIEWER | 0.0 or GL_FALSE | Como o angulo de reflexão especular e calculado |
| GL_LIGHT_MODEL_TWO_SIDE | 0.0 or GL_FALSE | Define entre a iluminação de um ou dois lados. |

Tabela 6. Valores para o modelo de iluminação - glLightModel

## 7.3.5 - Propriedades de materiais

Propriedades de materiais descrevem a cor e propriedades de superfície de um material . OpenGL suporta propriedades de materiais para frente e costas de objetos.[ Dave Shreiner 99 ]

As propriedades definidas pelo OpenGL são:

• GL_DIFFUSE - cor básica de objeto

• GL_SPECULAR - cor de destaque do objeto

• GL_AMBIENT - cor do objeto quando não diretamente iluminado

• GL_EMISSION - cor emitida pelo objeto

• GL_SHININESS - concentração de brilhos em objetos. Valores variam de 0 (superfície muito áspera - nenhum destaque) até 128 (muito brilhante)

Podem ser fixadas propriedades materiais separadamente para cada face especificando qualquer uma destas GL_FRONT ou GL_BACK, ou para ambas as faces simultaneamente por GL_FRONT_AND_BACK. .[ Dave Shreiner 99 ]

Comando :

*void **glMaterial**{if}(GLenum face, GLenum pname, TYPEparam);*
*void **glMaterial**{if}**v**(GLenum face, GLenum pname, TYPE *param);*

| Nome do Parâmetro | Valor Padrão | Significado |
|---|---|---|
| GL_AMBIENT | (0.2, 0.2, 0.2, 1.0) | Cor ambiente do material |
| GL_DIFFUSE | (0.8, 0.8, 0.8, 1.0) | Cor difusa do material |
| GL_AMBIENT_AND_DIFFUSE | | Cores ambiente e difusa do material. |
| GL_SPECULAR | (0.0, 0.0, 0.0, 1.0) | Cor especular do material |
| GL_SHININESS | 0.0 | Expoente especular |
| GL_EMISSION | (0.0, 0.0, 0.0, 1.0) | Cor emissiva do material |
| GL_COLOR_INDEXES | (0,1,1) | Índice de cores, ambiente, difusa e especular |

Tabela 7. Propriedades possíveis para os materiais – glMaterial*

## 7.3.6 - Exemplo de propriedades dos materiais

```
/* Exemplo7.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta uma variação do Exemplo6.c, com a funcionalidade
*          de luzes na cena, além de apresentar as características de materiais
*         A posição da Luz pode ser modicada através do botão direito do mouse.
* Referência do Código: OpenGL Programming Guide - RedBook
*                       planet.c, movelight.c, material.c
*/

#include <windows.h>
```

```
#include <GL/gl.h>
#include <GL/glut.h>
static int year = 0, day = 0, wire = 0;
int ligacor = 0;
int posicaoluz = 0;


void init(void)
{

   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);

}

/*
  Função responsável pelo desenho das esferas.
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado.
*/

void display(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

  /* Posição da luz */
  GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);

  /* Armazena o estado anterior para
     rotação da posição da luz */

  glPushMatrix () ;

    glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
    glLightfv (GL_LIGHT0, GL_POSITION, posicao);

  glPopMatrix(); // Posição da Luz

    /* Armazena a situação atual da pilha de matrizes */
    glPushMatrix();
      glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
      glTranslatef (2.0, 0.0, 0.0);
```

```
      glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
      glColor3f (0.0, 0.0, 1.0);


      /* Define a propriedade do material */
      //refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, especular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,20);

      /* Se a tecla w for pressionada irá fazer o controle
         do tipo de apresentação de sólido ou Wire (aramado) */

      if (wire == 0)
          glutSolidSphere(0.2, 20, 18);
      else
          glutWireSphere(0.2,10, 8);

    /*Descarrega a pilha de matrizes até o último glPushMatrix */
    glPopMatrix();

    glPushMatrix();
      glColor3f (0.89, 0.79, 0.09);

      /* Define a propriedade do material */
      //refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,100);


      if (wire == 0)
         glutSolidSphere(1.0, 30, 26);
      else
         glutWireSphere(1.0, 20, 16);

    glPopMatrix();



  // Executa os comandos
  glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla d, será executada uma rotação no
   próprio eixo da esfera menor. Quando pressionada a tecla y
   a esfera menor irá rotacionar em torno da esfera maior, em uma
   órbida determinada na translação na função display()
   A tecla w é responsável por determinar se as esferas serão sólidas
   ou aramadas (wire)
```

```
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 'd':
      case 'D':
           day = (day + 10) % 360;
           glutPostRedisplay();
           break;
      case 'Y':
      case 'y':
           year = (year + 5) % 360;
           glutPostRedisplay();
           break;
      case 'w' :
      case 'W' :
           wire = wire == 1 ? 0 : 1;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
*/
void spinDisplay(void)
{
  year = (year + 1) % 360;
  day  = (day + 2 ) % 360;
  glutPostRedisplay();
}


/*
  Esta função irá controlar os botões do mouse.
  Se pressionado o botão da esquerda ela define
  a função spinDisplay como a função de "idle" do GLUT
  o comando glutIdelFunc, executa uma determinada função quando
  nenhum evento estiver ocorrendo. (pressionamento de botões etc.)
  Quando o botão do meio é pressionado a função de Idle recebe NULL
  desabilitando a animação
*/
void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(spinDisplay);
           break;
      case GLUT_MIDDLE_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(NULL);
           break;
      case GLUT_RIGHT_BUTTON:
           posicaoluz = (posicaoluz + 1) % 360;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
```

```
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 7 - Propriedades de Materiais");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```


Figura 39. Exemplo 7 – Propriedades de Materiais

# Capítulo 8 - Modelagem hierárquica

A modelagem hierarquica consisite na tecnica de criar e tratar um grupo de objetos como se fossem um único objeto

Na modelagem hierárquica um objeto é descrito inicialmente em termos de coordenadas locais e as transformações de modelagem geram posicionamento do objeto em relação aos outros objetos.

O OpenGL permite dois mecanismos para tratamento de modelagem hierárquica:

*Pilhas de matrizes*

Quando um novo componente geométrico é adicionado a uma estrutura, sua matriz de transformação é empilhada na estrutura

*Display lists (Listas de Visualização)*

Permitem encapsular os atributos de cada componente.

## 8.1 - Pilhas de Matrizes

Através dos comandos glPushMatrix() e glPopMatrix a pilha de matrizes é controlada. Deste modo as transformações podem ser agrupadas nas respectivas matrizes e através deste agrupamento uma construção hierárquica pode ser obtida, conseguindo assim um efeito desejado na montagem de objetos complexos, que passam a ter tratamento como um objeto único e simples.

Comando :

***void** glPushMatrix(**void**);*
*Faz o empilhamento do estado atual do objeto na pilha. A pilha corrente é determinada pelo comando **glMatrixMode**().*

***void*** *glPopMatrix(**void**);*

*Desempilha o topo da pilha, destruindo o conteúdo retirado da matriz.A pilha corrente é determinada por* ***glMatrixMode****().*

Exemplo :

```
 /*
  Trecho de código do programa exemplo : Exemplo6.c
 */
glPushMatrix();
  glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
  glTranslatef (2.0, 0.0, 0.0);
  glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
  glColor3f (0.0, 0.0, 1.0);


  /* Define a propriedade do material */
  //refletância do material
  glMaterialfv(GL_FRONT,GL_SPECULAR, especular);
    // Define a concentração do brilho
  glMateriali(GL_FRONT,GL_SHININESS,20);

  /* Se a tecla w for pressionada irá fazer o controle
     do tipo de apresentação de sólido ou Wire (aramado) */

  if (wire == 0)
      glutSolidSphere(0.2, 20, 18);
  else
      glutWireSphere(0.2,10, 8);

/*Descarrega a pilha de matrizes até o último glPushMatrix */
glPopMatrix();
```

## 8.2 - Display Lists (Listas de Exibição)

As listas de exibição aperfeiçoam o desempenho de aplicações OpenGL. Porém uma vez que uma lista de exibição é criada, não pode ser modificada.

O modo no qual são otimizados os comandos em uma lista de exibição pode variar de implementação para implementação. Por exemplo, um comando tão simples quanto glRotate * () poderia mostrar uma significante melhoria de performance se estivesse em uma lista de exibição, uma vez que os cálculos para produzir a matriz de rotação não são triviais.

As listas de exibição são indicadas para: Operações complexas com matrizes, luzes, propriedades de materiais e modelos de iluminação complexos, texturas.

Comandos :

***GLuint*** *glGenLists(**GLsizei** range);*

*Este comando aloca um conjunto de números contiguos, previamente não alocados por um índice da lista de exibição. O valor inteiro retornado é o índice que marca o início de um bloco contíguo de um display de visualização*
*Os índices retornados são marcados como vazios e usados então por chamadas subsequentes do comando glGenLists().*

***void*** *glNewList (**GLuint** lista, **GLenum** modo);*

*Especifica o início de uma lista de exibição. As rotinas OpenGL chamadas subseqüentemente ( enquanto o comando glEndList() não for executado) são armazenados na lista de visualização, exceto uma lista restrita de rotinas OpenGL que não podem ser armazenados na lista . O parâmetro lista é um inteiro positivo maior que zero, que identifica unicamente a lista de visualização . Os possíveis valores para modo são GL_COMPILE e GL_COMPILE_AND_EXECUTE. O parâmetro GL_COMPILE deve ser usado se deseja-se que os comandos da lista de exibição nao sejam executados imediatamente após serem colocados na lista, neste caso deve ser utilizado o parâmetro GL_COMPILE_AND_EXECUTE.*

***void*** *glEndList (**void**);*

*Marca o final de uma lista de exibição.*

Após a lista ter sido criada a mesma poderá ser executada através do comando glCallList().

***void*** *glCallList **(GLuint** list**);***

*Esta rotina executa a lista de exibição especificada pelo parâmetro. Os comandos na lista de execução são então executados na ordem que foram salvos.*

Comandos que não podem ser armazenados em uma lista de exibição:

glColorPointer(), glFlush(), glNormalPointer(),glDeleteLists(), glGenLists(), glPixelStore(),glDisableClientState(), glGet*(), glReadPixels(),glEdgeFlagPointer(), glIndexPointer(), glRenderMode(), glEnableClientState(), glInterleavedArrays(), glSelectBuffer(),glFeedbackBuffer(), glIsEnabled(), glTexCoordPointer(),glFinish(), glIsList(), glVertexPointer().

## 8.3 - Listas de visualização hierárquicas

Listas de visualização hierárquicas são listas executadas por outras listas de exebição através do comando glCallList chamando () entre um par glNewList () e glEndList () . Uma lista de exibição hierárquica é útil para criação de

componentes de objetos especialmente se alguns desses componentes são usados várias vezes no ciclo de construção.

Exemplo :

```
glNewList(listIndex,GL_COMPILE);
        glCallList(handlebars);
        glCallList(frame);
        glTranslatef(1.0,0.0,0.0);
        glCallList(wheel);
        glTranslatef(3.0,0.0,0.0);
        glCallList(wheel);
glEndList();
```

## 8.4 - Exemplo de Lista de visualização

```
/* Exemplo8.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta uma variação do Exemplo7.c, com a funcionalidade
*          de utilização de Display Lists
* Referência do Código: OpenGL Programming Guide - RedBook
*                       planet.c, movelight.c, material.c
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

int ligacor = 0;
int posicaoluz = 0;
GLuint lista1, lista2;


void init(void)
{

   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);

   //Chama a funcao de criação dos Display Lists
   crialista();
}
```

```
/*
  Função responsável pelo desenho das esferas.
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado dentro das "Display Lists" criadas.
*/

void crialista(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

  /* Posição da luz */
  GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  lista1 = glGenLists (1);
  lista2 = glGenLists (2);

glNewList (lista2, GL_COMPILE);

      glColor3f (0.89, 0.79, 0.09);

      glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,100);

     glPushMatrix();
      // glutSolidSphere(1.0, 30, 26);
      glRotatef (0.0, 0.0, 1.0, 0.0);
      glTranslatef (0, 0.0, 0.0);
      glRotatef (23, 1.0, 0, 0.0);

      glutSolidCube (2.0);
     glPopMatrix();

glEndList();



glNewList (lista1, GL_COMPILE);
  /* Armazena o estado anterior para
     rotação da posição da luz */
  glColor3f (1.0, 1.0, 1.0);

  glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
  glLightfv (GL_LIGHT0, GL_POSITION, posicao);


  glCallList(lista2);

  glPushMatrix();
      glRotatef (0, 1.0, 0.0, 0.0);
      glTranslatef (0.0, 2.0, 0.0);
      glRotatef (0, 0.0, 1.0, 0.0);
      glColor3f (0.0, 0.0, 1.0);

      /* Define a propriedade do material */
      //refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, especular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,20);

       glutSolidTorus(0.1,0.2,10,20);

    glPopMatrix();


glEndList ();
```

```
}


void display (void )
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // Executa os comandos
  glCallList (lista1);
  glTranslatef (3.0, 2.0, -7.0);
  glCallList(lista1);
  glutSwapBuffers();
}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla ESC o programa será encerrado.
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 27 :
           exit(0);
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 8 - Listas de Visualização");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

Figura 40. Exemplo 8 – Listas de visualização

# Capítulo 9 Texturas

O mapeamento de texturas é a aplicação de imagens em uma superfície, conforme se procede na aplicação de decalques em um brinquedo.

Quando uma imagem é mapeada sobre um objeto, a cor de cada pixel do objeto é modificada por uma cor correspondente da imagem. A imagem normalmente é armazenada em uma matriz como uma imagem contínua deve ser reconstruída a partir desta matriz. Logo a imagem deve ser deformada para adequar-se a qualquer distorção (causada, talvez, pela perspectiva ) no objeto a ter a imagem projetada. Então esta imagem deformada é filtrada para remover componentes de alta freqüência que geram o efeito de "aliasing" no passo final.
Este mapeamento aumenta as cores especificadas para uma primitiva geométrica com as cores armazenadas na imagem. Uma imagem pode conter conjuntos de cores 1D, 2D ou 3D que são chamdas "texels" [Angel 1999].
Alguns dos muitos usos de mapeamentos de texturas incluem :

- Simulação de materiais como madeira, tijolos ou granito
- Redução da complexidade (número de polígonos ) de um objeto geométrico
- Técnicas de processamento de imagens como deformação e retificação, rotação e escala
- Simulação de superfícies refletivas como espelhos ou chão polido.

## 9.1 - Aplicação de Texturas no OpenGL

Para habilitar o mapeamento de texturas no OpenGL deve-se seguir os seguintes passos :

1) Especificar texturas no objeto de textura
2) Especificar o filtro de textura
3) Especificar a função de textura
4) Especificar o modo de deformação da textura
5) Especificar opcionalmente a correção de perspectiva
6) Associar o objeto à textura

7) Habilitar a textura
8) Fornecer as coordenadas de textura para o vértice

Como em qualquer outra função OpenGL, o mapeamento de textura requer o comando glEnable (), para habilitação da mesma. Os parâmetros possíveis para habilitação de textura são :

`GL_TEXTURE_1D` – textura unidimensional;
`GL_TEXTURE_2D` - textura bidimensional;
`GL_TEXTURE_3D` - textura tridimensional.

As texturas 2D são as normalmente usadas. Texturas 1D são úteis para aplicação de contornos em objetos (como contornos de altitude para montanhas). Texturas 3D são úteis para para "rendering" de volume.

A forma de armazenamento dos pixels na textura deve ser definida. Para tal o comando glPixelStorei (GLenum pname, Glint param) ou glPixelStoref(GLenum pname, GLfloat param) deve ser utilizado.

O parâmetro *pname* irá indicar o nome do parâmetro simbólico a ser habilitado pela função (vide tabela abaixo) enquanto o parâmetro param armazena o valor associado ao parâmetro pname

| panme | Tipo | Valor Inicial | Limite válido |
|---|---|---|---|
| GL_PACK_SWAP_BYTES | Boleano | false | true or false |
| GL_PACK_LSB_FIRST | Boleano | false | true or false |
| GL_PACK_ROW_LENGTH | inteiro | 0 | [0,∞) |
| GL_PACK_SKIP_ROWS | Inteiro | 0 | [0,∞) |
| GL_PACK_SKIP_PIXELS | Inteiro | 0 | [0,∞) |
| GL_PACK_ALIGNMENT | Inteiro | 4 | 1, 2, 4, or 8 |
| GL_PACK_IMAGE_HEIGHT | Inteiro | 0 | [0,∞) |
| GL_PACK_SKIP_IMAGES | Inteiro | 0 | [0,∞) |
| GL_UNPACK_SWAP_BYTES | Boleano | false | true or false |
| GL_UNPACK_LSB_FIRST | Boleano | false | true or false |
| GL_UNPACK_ROW_LENGTH | Inteiro | 0 | [0,∞) |
| GL_UNPACK_SKIP_ROWS | Inteiro | 0 | [0,∞) |
| GL_UNPACK_SKIP_PIXELS | Inteiro | 0 | [0,∞) |
| GL_UNPACK_ALIGNMENT | Inteiro | 4 | 1, 2, 4, or 8 |
| GL_UNPACK_IMAGE_HEIGHT | Inteiro | 0 | [0,∞) |

| | | | |
|---|---|---|---|
| GL_UNPACK_SKIP_IMAGES | Inteiro | 0 | [0,∞) |

Tabela 8. Modo de armazenamento de pixels no OpenGL - glPixelStorei

## 9.1.1 - Especificação de textura

O comando glTexImage2D() define uma textura bi-dimensional. Ele utiliza-se de vários argumentos, os quais são descritos abaixo.

Comando:

*void **glTexImage2D**(GLenum target, GLint level, GLint internalFormat,*
*GLsizei width, GLsizei height, GLint border,*
*GLenum format, GLenum type,*
*const GLvoid *pixels);*

*Define uma textura bi-dimensional. O parâmetro target pode conter um dos dois valores : GL_TEXTURE_2D ou GL_PROXY_TEXTURE_2D. Se várias resoluções de texuras forem utilizadas então o parâmetro level deverá conter o valor apropriado. Caso apenas uma resolução seja utilizada o valor do mesmo deverá ser 0*

*O parâmetro, internalFormat, indica quais valores para componentes RGBA ou luminosidade ou intensidade é selecionado para utilização na descrição dos texels de uma imagem. O valor de internalFormat é um inteiro que varia de 1até 4, ou uma das 38 constantes simbólicas: GL_ALPHA, GL_ALPHA4, GL_ALPHA8, GL_ALPHA12, GL_ALPHA16, GL_LUMINANCE,GL_LUMINANCE4, GL_LUMINANCE8, GL_LUMINANCE12, GL_LUMINANCE16, GL_LUMINANCE_ALPHA, GL_LUMINANCE4_ALPHA4, GL_LUMINANCE6_ALPHA2,GL_LUMINANCE8_ALPHA8, GL_LUMINANCE12_ALPHA4, GL_LUMINANCE12_ALPHA12,GL_LUMINANCE16_ALPHA16, GL_INTENSITY, GL_INTENSITY4, GL_INTENSITY8, GL_INTENSITY12, GL_INTENSITY16, GL_RGB, GL_R3_G3_B2, GL_RGB4, GL_RGB5, GL_RGB8, GL_RGB10, GL_RGB12, GL_RGB16, GL_RGBA, GL_RGBA2, GL_RGBA4, GL_RGB5_A1, GL_RGBA8, GL_RGB10_A2, GL_RGBA12, e GL_RGBA16.*

*Os parâmetros width e height, consistem na dimensão da imagem de textura; border indica a largura da borda, a qual pode ser zero (sem borda) ou um. Ambos os valores de width e height devem ter a forma 2m + 2b, onde m é um inteiro não negativo ( o qual pode ter um valor diferente tanto para width quanto para height ) e b é o valor da borda. O tamanho máximo de um mapa de textura depende da implementação do OpenGL, mas deve ser no mínimo 64 × 64 (ou 66 × 66 com bordas).*

*Os parâmetros format e type descrevem o formato e o tipo de dados da imagem de textura.*

*O parametro format pode ser: GL_COLOR_INDEX, GL_RGB, GL_RGBA, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_LUMINANCE, ou GL_LUMINANCE_ALPHA*
*Similarmente, o parâmetro type pode ser GL_BYTE, GL_UNSIGNED_BYTE, GL_SHORT, GL_UNSIGNED_SHORT,GL_INT, GL_UNSIGNED_INT, GL_FLOAT, ou GL_BITMAP.*
*Finalmente, pixels contém os dados da imagem de textura. Estes dados descrevem a imagem de textura bem como as suas bordas.*

Mipmaps são um conjunto ordenado de arrays que representam uma mesma imagem em resoluções progressivamente mais baixas: 2a para 1D mipmaps, 2a2b para 2D mipmaps

Se a imagem original não tem dimensões exatas que são potência de 2, gluBuild*DMipmaps() ajuda a escalar a imagem para um valor próximo à potência de 2.

Comando :

*int **gluBuild1DMipmaps**(GLenum target, GLint internalFormat, GLint width,*
*GLenum format, GLenum type, void *data);*
*int **gluBuild2DMipmaps**(GLenum target, GLint internalFormat, GLint*
*width,GLint height, GLenum format, GLenum type,void *data);*

*Constrói uma série de mipmaps e chama **glTexImage*D**() para carregar a imagem. Os parâmetros target, internalFormat, width, height, format, type, e data são exatamente os mesmos para **glTexImage1D()** e **glTexImage2D()**. Um valor 0 é retornado se todos os mipmaps são construídos com sucesso; entretanto, um código de erro GLU é retornado na ocorrência do mesmo.*

*O parâmetro target especifica a textura alvo, deve ser GL_TEXTURE_2D para gluBuild2DMipmaps ou GL_TEXTURE_1D gluBuild1DMipmaps*
O parâmetro internalFormat tem as mesmas características descritas para este parâmetro no comando ***glTexImage2D assim como os parâmetros width e height.***

O parâmetro *format* especifica o formato dos dados de pixels. Deve ser um destes : GL_COLOR_INDEX, GL_DEPTH_COMPONENT, GL_RED, GL_GREEN, GL_BLUE, GL_ALPHA, GL_RGB, GL_RGBA, GL_BGR, GL_BGRA, GL_LUMINANCE, ou GL_LUMINANCE_ALPHA.

Type especifica o tipo de dados para os dados. Deve ser um destes: GL_UNSIGNED_BYTE, GL_BYTE, GL_BITMAP, GL_UNSIGNED_SHORT, GL_SHORT, GL_UNSIGNED_INT, GL_INT, GL_FLOAT, GL_UNSIGNED_BYTE_3_3_2, GL_UNSIGNED_BYTE_2_3_3_REV,

GL_UNSIGNED_SHORT_5_6_5, GL_UNSIGNED_SHORT_5_6_5_REV,
GL_UNSIGNED_SHORT_4_4_4_4, GL_UNSIGNED_SHORT_4_4_4_4_REV,
GL_UNSIGNED_SHORT_5_5_5_1, GL_UNSIGNED_SHORT_1_5_5_5_REV,
GL_UNSIGNED_INT_8_8_8_8, GL_UNSIGNED_INT_8_8_8_8_REV,
GL_UNSIGNED_INT_10_10_10_2, ou GL_UNSIGNED_INT_2_10_10_10_REV.

Finalmente *data* é um ponteiro para a imagem na memória.

## 9.1.2 - Aplicação de Filtros

Os mapas de texturas são quadrados ou retangulares, mas após serem mapeados em um polígono ou uma superfície e transformados em coordenadas da tela, o texel individual de uma textura raramente corresponde ao pixel individual da imagem final da tela. Dependendo das transformações utilizadas e o mapeamento de textura aplicado, um único pixel na tela pode corresponder de qualquer porção de um texel (ampliação) a um grande conjunto de texels ( redução ). Em qualquer caso, não é claro qual valor do texel deveria ser utilizado e como deve ser calculada a sua media ou interpolada. Conseqüentemente o OpenGL permite que sejam especificados quaisquer das várias opções de filtro para determinar estes cálculos. As opções provêem diferentes escolhas entre velocidade e qualidade da imagem. Também é possível especificar um filtro independente do método de ampliação e redução.

Os filtros são aplicados na textura com a utilização do comando :

***glTexParameter****(GLenum target, GLenum pname,<tipo> param).

(<tipo> deve ser um dos valores : GLfloat, GLint, const GLfloat *, const GLint * )

O parâmetro *target* deverá ser um destes : GL_TEXTURE_1D, GL_TEXTURE_2D, o que irá depender do tipo de mapeamento desejado unidimensional ou bidimensional.

Os demais parâmetros devem seguir a tabela abaixo :

| Pname | param |
|---|---|
| GL_TEXTURE_MAG_FILTER | GL_NEAREST ou GL_LINEAR |
| GL_TEXTURE_MIN_FILTER | GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_NEAREST, ou GL_LINEAR_MIPMAP_LINEAR |

Tabela 9. Filtros de texturas

**GL_TEXTURE_MIN_FILTER :**

A função de redução é usada sempre que o pixel do mapa de textura para uma área é maior que um elemento de textura ( texel ).

**GL_TEXTURE_MAG_FILTER**

A função de ampliação é usada quando o pixel do mapa de textura para uma área é menor ou igual a um elemento de textura (texel). Fixa a função de ampliação de textura para GL_NEAREST ou GL_LINEAR.

**GL_NEAREST**

Retorna o valor do elemento de textura que esta mais próximo ( na distancia de Manhatan) do centro do pixel a ser texturizado.

**GL_LINEAR**

Retorna a media dos pesos dos quatro elementos de textura que estão mais proximos ao centro do pixel texturizado.

**GL_NEAREST_MIPMAP_NEAREST**

Escolhe o mipmap (conjunto ordenado de arrays que representam uma mesma imagem em resoluções progressivamente mais baixas: $2^a$ para 1D mipmaps, $2^a2^b$ para 2D mipmaps ) que mais aproxima ao tamanho do pixel a ser aplicado à textura e usa o critério de GL_NEAREST (o elemento de textura mais próximo ao centro do pixel) para produzir o valor de textura.

**GL_LINEAR_MIPMAP_NEAREST**

Escolhe o mipmap que mais se aproxima do tamanho do pixel a ser aplicado à textura e utiliza GL_LINEAR ( o peso médio dos quatro elementos de testura que estão mais próximos ao centro do pixel ) como critério para produzir o valor de textura.

**GL_NEAREST_MIPMAP_LINEAR**

Escolhe dois mipmaps que mais se aproximam ao tamanho do pixel a ser mapeado. Utiliza GL_NEAREST como critério para produzir um valor de cada mipmap. O valor final de textura é o peso médio desses dois valores.

**GL_LINEAR_MIPMAP_LINEAR**

Escolhe dois mipmaps que mais se aproximam ao tamanho do pixel a ser mapeado.Utiliza GL_LINEAR como critério para produzir um valor de cada mipmap. O valor final de textura é o peso médio desses dois valores.

## 9.1.3 - Objetos de Textura

Um objeto de textura armazena dados de textura.Podem ser controladas muitas texturas simultaneamente podendo ser utilizadas a qualquer momento desde que previamente carregadas. A utilização de objetos de texturas normalmente é o modo mais rápido para aplicação de texturas e resulta em grandes ganhos de desempenho porque quase sempre é muito mais rápido ligar (usar novamente) um objeto de textura existente do que recarregar uma imagem de textura utilizando glTexImage*D.

Para usar objetos de textura devem ser seguidos os seguintes passos :

1 ) Gerar os nomes das texturas :

comando :

*void **glGenTextures**(GLsizei n, GLuint *textureNames);*
*Retorna n nomes não utilizados correntemente para os objetos de texturas contidos no array. O nome retornado em textureNames não tem de ser um conjunto contíguo de inteiros. Os nomes em textureNames sao marcados como utilizados, mas eles adquirem o estado e o dimensionamento (1D or 2D) somente em sua primeira habilitação.*
*Zero é um nome de textura reservada e nunca retornan um nome de textura pelo commando **glGenTextures()**.*

**glIsTexture()** determina se o nome da textura está atualmente em uso. Se um nome de textura é retornado pelo comando glGenTextures() mas ele ainda não foi habilitado (através do comando glBindTexture() com este nome ) então glIsTexture retornará GL_FALSE.

Comando :

*GLboolean **glIsTexture**(GLuint textureName);*
*Retorna GL_TRUE se textureName é um nome de textura que está habilitado e não foi subseqüentemente apagado da memoria. Retorna GL_FALSE se textureName é zero ou textureName é um valor que não tem um nome de uma textura existente.*

2) "Bind" (criar) o objeto de textura para os dados de textura, incluindo o array da imagem e as propriedades da textura.

A mesma rotina, glBindTexture(), cria e usa objetos de texturas. Quando o nome de uma textura é inicialmente ligado (usado com glBindTexture), um novo objeto de textura é criado com os valores padrão para a imagem de textura e as propriedades da textura.

Um objeto de textura pode conter uma imagem de textura e associar a imagens mipmap, incluindo dados associados como, largura, altura, largura da borda, formato interno, componentes de resolução, e propriedades de texturas. Propriedades de texturas incluem filtros de redução e ampliação, modos de distorção, cores de bordas e prioridade da textura.

Comando :

*void **glBindTexture**(GLenum target, GLuint textureName);*

*glBindTexture faz três coisas. Quando utilizado com o textureName de um inteiro não sinalizado como zero para a primeira vez de chamada, um novo objeto de textura é criado e atribuído ao nome. Quando utilizado com um objeto de textura previamente criado, o objeto de textura se torna ativo. Quando utilizado textureName com valor zero, OpenGL pára de usar o objeto de textura e retorna para um valor não nomeado padrão para a textura.*

Quando um objeto é inicialmente criado, ele assume o dimensionamento do parâmetro *target*, o qual é ou GL_TEXTURE_1D or GL_TEXTURE_2D. Imediatamente à sua ligação inicial, o estado de objeto de textura é equivalente ao estado padrão GL_TEXTURE_1D ou GL_TEXTURE_2D ( Dependendo de seu dimensionamento) à inicialização do OpenGL.

3) Limpar objetos de texturas

Se o recurso de textura é limitado, limpar as texturas pode ser uma forma de liberar recursos.

Comando :

*void **glDeleteTextures**(GLsizei n, const GLuint *textureNames);*

*Apaga n objetos de textura, nomeados por elementos do array textureNames. Os nomes de texturas liberados agora podem ser reutilizados (por exemplo por **glGenTextures()**).*
*Se uma textura que está sendo utilizada for apagada, o comando bind irá reverter o valor padrão da textura. Como se glBindTexture fosse chamado com o valor zero para o parâmetro textureName.*

## 9.1.4 - Funções de Texturas

Os valores de cores dos mapas de texturas podem ser usados diretamente como cores a serem desenhadas em uma superfície. No OpenGL também pode-se utilizar os valores dos mapas de texturas para modular a cor que a superfície deverá ter sem a aplicação de textura, ou para transparências de cores no mapa de textura com a cor original da superfície.

Comando :

*void **glTexEnv**{if}(GLenum target, GLenum pname, TYPEparam);*
*void **glTexEnv**{if}v(GLenum target, GLenum pname, TYPE *param);*

*Especifica a função corrente de textura. O parâmetro target deve ser GL_TEXTURE_ENV. Se pname é GL_TEXTURE_ENV_MODE, param pode ser GL_DECAL, GL_REPLACE, GL_MODULATE, ou GL_BLEND, para especificar como os valores de texturas serão combinados com os valores de cores do fragmento processado. Se pname é GL_TEXTURE_ENV_COLOR, param é um array de quatro valores de pontos flutuantes representando R, G, B, e A. Estes valores são usados somente se a função de textura GL_BLEND foi especificada.*

A combinação de função de texturas e o formato da base interna determinam como a textura será aplicada para cada componente de textura. A função de textura opera nas componentes de seleção de texturas e dos valores de cores que serão utilizados sem textura.

A tabela abaixo mostra como as funções de texturas e o formato de base interna determinam a fórmula de aplicação de textura usada para cada componente de textura.

Valores utilizados na tabela : GL_ALPHA (A), GL_LUMINANCE (L), GL_LUMINANCE_ALPHA (L e A),GL_INTENSITY (I), GL_RGB (C), e GL_RGBA (C e A).

| Formato Interno Base | GL_MODULATE | GL_DECAL | GL_BLEND | GL_REPLACE |
|---|---|---|---|---|
| **GL_ALPHA** | $C_v = C_f$<br>$A_v = A_t A_f$ | indefinido | $C_v = C_f$<br>$A_v = A_t A_f$ | $C_v = C_f$<br>$A_v = A_t$ |
| **GL_LUMINANCE** | $C_v = L_t C_f$<br>$A_v = A_f$ | indefinido | $C_v = (1 - L_t)C_f + L_t C_c$<br>$A_v = A_f$ | $C_v = L_t$<br>$A_v = A_f$ |
| **GL_LUMINANCE_ALPHA** | $C_v = L_t C_f$<br>$A_v = A_t A_f$ | indefinido | $C_v = (1 - L_t)C_f + L_t C_c$<br>$A_v = A_t A_f$ | $C_v = L_t$<br>$A_v = A_t$ |
| **GL_INTENSITY** | $C_v = I_t C_f$<br>$A_v = I_t A_f$ | indefinido | $C_v = (1 - I_t)C_f + I_t C_c$<br>$A_v = I_t A_f$ | $C_v = I_t$<br>$A_v = I_t$ |
| **GL_RGB** | $C_v = C_t C_f$<br>$A_v = A_f$ | $C_v = C_t$<br>$A_v = A_f$ | $C_v = (1 - C_t)C_f + C_t C_c$<br>$A_v = A_f$ | $C_v = C_t$<br>$A_v = A_f$ |
| **GL_RGBA** | $C_v = C_t C_f$<br>$A_v = A_t A_f$ | $C_v = (1 - A_t)C_f + A_t C_t$<br>$A_v = A_f$ | $C_v = (1 - C_t)C_f + C_t C_c$<br>$A_v = A_t A_f$ | $C_v = C_t$<br>$A_v = A_t$ |

Tabela 10. Fórmulas de aplicação de texturas – glTexEnv*

### 9.1.5 - Atribuição de coordenadas às Texturas

Uma vez que o mapeamento de textura foi feito, deve-se prover ambas as coordenadas : a do objeto e a da textura para cada vértice. Após as transformacoes as coordenadas dos objetos determinam onde, na tela, um vertice em particular deverá ser renderizado. As coordenadas de texturas determinam qual textel no mapa de textura será atribuído a cada vértice. Exatamente do mesmo modo que as cores são interpoladas entre dois vértices, polígonos de sombras e linhas, coordenadas de texturas são interpoladas entre os vértices.

Coordenadas de texturas compreendem uma, duas, três ou quatro coordenadas. Usualmente estas coordenadas são referidas às coordenadas s, t, r e q para distinguir as coordenadas dos objetos (x, y, z e w). Para duas dimensões, usa-se as coordenadas s e t. A coordenada q, como w, é tipicamente o valor 1 e pode ser utilizada para criação de coordenadas homogêneas.

Comando :

*void **glTexCoord**{1234}{sifd}(TYPEcoords);*
*void **glTexCoord**{1234}{sifd}**v**(TYPE *coords);*

*Especifica a coordenada da textura corrente (s, t, r, q). A chamada subseqüente do comando glVertex*() atribui a coordenada corrente da textura ao vértice. Com glTexCoord1*(), a coordenada s é habilitada para especificar o valor, t e r são 0, e q tem o valor. A utilização de glTexCoord2*() permite a especificação de s e t; r e q são respectivamente 0 e . Com glTexCoord3*(), q é 1 e as outras coordenadas têm seus valores específicos. Todas as coordenadas com glTexCoord4*() podem ser especificadas.*

Exemplo :

```
glBegin(GL_QUADS);

      glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.5,-0.5,0.5);
      glTexCoord2f(1.0f, 0.0f);glVertex3f(-0.5,0.5,0.5);
      glTexCoord2f(1.0f, 1.0f);glVertex3f(0.5,0.5,0.5);
      glTexCoord2f(0.0f, 1.0f);glVertex3f(0.5,-0.5,0.5);

glEnd();
```

## 9.1.6 - Geração automática de coordenadas

O OpenLG permite a geração automática de coordenadas de texturas sem a necessidade da explicitação do comando glTexCoord*(). Para gerar coordenadas automaticamente utiliza-se o comando glTexGen().

Comando :

*void **glTexGen**{ifd}(GLenum coord, GLenum pname, TYPEparam);*
*void **glTexGen**{ifd}**v**(GLenum coord, GLenum pname, TYPE *param);*

*Função específica para geração automática de coordenadas para texturas. O primeiro parâmetro, coord deve ser GL_S, GL_T, GL_R, or GL_Q para indicar se a coordenada de textura s, t, r,ou q será gerada. O parâmetro pname é GL_TEXTURE_GEN_MODE, GL_OBJECT_PLANE, ou GL_EYE_PLANE. Se GL_TEXTURE_GEN_MODE, param é um inteiro (ou, na versão de vetor do comando, pontos para um inteiro) ou é GL_OBJECT_LINEAR, GL_EYE_LINEAR, ou GL_SPHERE_MAP. Estas constantes simbólicas determinam qual função será usada para gerar as coordenadas de textura. Com qualquer um dos possíveis valores para pname, param é um ponteiro para um array de valores (para a versão de vetor) especificando parâmetros para a função de geração de textura.*

### 9.1.6.1 Exemplo

```
/*
  Programa : texgen.c
  Exemplo integralmente retirado do livro : OpenGL Programming Guide - Red Book
  Descrição : Apresenta a geração automatica para aplicação de textura
              sobre um objeto
  Objetivo : Apresentação da geração automática de coordenadas de texturas

*/


#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdlib.h>
#include <stdio.h>

#define stripeImageWidth 32
GLubyte stripeImage[4*stripeImageWidth];
static GLuint texName;
void makeStripeImage(void)
{
   int j;
    for (j = 0; j < stripeImageWidth; j++)
    {
        stripeImage[4*j] = (GLubyte) ((j<=4) ? 255 : 0);
        stripeImage[4*j+1] = (GLubyte) ((j>4) ? 255 : 0);
        stripeImage[4*j+2] = (GLubyte) 0;
        stripeImage[4*j+3] = (GLubyte) 255;
     }
}
```

```
/* planes for texture coordinate generation */
static GLfloat xequalzero[] = {1.0, 0.0, 0.0, 0.0};
static GLfloat slanted[] = {1.0, 1.0, 1.0, 0.0};
static GLfloat *currentCoeff;
static GLenum currentPlane;
static GLint currentGenMode;

void init(void)
{

  glClearColor (0.0, 0.0, 0.0, 0.0);
  glEnable(GL_DEPTH_TEST);
  glShadeModel(GL_SMOOTH);
  makeStripeImage();
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glGenTextures(1, &texName);
  glBindTexture(GL_TEXTURE_1D, texName);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MAG_FILTER,GL_LINEAR);
  glTexParameteri(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER,GL_LINEAR);
  glTexImage1D(GL_TEXTURE_1D, 0, GL_RGBA, stripeImageWidth, 0,
               GL_RGBA, GL_UNSIGNED_BYTE, stripeImage);
  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
  currentCoeff = xequalzero;
  currentGenMode = GL_OBJECT_LINEAR;
  currentPlane = GL_OBJECT_PLANE;
  glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
  glTexGenfv(GL_S, currentPlane, currentCoeff);
  glEnable(GL_TEXTURE_GEN_S);
  glEnable(GL_TEXTURE_1D);
  glEnable(GL_CULL_FACE);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  glFrontFace(GL_CW);
  glCullFace(GL_BACK);
  glMaterialf (GL_FRONT, GL_SHININESS, 64.0);
}
void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix ();
  glRotatef(45.0, 0.0, 0.0, 1.0);
  glBindTexture(GL_TEXTURE_1D, texName);
  glutSolidTeapot(2.0);
  glPopMatrix ();
  glFlush();
}

void reshape(int w, int h)
{
  glViewport(0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
     glOrtho (-3.5, 3.5, -3.5*(GLfloat)h/(GLfloat)w,3.5*
              (GLfloat)h/(GLfloat)w, -3.5, 3.5);
  else
     glOrtho (-3.5*(GLfloat)w/(GLfloat)h,3.5*
              (GLfloat)w/(GLfloat)h, -3.5, 3.5, -3.5, 3.5);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void keyboard (unsigned char key, int x, int y)
{
   switch (key) {
      case 'e':
      case 'E':
         currentGenMode = GL_EYE_LINEAR;
         currentPlane = GL_EYE_PLANE;
         glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
         glTexGenfv(GL_S, currentPlane, currentCoeff);
```

```
            glutPostRedisplay();
            break;
         case 'o':
         case 'O':
            currentGenMode = GL_OBJECT_LINEAR;
            currentPlane = GL_OBJECT_PLANE;
            glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, currentGenMode);
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 's':
         case 'S':
            currentCoeff = slanted;
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 'x':
         case 'X':
            currentCoeff = xequalzero;
            glTexGenfv(GL_S, currentPlane, currentCoeff);
            glutPostRedisplay();
            break;
         case 27:
            exit(0);
            break;
         default:
         break;
}
}
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(256, 256);
   glutInitWindowPosition(100, 100);
   glutCreateWindow ("Geração Automática de Coordenadas de Texturas");
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

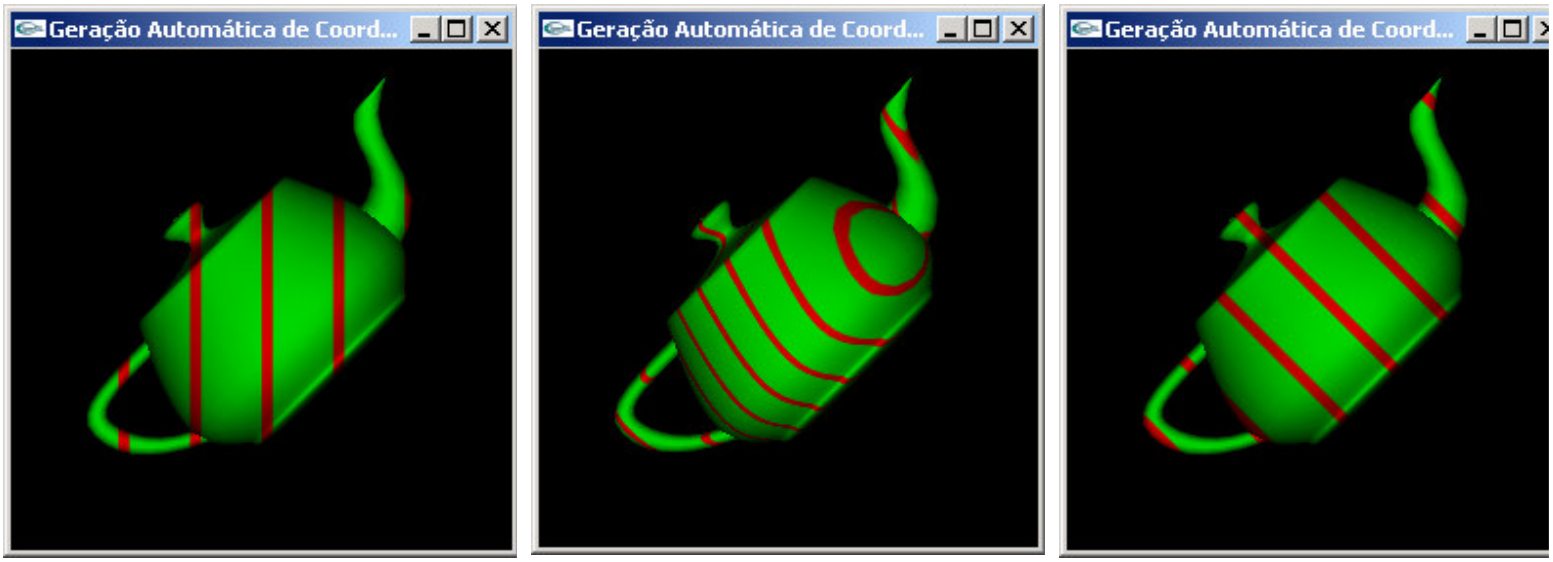



Figura 41. Exemplo de Geração automática de coordenadas de texturas

## 9.2 - Carga de texturas através de arquivos

Não existem funções específicas no OpenLG que permitem a carga direta de um arquivo de imagem para o mapeamento de texturas. Em muitos casos a biblioteca auxiliar do OpenGL GLAUX é utilizada. Uma vez que o enfoque da presente obra consiste em apresentar funcionalidades independente de plataforma ( utilização do GLUT ) será apresentado aqui a utilização das bibliotecas padrão do C/C++ aliadas às funcionalidades do GLU/GLUT.

A carga de textura através de arquivos consiste na leitura física do arquivo armazenado e armazenamento do mesmo em um array que será tratado pelas funções de criação e mapeamento de texturas.

Passos básicos utilizando-se um *raw bitmap :*

1 ) Definição do ponteiro para o array que conterá a imagem a ser carregada bem como o tipo de dados :

```
GLubyte *raw_bitmap ;
```

2) Abertura do Arquivo :

```
FILE *file;
file = fopen(file_name, "rb")
```

3) Alocação do espaço de memória necessário para armazenamento da imagem :

```
raw_bitmap = (GLubyte *)malloc(width * height * depth * (sizeof(GLubyte)));
```

Onde : *width* representa a largura da imagem, *height* a altura da imagem e *depth* a profundidade da imagem.

4) Leitura do arquivo diretamente para memória e fechamento do mesmo

```
fread  ( raw_bitmap , width * height * depth, 1 , file );
fclose ( file);
```

5) Definição do modo de armazenamento e criação do objeto de textura

```
GLuint  texture_id;
glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
glGenTextures ( 1, texture_id );
glBindTexture ( GL_TEXTURE_2D, texture_id );
```

6) Definição dos tipos de filtros a serem aplicados na textura.

Comando: ***glTexParameter****()

7) Definição do ambiente de textura :

Comando : ***glTexEnvf*** ()

8) Construção dos mipmaps

exemplo :

```
gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width, height, colour_type,
                    GL_UNSIGNED_BYTE, raw_bitmap );
```

9) Liberação  do espaço de memória reservado para carga.

```
free ( raw_bitmap );
```

Importante observar que a altura e a largura do arquivo, para arquivos bmp e jpg, devem ser $2^n$ . Exemplo : 4x4, 16x32, 256 x 256;

Uma vez que os passos foram seguidos a textura está pronta para ser utilizada pelo programa.

Nestes passos foi considerada a carga de um arquivo do tipo RAW. O formato RAW é basicamente um tipo de formato de importação e exportação ao invés de um formato de armazenamento contendo os valores “brutos” da imagem . Este tipo de formato pode ser gerado através de ferramentas gráficas como o Adobe Photoshop.

Qualquer outro tipo de formato pode ser carregado como textura no OpenGL. No entanto funções específicas de cargas devem ser criadas considerando estes tipos (bmp, tga, tiff, jpg, etc..). Para os arquivos do tipo bmp e jpg o tamanho da imagem (altura e largura ) devem ser do tipo $2^n$ em modo RGB. Ex.: 4x4, 16x32, 256x512 etc.

## 9.2.1 - Exemplo de carga de texturas – Arquivo .RAW

```
/* Exemplo9.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta a utilização de texturas em um cubo
*          carregando estas texturas através de arquvios
*
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include <stdio.h>

static int angulox = 0, anguloy = 0;
int posicaoluz = 0;
int orientacao = 1;
GLubyte * earthTex;


int valx =-3.0, valy=2.0, valz=100.0;

GLuint  texture_id[4];


/*
  Função responsável pela carga de textura de um arquivo
  no formato RAW.
*/
void load_texture ( char *file_name, int width, int height, int depth,
                    GLenum colour_type, GLenum filter_type )
{
   GLubyte *raw_bitmap ;
   FILE *file;

   if (( file = fopen(file_name, "rb"))==NULL )
   {
      printf ( "Arquivo não Encontrado : %s\n", file_name );
      exit   ( 1 );
   }

   raw_bitmap = (GLubyte *)malloc(width * height * depth * (sizeof(GLubyte)));

   if ( raw_bitmap == NULL )
   {
      printf ( "Impossível alocar espaço de memória para a textura\n" );
      fclose ( file );
      exit   ( 1 );
   }

   fread  ( raw_bitmap , width * height * depth, 1 , file );
   fclose ( file);

   //  Define o tipo de filtro a ser utilizado
   glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, filter_type );
   glTexParameteri ( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, filter_type );

   //  Define o ambiente de Textura
   glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );

   //  Construção dos MipMaps
   gluBuild2DMipmaps ( GL_TEXTURE_2D, colour_type, width, height, colour_type,
                       GL_UNSIGNED_BYTE, raw_bitmap );

   //  Libera a memoria alocada para o array
   free ( raw_bitmap );
}



/*
    Define o modelo de cores a ser utilizado
    alem fazer a carga inicial e criacao das texturas
    utilizadas.
```

```
    Cria 4 Texturas e faz a carga da imagem para as mesmas.
*/

void init(void)
{

   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 0.0, 0.0, 3.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);

   glEnable ( GL_TEXTURE_2D );

   glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
   glGenTextures ( 4, texture_id );

   glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
   load_texture  ( "unibh.raw", 512,256, 3, GL_RGB, GL_NEAREST );

   glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
   load_texture  ( "earth.raw", 512, 256, 3, GL_RGB, GL_NEAREST );

   glBindTexture ( GL_TEXTURE_2D, texture_id[2] );
   load_texture  ( "psico.raw", 512, 256, 3, GL_RGB, GL_NEAREST );

   glBindTexture ( GL_TEXTURE_2D, texture_id[3] );
   load_texture  ( "9people.raw", 388, 529, 3, GL_RGB, GL_NEAREST );


   glColor3f (0.89, 0.79, 0.09);
   glMateriali(GL_FRONT,GL_SHININESS,100);
   glColor4f     ( 1.0, 1.0, 1.0, 1.0  );

}

/*
  Função responsável pelo desenho de um cubo e aplicacao de texturas
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado.
*/

void display(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };


  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
glColor3f (1.0, 1.0, 1.0);


glPopMatrix();

  glPushMatrix();

    glRotatef (angulox, 1.0, 0.0, 0.0);
    glRotatef (anguloy, 0.0, 1.0, 0.0);

 glPushMatrix(); // Construção do Cubo e associacao de coordenadas de texturas;


    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
    glBegin(GL_QUADS);
        //Frente
               glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.5,-0.5,0.5);
               glTexCoord2f(1.0f, 0.0f);glVertex3f(-0.5,0.5,0.5);
               glTexCoord2f(1.0f, 1.0f);glVertex3f(0.5,0.5,0.5);
               glTexCoord2f(0.0f, 1.0f);glVertex3f(0.5,-0.5,0.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
    glBegin(GL_QUADS);
        // Tras:
             glTexCoord2f(1.0f, 0.0f);glVertex3f(-0.5,-0.5,-0.5);
             glTexCoord2f(1.0f, 1.0f);glVertex3f(0.5,-0.5,-0.5);
             glTexCoord2f(0.0f, 1.0f);glVertex3f(0.5,0.5,-0.5);
             glTexCoord2f(0.0f, 0.0f);glVertex3f(-0.5,0.5,-0.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[2] );
    glBegin(GL_QUADS);
        //Topo:
             glTexCoord2f(0.0f, 1.0f);glVertex3f(-0.5,0.5,-0.5);
             glTexCoord2f(0.0f, 0.0f);glVertex3f(0.5,0.5,-0.5    );
             glTexCoord2f(1.0f, 0.0f);glVertex3f(0.5,0.5,0.5);
             glTexCoord2f(1.0f, 1.0f);glVertex3f(-0.5,0.5,0.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
    glBegin(GL_QUADS);
        // baixo:
             glTexCoord2f(1.0f, 1.0f);glVertex3f(-0.5,-0.5,-0.5);
             glTexCoord2f(0.0f, 1.0f);glVertex3f(-0.5,-0.5,0.5);
             glTexCoord2f(0.0f, 0.0f);glVertex3f(0.5,-0.5,0.5);
             glTexCoord2f(1.0f, 0.0f);glVertex3f(0.5,-0.5,-0.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);

    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[3] );
    glBegin(GL_QUADS);
      //Esquerda
             glTexCoord2f(1.0,0.0);glVertex3f(-0.5,-0.5,-0.5);
             glTexCoord2f(1.0,1.0);glVertex3f(-0.5,0.5,-0.5);
             glTexCoord2f(0.0,1.0);glVertex3f(-0.5,0.5,0.5);
             glTexCoord2f(0.0,0.0);glVertex3f(-0.5,-0.5,0.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);


    glEnable(GL_TEXTURE_2D);
    glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
    glBegin(GL_QUADS);
      //Direita
      glTexCoord2f(0.0,0.0);glVertex3f(0.5,-0.5,-0.5);
```

```
                glTexCoord2f(1.0,0.0);glVertex3f(0.5,-0.5,0.5);
                glTexCoord2f(1.0,1.0);glVertex3f(0.5,0.5,0.5);
                glTexCoord2f(0.0,1.0);glVertex3f(0.5,0.5,-0.5);
      glEnd();
      glDisable(GL_TEXTURE_2D);

       glPopMatrix();

 glPopMatrix();

 glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(20.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);


}

/*
   Função responsável pelo controle de teclado
   quando pressionada a tecla EXC o programa é encerrado
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 27 :
            exit(0);
            break;

      default:
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
*/
void spinDisplay(void)
{
  angulox  = (angulox + (1 * orientacao)) % 360;
  anguloy  = (anguloy + (2 * orientacao)) % 360;
  glutPostRedisplay();
}


/*
  Esta função irá controlar os botões do mouse.
  Se pressionado o botão da esquerda ela define
  a função spinDisplay como a função de "idle" do GLUT
  e rotaciona para a sentido anti horario fazendo o contrario
  no pressionamento do botão da direita
```

```
  o comando glutIdelFunc, executa uma determinada função quando
  nenhum evento estiver ocorrendo. (pressionamento de botões etc.)
  Quando o botão do meio é pressionado a função de Idle recebe NULL
  desabilitando a animação

*/
void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
           {
              orientacao = 1;
              glutIdleFunc(spinDisplay);
           }
           break;
      case GLUT_MIDDLE_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(NULL);
           break;
      case GLUT_RIGHT_BUTTON:
           if (state == GLUT_DOWN)
           {
             orientacao = -1;
             glutPostRedisplay();
           }
           break;
      default:
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 9 - Carga de Texturas");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

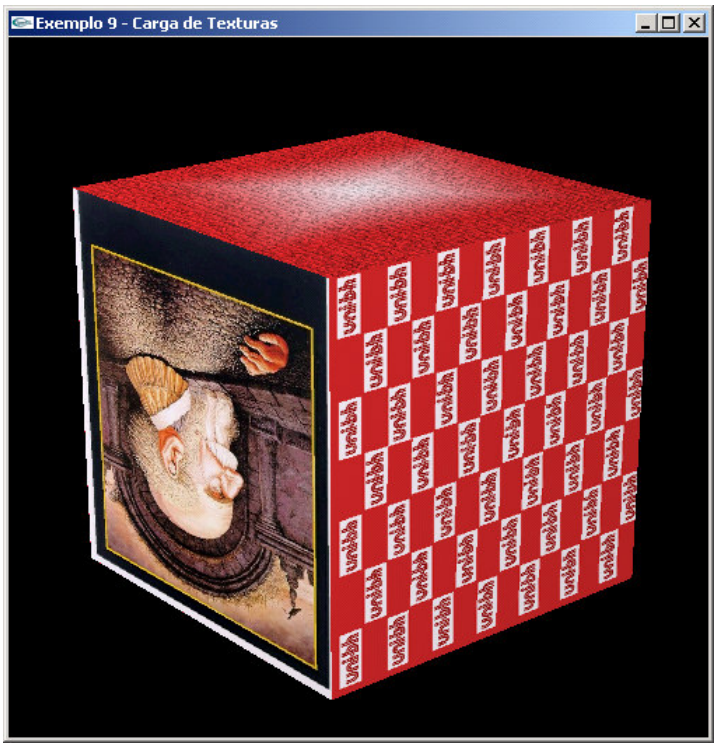



Figura 42. Exemplo 9 – Cargas de texturas através de arquivos RAW

## 9.2.2 - Exemplo de carga de texturas – Arquivo .BMP

```
/* Exemplo10.c - Marcionílio Barbosa Sobrinho
*  Carga de textura através de arquivo BMP
* Referência do Código: OpenGL Programming Guide - RedBook
*         planet.c, movelight.c, material.c
*
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdio.h>
static int year = 0, day = 0;
int posicaoluz = 0;

int tx;

GLuint  texture_id[1];

/*
  Função responsável pela carga de
  um arquivo BMP

  Esta função utiliza leitura direta do BMP sem
  a necessidade de outras bibliotecas assim
  segue abaixo a descrição de cada deslocamento
  do Header.
  Referencia :http://www.fastgraph.com/help/bmp_header_format.html

 Formato do header de arquivos BMP (Windows)
 Windows BMP files begin with a 54-byte header:
 offset  size   description
   0       2     signature, must be 4D42 hex
   2       4     size of BMP file in bytes (unreliable)
   6       2     reserved, must be zero
   8       2     reserved, must be zero
  10       4     offset to start of image data in bytes
  14       4     size of BITMAPINFOHEADER structure, must be 40
  18       4     image width in pixels
  22       4     image height in pixels
  26       2     number of planes in the image, must be 1
  28       2     number of bits per pixel (1, 4, 8, or 24)
  30       4     compression type (0=none, 1=RLE-8, 2=RLE-4)
  34       4     size of image data in bytes (including padding)
  38       4     horizontal resolution in pixels per meter (unreliable)
  42       4     vertical resolution in pixels per meter (unreliable)
  46       4     number of colors in image, or zero
  50       4     number of important colors, or zero

*/

int  LoadBMP(char *filename)
 {
    #define SAIR        {fclose(fp_arquivo); return -1;}
    #define CTOI(C)     (*(int*)&C)

    GLubyte     *image;
    GLubyte     Header[0x54];
    GLuint      DataPos, imageSize;
    GLsizei     Width,Height;

    int nb = 0;


    // Abre o arquivo e efetua a leitura do Header do arquivo BMP
    FILE * fp_arquivo = fopen(filename,"rb");
    if (!fp_arquivo)
        return -1;
    if (fread(Header,1,0x36,fp_arquivo)!=0x36)
        SAIR;
```

```
    if (Header[0]!='B' || Header[1]!='M')
        SAIR;
    if (CTOI(Header[0x1E])!=0)
        SAIR;
    if (CTOI(Header[0x1C])!=24)
        SAIR;

    // Recupera a informação dos atributos de
    // altura e largura da imagem

    Width   = CTOI(Header[0x12]);
    Height  = CTOI(Header[0x16]);
    ( CTOI(Header[0x0A]) == 0 ) ? ( DataPos=0x36 ) : ( DataPos = CTOI(Header[0x0A]) );

    imageSize=Width*Height*3;

    // Efetura a Carga da Imagem
    image = (GLubyte *) malloc ( imageSize );
    int retorno;
    retorno = fread(image,1,imageSize,fp_arquivo);

    if (retorno !=imageSize)
     {
        free (image);
        SAIR;
     }

    // Inverte os valores de R e B
    int t, i;

    for ( i = 0; i < imageSize; i += 3 )
     {
        t = image[i];
        image[i] = image[i+2];
        image[i+2] = t;
     }

    // Tratamento da textura para o OpenGL

    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

    glTexEnvf ( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );

    // Faz a geraçao da textura na memória
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, Width, Height, 0, GL_RGB, GL_UNSIGNED_BYTE,
image);

    fclose (fp_arquivo);
    free (image);
    return 1;
 }


void init(void)
{

   tx=0;
   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);
```

```
   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);


   /*
     Habilita a Texturizacao.
     Criacao inicial das texturas.
   */

   glEnable ( GL_TEXTURE_2D );
   glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
   glGenTextures ( 1, texture_id );
   glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
   LoadBMP ("tex3.bmp");


}

/*
  Esta função cria uma esfera através do
  comando gluNewQuadric do GLU
  Esta esfera permite o mapeamento de texturas

  comando : gluSphere ( GLUquadricObj, raio, subdivisoes_em_torno_de_Z,
                        subdivisoes_ao_longo_de_Z )
*/

void esfera ( int raio,int longitude,int latitude  )
{
   GLUquadricObj*  q = gluNewQuadric ( );
   gluQuadricDrawStyle ( q, GLU_FILL   );
   gluQuadricNormals   ( q, GLU_SMOOTH );
   gluQuadricTexture   ( q, GL_TRUE    );
   gluSphere ( q, raio, longitude, latitude );
   gluDeleteQuadric ( q );
}


/*
  Função responsável pelo desenho da esfera.
  E da aplicação da textura na mesma
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado.
*/

void display(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

  /* Posição da luz */
  GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);

  /* Armazena o estado anterior para
     rotação da posição da luz */

  glPushMatrix () ;
```

```
    glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
    glLightfv (GL_LIGHT0, GL_POSITION, posicao);

  glPopMatrix(); // Posição da Luz

    /* Armazena a situação atual da pilha de matrizes */

 glPushMatrix ();
     glRotatef (tx, 1.0, 0.0, 0.0);
     glTranslatef( 0.0, 0.0, 2.0);
 glPushMatrix ();
     glTranslatef (0.0, 0.0, -3.0);
 glPushMatrix ();
     glRotatef (9, 0.0, 0.0, 1.0);

    glPushMatrix();
      glRotatef ((GLfloat) year, 1.0, 0.0, 0.0);
      //glTranslatef (tx, ty, tz);
      glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
      glColor3f (1.0, 1.0, 1.0);


      /* Define a propriedade do material */
      //refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,20);


      /* Habilita a textura e cria a esfera */
      glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
      esfera ( 1.50, 20, 18);

    glPopMatrix();
 glPopMatrix();
 glPopMatrix();
 glPopMatrix();

 // Executa os comandos
  glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla ESC
   o programa é terminado.
   a tecla D desabilita a textura enquanto
   a tecla H habilita a mesma.
*/

void keyboard (unsigned char key, int x, int y)
```

```
{
  switch (key) {
      case 'd':
      case 'D':
           glDisable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
      case 'A':
      case 'a':
           glEnable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
*/

void spinDisplay(void)
{
  year = (year + 1) % 360;
  day  = (day + 2 ) % 360;
  tx = (tx + 1) % 360 ;

  glutPostRedisplay();
}


/*
  Esta função irá controlar os botões do mouse.
  Se pressionado o botão da esquerda ela define
  a função spinDisplay como a função de "idle" do GLUT
  o comando glutIdelFunc, executa uma determinada função quando
  nenhum evento estiver ocorrendo. (pressionamento de botões etc.)
  Quando o botão do meio é pressionado a função de Idle recebe NULL
  desabilitando a animação
*/
void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(spinDisplay);
           break;
      case GLUT_MIDDLE_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(NULL);
           break;
      case GLUT_RIGHT_BUTTON:
           posicaoluz = (posicaoluz + 1) % 360;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (800, 600);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 10 - Carga de Textura .BMP");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
```

```
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

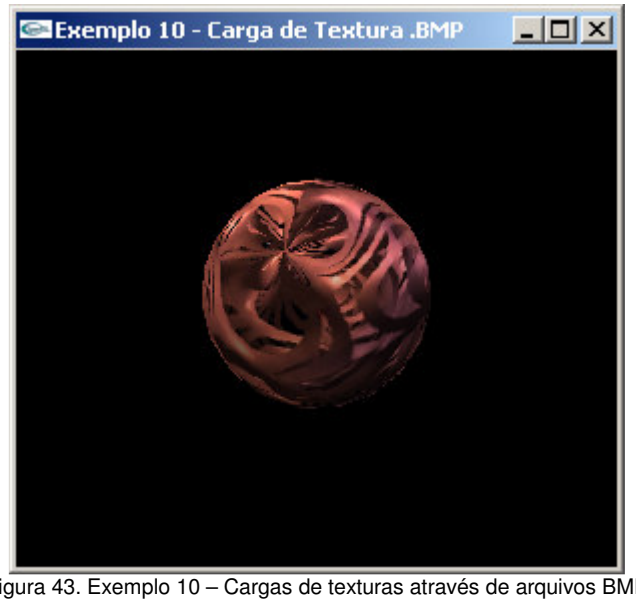



Figura 43. Exemplo 10 – Cargas de texturas através de arquivos BMP

### 9.2.3 - Exemplo de carga de texturas – Arquivo .JPG

Para carga de texturas de arquivos do tipo jpeg, é necessária a utilização de uma biblioteca auxiliar para carga e descomactação destes arquivos.

O nome da biblioteca é libjpeg, e foi desenvolvida por um grupo independente intitulado : Independent JPEG Group (http://www.ijg.org/).

Para utilização do exemplo seguinte esta biblioteca deve será utilizada.

No compilador Dev-Cpp, esta bilblioteca deve ser incluída nos parâmetro do linker. ( Em Opções do Projeto -> Parâmetros -> Linker, incluir : -ljpeg ).

```
/* Exemplo11.c - Marcionílio Barbosa Sobrinho
*  Carga de textura através de arquivo JPG
* Referência do Código: OpenGL Programming Guide - RedBook
*         planet.c, movelight.c, material.c
*
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdio.h>

// Como a biblioteca foi compilada
// como o compilador C devemos incluir
// a seguinte diretiva antes de adicionar
// os cabeçalhos
extern "C"
{
    #include <jpeglib.h>
    #include <jerror.h>
}
static int year = 0, day = 0;
int posicaoluz = 0;

int tx;

GLuint  texture_id[1];

int  LoadJPEG ( char *filename )
{
    // Contém as informações do arquivo
    struct  jpeg_decompress_struct cinfo;

    struct  jpeg_error_mgr jerr;
    GLubyte *linha;

    // Conterá a imagem carregada
    GLubyte *image;
    // Tamanho da Imagem
    int     ImageSize;

    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_decompress(&cinfo);

    // Abre o arquivo, lê seu cabeçalho
    // e processa a descompressão da mesma
    FILE *fd_arquivo=fopen(filename, "rb");
    jpeg_stdio_src(&cinfo, fd_arquivo);
    jpeg_read_header(&cinfo, TRUE);
```

```
    jpeg_start_decompress ( &cinfo );

    ImageSize = cinfo.image_width * cinfo.image_height * 3;
    image = (GLubyte *) malloc ( ImageSize );
    linha=image;

    while ( cinfo.output_scanline < cinfo.output_height )
     {
        linha = image + 3 * cinfo.image_width * cinfo.output_scanline;
        jpeg_read_scanlines ( &cinfo, &linha, 1 );
     }

    jpeg_finish_decompress(&cinfo);
    jpeg_destroy_decompress(&cinfo);


    //Aplicação de filtros para tratamento da imagem
    //pelo OpenGL
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

    // Efetua a geração da imagem na memória
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, cinfo.image_width, cinfo.image_height,
                                0, GL_RGB, GL_UNSIGNED_BYTE, image);

    fclose (fd_arquivo);
    free (image);
    return 1;
 }


void init(void)
{

   tx=0;
   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);


   /*
     Habilita a Texturizacao.
     Criacao inicial das texturas.
   */

   glEnable ( GL_TEXTURE_2D );
   glPixelStorei ( GL_UNPACK_ALIGNMENT, 1 );
   glGenTextures ( 2, texture_id );
   glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
   LoadJPEG ("imagem1.jpg");
```

```
   glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
   LoadJPEG ("imagem2.jpg");
}

/*
  Função responsável pelo desenho da esfera.
  E da aplicação da textura na mesma
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado.
*/

void display(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

  /* Posição da luz */
  GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
  */
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glColor3f (1.0, 1.0, 1.0);

  /* Armazena o estado anterior para
     rotação da posição da luz */

  glPushMatrix () ;

    glRotated ((GLdouble) posicaoluz, 1.0, 0.0, 0.0);
    glLightfv (GL_LIGHT0, GL_POSITION, posicao);

  glPopMatrix(); // Posição da Luz

    /* Armazena a situação atual da pilha de matrizes */

 glPushMatrix ();
     glRotatef (tx, 1.0, 0.0, 0.0);
     glTranslatef( 0.0, 0.0, 2.0);
 glPushMatrix ();
     glTranslatef (0.0, 0.0, -3.0);
 glPushMatrix ();
     glRotatef (9, 0.0, 0.0, 1.0);

    glPushMatrix();
      glRotatef ((GLfloat) year, 1.0, 0.0, 0.0);
      //glTranslatef (tx, ty, tz);
      glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
      glColor3f (1.0, 1.0, 1.0);


      /* Define a propriedade do material */
      //refletância do material
      glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
        // Define a concentração do brilho
      glMateriali(GL_FRONT,GL_SHININESS,20);


      /* Habilita a textura  */
      glEnable ( GL_TEXTURE_2D );
      glBindTexture ( GL_TEXTURE_2D, texture_id[0] );
      glutSolidTeapot(1.0);
      glDisable ( GL_TEXTURE_2D );

      glPushMatrix ();
          glRotatef (tx, 1.0, 0.0, 0.0);
          glTranslatef( 0.0, 0.0, 2.0);
          glPushMatrix ();
             glTranslatef (0.0, 0.0, -3.0);
             glPushMatrix ();
```

```
               glTranslatef(-3.0,0.0,0.0);
               glEnable ( GL_TEXTURE_2D );
               glBindTexture ( GL_TEXTURE_2D, texture_id[1] );
               glutSolidTeapot(1.0);
               glDisable ( GL_TEXTURE_2D );
            glPopMatrix();
         glPopMatrix();
      glPopMatrix();

    glPopMatrix();
 glPopMatrix();
 glPopMatrix();
 glPopMatrix();

 // Executa os comandos
  glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla ESC
   o programa é terminado.
   a tecla D desabilita a textura enquanto
   a tecla H habilita a mesma.
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 'd':
      case 'D':
           glDisable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
      case 'A':
      case 'a':
           glEnable(GL_TEXTURE_2D);
           glutPostRedisplay();
           break;
  }
}

/*
  Esta função é chamada quando o botão esquerdo do
  mouse é pressionado, a mesma irá calcular um novo
  valor para os valores dos ângulos contidos em year e day
*/

void spinDisplay(void)
{
  year = (year + 1) % 360;
  day  = (day + 2 ) % 360;
```

```
  tx = (tx + 1) % 360 ;

  glutPostRedisplay();
}


/*
  Esta função irá controlar os botões do mouse.
  Se pressionado o botão da esquerda ela define
  a função spinDisplay como a função de "idle" do GLUT
  o comando glutIdelFunc, executa uma determinada função quando
  nenhum evento estiver ocorrendo. (pressionamento de botões etc.)
  Quando o botão do meio é pressionado a função de Idle recebe NULL
  desabilitando a animação
*/
void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(spinDisplay);
           break;
      case GLUT_MIDDLE_BUTTON:
           if (state == GLUT_DOWN)
              glutIdleFunc(NULL);
           break;
      case GLUT_RIGHT_BUTTON:
           posicaoluz = (posicaoluz + 1) % 360;
           glutPostRedisplay();
           break;
      default:
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (800, 600);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 11 - Carga de Textura .JPG");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMouseFunc(mouse);
  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```


Figura 44. Exemplo 11 – Cargas de texturas através de arquivos JPG

# Capítulo 10 - Sombra Planar

Uma sombra é produzida quando incide sobre um objeto uma fonte de luz, e este objeto retém esta luz de forma que ela não incida sobre um outro objeto qualquer ou superfície. A área na superfície, referente ao objeto no qual a luz é retida, aparece escura.

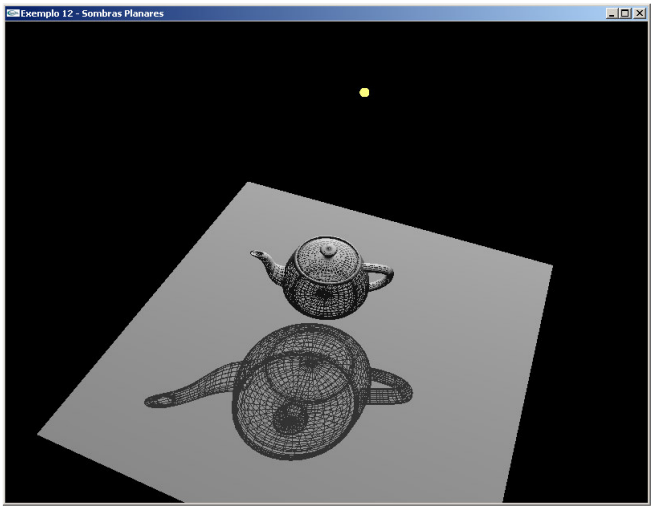



Figura 45. Sombra planar

## 10.1 - Calculando a Sombra de um objeto

Toda possível projeção de espaço tridimensional para espaço tridimensional pode ser obtida através de uma matriz 4x4, passível de inversão, e de coordenadas homogêneas.

Se a matriz não pode ser invertida e tem grau 3, ela projeta o espaço tridimensional sobre um plano bidimensional. Para encontrar a sombra de um objeto arbitrário em um plano arbitrário de uma fonte de luz arbitrária, é necessário encontrar uma matriz que representa aquela projeção, multiplicá-la na pilha de matriz, e desenhar o objeto na cor de sombra. É importante ter em mente que é necessário que a sombra se projete sobre cada plano que será chamado de "superfície".

Por exemplo, se uma luz está na origem, e a equação do plano de superfície é ax+by+c+d=0. Determinando um vértice S=(sx,sy,sz,1), a linha de

iluminação por S inclui todos os pontos α S onde α é um número real arbitrário. O ponto onde esta linha cruza com o plano acontece quando

***α (a*sz+b*sy+c*sz) + d = 0,***

então

***α = - δ/(a*sx+b*sy+c*sz).***

Substituindo na equação anterior nós obtemos :

***- δ (σ ξ, σ ψ , σ ζ ) / ( α * σ ξ + β σ ψ + χ σ ζ )***

para o ponto de interseção

A matriz que mapeia S neste ponto para todo S é

$$\begin{bmatrix} -d & 0 & 0 & a \\ 0 & -d & 0 & b \\ 0 & 0 & -d & c \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Esta matriz pode ser usada quando a fonte de luz está na origem. Se a luz é de uma fonte infinita, então só existe um ponto S e uma direção D = (dx,dy,dz). Pontos ao longo da linha são determinados por :

***a(sx+ α dx)+b(sy+ α dy)+c(sz+ α dz)+d = 0***

Resolvendo para α, colocando de volta na equação da linha, e determinando uma matriz de projeção temos :

$$\begin{bmatrix} b*dy+c*dz & -a*dy & -a*dz & 0 \\ -b*dx & a*dx+c*dz & -b*dz & 0 \\ -c*dx & -c*dy & a*dx+b*dy & 0 \\ -d*dx & -d*dy & -d*dz & a*dx+b*dy*c*dz \end{bmatrix}$$

## 10.2 - Exemplo de Sombra Planar

```
/* Exemplo12.c - Marcionílio Barbosa Sobrinho
*              Criação de sombras planares.
*              Este programa tem como referência o programa
*              desenvolvido por : Kevin Harris (kevin@codesampler.com)
*              O programa original não era portável por estar vinculado
*              às bibliotecas padrão do windows.
* Referência do Código Original: OpenGL Super Bible
*         Shadow.cpp ( página 339 )
*
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#include <stdio.h>

// Variaveis para controle de giro de visao
float  g_fSpinX_L =   0.0f;
float  g_fSpinY_L = -10.0f;

// Controle do Giro do Objeto
float  g_fSpinX_R =   0.0f;
float  g_fSpinY_R =   0.0f;

// Matriz da sombra
float g_shadowMatrix[16];
// Posição  da fonte de luz no espaço
float g_lightPosition[] = { 2.0f, 6.0f, 0.0f, 1.0f };

// Controla o objeto a ser desenhado

int objeto = 1;

/*
   Variáveis para desenho da
   superfície de projeção da sombra
*/


struct Vertex
{
    float nx, ny, nz;
    float x, y, z;
};

Vertex g_floorQuad[] =
{
    { 0.0f, 1.0f,  0.0f,  -5.0f, 0.0f, -5.0f },
    { 0.0f, 1.0f,  0.0f,  -5.0f, 0.0f,  5.0f },
    { 0.0f, 1.0f,  0.0f,   5.0f, 0.0f,  5.0f },
    { 0.0f, 1.0f,  0.0f,   5.0f, 0.0f, -5.0f },
};


void init(void)
{

//    glClearColor( 0.35f, 0.53f, 0.7f, 1.0f );
    glClearColor( 0.0f, 0.0f, 0.0f, 0.0f );
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);

   glShadeModel(GL_SMOOTH);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

```
    gluPerspective( 45.0f, 640.0f / 480.0f, 0.1f, 100.0f);

    float luzAmbiente[] = {0.2f, 0.2f, 0.2f, 1.0f};
    float luzDifusa[] = {1.0, 1.0, 1.0, 1.0};
    float luzEspecular[] = {1.0, 1.0, 1.0, 1.0};
    glLightfv(GL_LIGHT0, GL_DIFFUSE, luzDifusa);
    glLightfv(GL_LIGHT0, GL_SPECULAR, luzEspecular);
    glLightfv(GL_LIGHT0, GL_AMBIENT, luzAmbiente);
}


/*
  Esta função é responsável pela construção
  da matriz de sombra.
*/

void ConstroiMatrizSombra( float Matriz[16], float PosicaoLuz[4], float Plano[4] )
{
    float Ponto;

    // Calcula o ponto prodizido entre o plano e a posição da luz
    Ponto = Plano[0] * PosicaoLuz[0] +
            Plano[1] * PosicaoLuz[1] +
            Plano[1] * PosicaoLuz[2] +
            Plano[3] * PosicaoLuz[3];

    // Primeira Coluna da Matriz
    Matriz[0]  = Ponto - PosicaoLuz[0] * Plano[0];
    Matriz[4]  = 0.0f - PosicaoLuz[0] * Plano[1];
    Matriz[8]  = 0.0f - PosicaoLuz[0] * Plano[2];
    Matriz[12] = 0.0f - PosicaoLuz[0] * Plano[3];

    // Segunda Coluna da Matriz
    Matriz[1]  = 0.0f - PosicaoLuz[1] * Plano[0];
    Matriz[5]  = Ponto - PosicaoLuz[1] * Plano[1];
    Matriz[9]  = 0.0f - PosicaoLuz[1] * Plano[2];
    Matriz[13] = 0.0f - PosicaoLuz[1] * Plano[3];

    // Terceira Coluna da Matriz
    Matriz[2]  = 0.0f - PosicaoLuz[2] * Plano[0];
    Matriz[6]  = 0.0f - PosicaoLuz[2] * Plano[1];
    Matriz[10] = Ponto - PosicaoLuz[2] * Plano[2];
    Matriz[14] = 0.0f - PosicaoLuz[2] * Plano[3];

    // Quarta Coluna da Matriz
    Matriz[3]  = 0.0f - PosicaoLuz[3] * Plano[0];
    Matriz[7]  = 0.0f - PosicaoLuz[3] * Plano[1];
    Matriz[11] = 0.0f - PosicaoLuz[3] * Plano[2];
    Matriz[15] = Ponto - PosicaoLuz[3] * Plano[3];
}

/*
   Esta função é responsável por encontrar a
   equacao do plano com base em tres pontos
*/
void EncontraPlano( GLfloat plano[4], GLfloat v0[3], GLfloat v1[3], GLfloat v2[3] )
{
    GLfloat vec0[3], vec1[3];

    // Necessicta de 2 vetores para encontrar a interseção
    vec0[0] = v1[0] - v0[0];
    vec0[1] = v1[1] - v0[1];
    vec0[2] = v1[2] - v0[2];

    vec1[0] = v2[0] - v0[0];
    vec1[1] = v2[1] - v0[1];
    vec1[2] = v2[2] - v0[2];

    // Encontra o produto de interseção para adquirir A, B, e C da equacao do plano
    plano[0] =   vec0[1] * vec1[2] - vec0[2] * vec1[1];
    plano[1] = -(vec0[0] * vec1[2] - vec0[2] * vec1[0]);
    plano[2] =   vec0[0] * vec1[1] - vec0[1] * vec1[0];

    plano[3] = -(plano[0] * v0[0] + plano[1] * v0[1] + plano[2] * v0[2]);
```

```
}

/*
  Efetua o desenho da superfície de projeção
  da sombra
*/

void Superficie()
{
    glColor3f( 1.0f, 1.0f, 1.0f );
    glInterleavedArrays( GL_N3F_V3F, 0, g_floorQuad );
    glDrawArrays( GL_QUADS, 0, 4 );
}


/*
  Função responsável pelo desenho do objeto.
*/

void DesenhaObjeto( void)
{
  switch (objeto) {
     case 1 :
        glRotatef(90,1.0f,0.0,0.0);
        glutSolidTorus(0.4,0.8,30,35);
        break;
     case 2:
        glutSolidTeapot( 1.0 );
        break;
  }

}



/*
  Função responsável pelo desenho dos objetos
  bem como projeção da sombra.

*/

void display(void)
{
    //
    // Define o plano da superficie planar que terá a sombra projetada.
    //

    GLfloat PlanoSombra[4];
    GLfloat v0[3], v1[3], v2[3];

    // Para definir o plano que contém a superfíces são necessários
    // 3 vértices

    v0[0] = g_floorQuad[0].x;
    v0[1] = g_floorQuad[0].y;
    v0[2] = g_floorQuad[0].z;

    v1[0] = g_floorQuad[1].x;
    v1[1] = g_floorQuad[1].y;
    v1[2] = g_floorQuad[1].z;

    v2[0] = g_floorQuad[2].x;
    v2[1] = g_floorQuad[2].y;
    v2[2] = g_floorQuad[2].z;

    EncontraPlano( PlanoSombra, v0, v1, v2 );

    //
    // Constroi a matriz de sombra utilizando a posicao da luz corrente e o plano.
    //

    ConstroiMatrizSombra( g_shadowMatrix, g_lightPosition, PlanoSombra );

    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
glTranslatef( 0.0f, -2.0f, -15.0f );
glRotatef( -g_fSpinY_L, 1.0f, 0.0f, 0.0f );
glRotatef( -g_fSpinX_L, 0.0f, 1.0f, 0.0f );

//
// Desenha superfície
//

Superficie();

//
// Cria a sombra para o objeto utilizando a matriz de sombra
//

glDisable(GL_DEPTH_TEST);
glDisable(GL_LIGHTING);

// Define a cor que a sombra terá
glColor3f(0.2f, 0.2f, 0.2f);
glPushMatrix();
{
    glMultMatrixf((GLfloat *)g_shadowMatrix);

    // Posição e orientação do objeto
    // necessita ter as mesmas transformações
    // utilizadas para a criação do objeto em si
    glTranslatef( 0.0f, 2.5f, 0.0f );
    glRotatef( -g_fSpinY_R, 1.0f, 0.0f, 0.0f );
    glRotatef( -g_fSpinX_R, 0.0f, 1.0f, 0.0f );

    switch (objeto) {
        case 1 :
             glRotatef(90,1.0f,0.0,0.0);
             glutSolidTorus(0.4,0.8,30,35);
             break;
        case 2:
             glutSolidTeapot( 1.0 );
             break;
        case 3 :
             glRotatef(90,1.0f,0.0,0.0);
             glutWireTorus(0.4,0.8,30,35);
             break;
        case 4:
             glutWireTeapot( 1.0 );
             break;
     }


}
glPopMatrix();

glEnable(GL_DEPTH_TEST);
glEnable(GL_LIGHTING);

//
// Cria uma pequena esfera na posição da luz.
//

glDisable( GL_LIGHTING );

glPushMatrix();
{

    glLightfv( GL_LIGHT0, GL_POSITION, g_lightPosition );

    // Esfera representando a luz
    glTranslatef( g_lightPosition[0], g_lightPosition[1], g_lightPosition[2] );

    glColor3f(1.0f, 1.0f, 0.5f);
    glutSolidSphere( 0.1, 8, 8 );
}
```

```
    glPopMatrix();


glEnable( GL_LIGHTING );
    //
    // Cria um objeto.
    //


    glPushMatrix();
    {

        // Orientação e posição do objeto.
        glTranslatef( 0.0f, 2.5f, 0.0f );
        glRotatef( -g_fSpinY_R, 1.0f, 0.0f, 0.0f );
        glRotatef( -g_fSpinX_R, 0.0f, 1.0f, 0.0f );
        glColor3f(1, 0, 0);
        switch (objeto) {
            case 1 :
                 glRotatef(90,1.0f,0.0,0.0);
                 glutSolidTorus(0.4,0.8,30,35);
                 break;
            case 2:
                 glutSolidTeapot( 1.0 );
                 break;
            case 3 :
                 glRotatef(90,1.0f,0.0,0.0);
                 glutWireTorus(0.4,0.8,30,35);
                 break;
            case 4:
                 glutWireTeapot( 1.0 );
                 break;
         }

    }
    glPopMatrix();



    glutSwapBuffers();

}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(45.0, (GLfloat) w/(GLfloat) h, 0.1, 100.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla ESC
   o programa é terminado.
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
```

```
      case 27 :
           exit (1);
           break;

  }
}

/* Função responsável pelo controle das
   teclas especiais através do GLUT
   as setas mudam o posicionamento da luz na cena
*/
void Special_keyboard (int key, int x, int y)
{
  switch (key) {
      case GLUT_KEY_LEFT :
           g_lightPosition[0] -= 0.1f;
           glutPostRedisplay();
           break;
      case GLUT_KEY_UP :
           g_lightPosition[1] += 0.1f;
           glutPostRedisplay();
           break;
      case GLUT_KEY_DOWN :
           g_lightPosition[1] -= 0.1f;
           glutPostRedisplay();
           break;
      case GLUT_KEY_RIGHT :
           g_lightPosition[0] += 0.1f;
           glutPostRedisplay();
           break;
  }
}

/*
  As duas funcoes que seguem controlam os eventos do mouse
  o efeito esperado é a movimentacao de toda a cena e a movimentacao
  somente do objeto. Estes efeitos sao obtivos respectivamente atraves
  do pressionamento dos botoes esquerdo ou direito e a movimentacao do mouse
  na tela.
*/

typedef struct PONTO
{
  int x;
  int y;
} PONTO_T;

static PONTO_T ptLastMousePosit_L;
static PONTO_T ptCurrentMousePosit_L;
static int   bMousing_L;

static PONTO_T ptLastMousePosit_R;
static PONTO_T ptCurrentMousePosit_R;
static int  bMousing_R;


void mouse(int button, int state, int x, int y)
{
  switch (button) {
      case GLUT_LEFT_BUTTON:
           if (state == GLUT_DOWN)
           {
               ptLastMousePosit_L.x = ptCurrentMousePosit_L.x = x;
               ptLastMousePosit_L.y = ptCurrentMousePosit_L.y = y;
               bMousing_L = 1;
           }
           else
               bMousing_L = 0;
           glutPostRedisplay();
           break;
      case GLUT_MIDDLE_BUTTON :
           if (state == GLUT_DOWN )
           {
              objeto = objeto + 1;
```

```
            if (objeto > 4 )
               objeto = 1;
            glutPostRedisplay();
         }
         break;
      case GLUT_RIGHT_BUTTON:
         if (state == GLUT_DOWN)
         {
            ptLastMousePosit_R.x = ptCurrentMousePosit_R.x = x;
            ptLastMousePosit_R.y = ptCurrentMousePosit_R.y = y;
            bMousing_R = 1;
         }
         else
            bMousing_R = 0;
         glutPostRedisplay();
         break;
      default:
         break;
  }
}

/*
  Obtem a posicao atual da movimentacao do mouse se algum botao esta pressionado.
*/

void motion_mouse( int x, int y)
{
    ptCurrentMousePosit_L.x = x;
    ptCurrentMousePosit_L.y = y;
    ptCurrentMousePosit_R.x = x;
    ptCurrentMousePosit_R.y = y;

    if( bMousing_L )
    {
       g_fSpinX_L -= (ptCurrentMousePosit_L.x - ptLastMousePosit_L.x);
       g_fSpinY_L -= (ptCurrentMousePosit_L.y - ptLastMousePosit_L.y);
    }

    if( bMousing_R )
    {
       g_fSpinX_R -= (ptCurrentMousePosit_R.x - ptLastMousePosit_R.x);
       g_fSpinY_R -= (ptCurrentMousePosit_R.y - ptLastMousePosit_R.y);
    }

    ptLastMousePosit_L.x = ptCurrentMousePosit_L.x;
    ptLastMousePosit_L.y = ptCurrentMousePosit_L.y;
    ptLastMousePosit_R.x = ptCurrentMousePosit_R.x;
    ptLastMousePosit_R.y = ptCurrentMousePosit_R.y;
    glutPostRedisplay();
}


/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (800, 600);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 12 - Sombras Planares");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);

  /*
    Funcao de Callback que controla o
    pressionamento de teclas especiais
    F1..F12, END, DELETE,SETAS etc..
  */
  glutSpecialFunc(Special_keyboard);
  glutMouseFunc(mouse);
```

```
  /*
    Funcao de Callback que controla
    a posição atual do ponteiro do mouse
    se algum dos botoes (esquerdo, direito, centro)
    esta pressionado.
  */
  glutMotionFunc(motion_mouse);

  glutKeyboardFunc(keyboard);
  glutMainLoop();
  return 0;
}
```

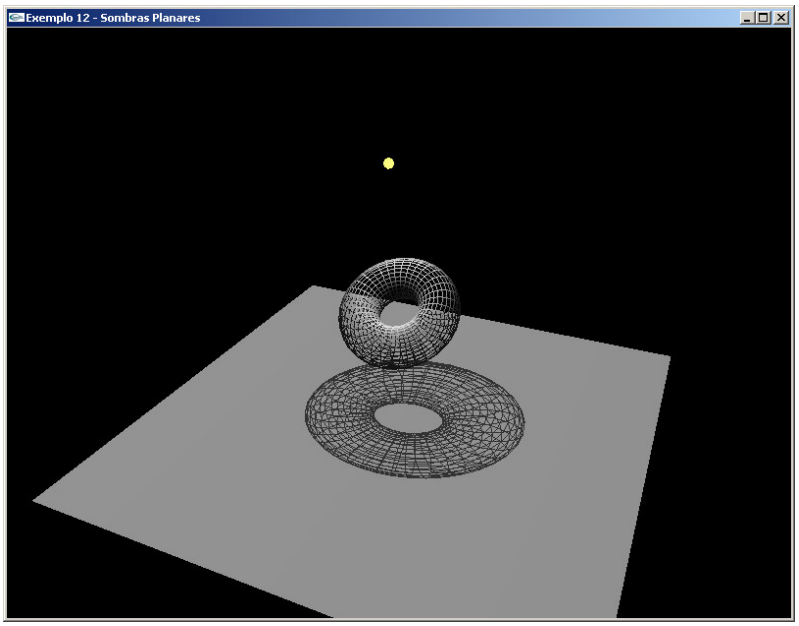



Figura 46. Exemplo 12 – Sombra planar

# Capítulo 11 - Blending

As funções de mistura de cores (blending) suportam efeitos como transparência que pode ser usada na simulação de janelas, copos, e outros objetos transparentes.

Quando “blending” está habilitado, o valor de alfa é usado freqüentemente para combinar o valor de cor do fragmento que está sendo processado com o do pixel armazenado no framebuffer. O efeito de “blending” acontece após a sena ter sido processada e convertida em fragmentos, mas antes os pixels finais são retirados do framebuffer.

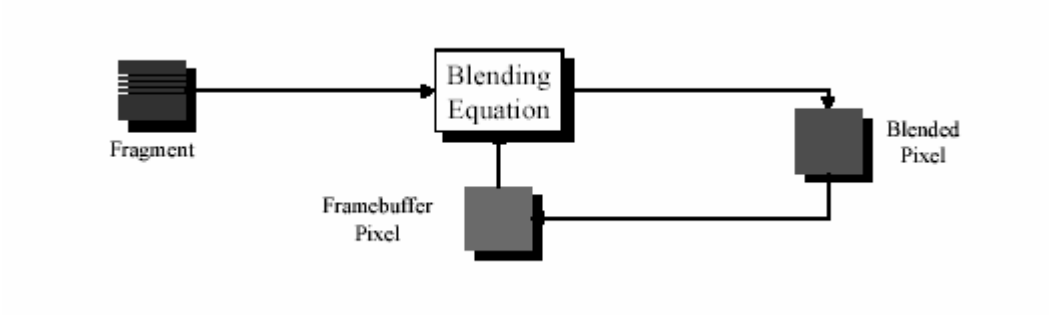



Figura 47. Processamento do “blend”

Durante o “blending”, valores de cores do fragmento entrante (a fonte) são combinados com os valores de cor do pixel atualmente armazenado (o destino) em um processo de duas fases. Primeiro deve ser especificado como calcular os fatores da fonte e do destino. Estes fatores são quádruplas de RGBA que são multiplicados por cada valor de componente R, G, B, e A na fonte e no destino, respectivamente. Então as componentes correspondentes nos dois conjuntos de quádruplas RGBA são somadas.

(RsSr+RdDr, GsSg+GdDg, BsSb+BdDb, AsSa+AdDa)

onde : (Sr, Sg, Sb, Sa) são as componentes da fonte e (Dr, Dg, Db, Da) são as compontentes do destino.

## 11.1 - Comandos Opengl

Para habilitar o “blending” deve ser utilizado o comando glEnable(GL_BLEND), e para desabilitar o comando glDisable(GL_BLEND)

É necessária a definição da função de blending que é feita através do comando:

*void* ***glBlendFunc****(GLenum sfactor, GLenum dfactor);*

Esta função controla como os valores de cor no fragmento que é processado (a fonte) é combinado com os valores já armazenados no framebuffer (o destino). O parâmetro sfactor indica como computar o fator de blending da fonte; dfactor indica como computar o fator de blending do destino.

Valores possíveis para os parâmetros de glBlendFunc :

| Constante | Aplicado a | Fator de Blending Calculado |
|---|---|---|
| GL_ZERO | Fonte ou Destino | (0, 0, 0, 0) |
| GL_ONE | Fonte ou Destino | (1, 1, 1, 1) |
| GL_DST_COLOR | Fonte | (Rd, Gd, Bd, Ad) |
| GL_SRC_COLOR | Destino | (Rs, Gs, Bs, As) |
| GL_ONE_MINUS_DST_COLOR | Fonte | (1, 1, 1, 1)-(Rd, Gd, Bd, Ad) |
| GL_ONE_MINUS_SRC_COLOR | Destino | (1, 1, 1, 1)-(Rs, Gs, Bs, As) |
| GL_SRC_ALPHA | Fonte ou Destino | (As, As, As, As) |
| GL_ONE_MINUS_SRC_ALPHA | Fonte ou Destino | (1, 1, 1, 1)-(As, As, As, As) |
| GL_DST_ALPHA | Fonte ou Destino | (Ad, Ad, Ad, Ad) |
| GL_ONE_MINUS_DST_ALPHA | Fonte ou Destino | (1, 1, 1, 1)-(Ad, Ad, Ad, Ad) |
| GL_SRC_ALPHA_SATURATE | Fonte | (f, f, f, 1); f=min(As, 1-Ad) |

Tabela 11. Faixa de valores para a função “Blend” - glBlendFunc

## 11.2 - Exemplo de Blending

```
/* Exemplo13.c - Marcionílio Barbosa Sobrinho
* Programa que apresenta uma variação do Exemplo8.c, com a funcionalidade
*          de aplicacao do efeito de Blending (Transparência)
* Referência do Código: OpenGL Programming Guide - RedBook
*                       planet.c, movelight.c, material.c
*/

#include <windows.h>
#include <GL/gl.h>
#include <GL/glut.h>

int posicaoluz = 0;
GLuint lista1;
int bBlend = 0;

void crialista(void);

void init(void)
{

   /* Cria as matrizes responsáveis pelo
      controle de luzes na cena */

   GLfloat ambiente[] = { 0.2, 0.2, 0.2, 1.0 };
   GLfloat difusa[] = { 0.7, 0.7, 0.7, 1.0 };
   GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };
   GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };
   GLfloat lmodelo_ambiente[] = { 0.2, 0.2, 0.2, 1.0 };


   glClearColor(0.0, 0.0, 0.0, 1.0);
   glEnable(GL_DEPTH_TEST);
   glShadeModel(GL_SMOOTH);

   /* Cria e configura a Luz para a cena */

   glLightfv(GL_LIGHT0, GL_AMBIENT, ambiente);
   glLightfv(GL_LIGHT0, GL_DIFFUSE, difusa);
   glLightfv(GL_LIGHT0, GL_POSITION, posicao);
   glLightfv(GL_LIGHT0, GL_SPECULAR, especular);
   glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodelo_ambiente);

   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);

   //Chama a funcao de criação dos Display Lists
   crialista();
}

/*
  Função responsável pelo desenho das esferas.
  Nesta função também serão aplicadas as tranformações
  necessárias para o efeito desejado dentro das "Display Lists" criadas.
*/

void crialista(void)
{
  /* Variáveis para definição da capacidade de brilho do material */
  GLfloat semespecular[4]={0.0,0.0,0.0,1.0};
  GLfloat especular[] = { 1.0, 1.0, 1.0, 1.0 };

  /* Posição da luz */
  GLfloat posicao[] = { 0.0, 3.0, 2.0, 0.0 };

  /*
    Limpa o buffer de pixels e
    determina a cor padrão dos objetos.
```

```
  */
  lista1 = glGenLists (1);

  glNewList (lista1, GL_COMPILE);

     glMaterialfv(GL_FRONT,GL_SPECULAR, semespecular);
        // Define a concentração do brilho
     glMateriali(GL_FRONT,GL_SHININESS,100);

     glPushMatrix();
      // glutSolidSphere(1.0, 30, 26);
      glRotatef (0.0, 0.0, 1.0, 0.0);
      glTranslatef (0, 0.0, 0.0);
      glRotatef (23, 1.0, 0, 0.0);

      glutSolidCube (1.5);
     glPopMatrix();

  glEndList();

}


void display (void )
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  // Executa os comandos



   // GLfloat mat_transparent[] = { 0.0, 0.8, 0.8, 0.6 };
    glColor4f (0.0, 0.8, 0.8, 0.6 );

    glPushMatrix();
       glTranslatef(0.0,2.0,-3.0);
       glCallList (lista1);
    glPopMatrix();

    glPushMatrix();
       glTranslatef(0.0,-2.0,-3.0);
       glCallList (lista1);
    glPopMatrix();

    glPushMatrix();
       glTranslatef(2.0,0.0,-3.0);
       glCallList (lista1);
    glPopMatrix();

    glPushMatrix();
       glTranslatef(-2.0,0.0,-3.0);
       glCallList (lista1);
    glPopMatrix();

    if (bBlend)
    {
      glEnable (GL_BLEND);
       glDepthMask (GL_FALSE);
       glBlendFunc (GL_SRC_ALPHA, GL_ONE);
    }
    glTranslatef(0.0,0.0,0.0);
    glColor4f (0.0, 0.8, 0.8, 0.6 );
    glutSolidSphere(1.5, 30, 26);
    glDepthMask (GL_TRUE);
    glDisable (GL_BLEND);



  glutSwapBuffers();
}

/*
   Função responsável pelo desenho da tela
   Nesta função são determinados o tipo de Projeção
```

```
   o modelo de Matrizes e
   a posição da câmera
   Quando a tela é redimensionada os valores
   da visão perspectiva são recalculados com base no novo tamanho da tela
   assim como o Viewport
*/

void reshape (int w, int h)
{

  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}

/* Função responsável pelo controle de teclado
   quando pressionada a tecla ESC o programa será encerrado.
   Pressionando-se a tecla B o efeito de Blending é habilitado
*/

void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
      case 27 :
           exit(0);
           break;
      case 'b' :
      case 'B' :
           bBlend = bBlend ? 0 : 1;
           glutPostRedisplay();
           break;
  }
}

/*
 Função principal do programa.
*/
int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Exemplo 13 - Blending");
  init ();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutKeyboardFunc(keyboard);

  glutMainLoop();
  return 0;
}
```

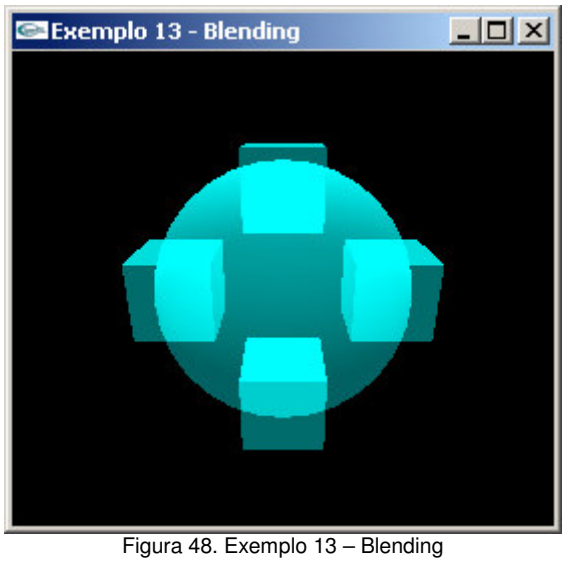


Figura 48. Exemplo 13 – Blending

# PARTE III - Conclusão

## Conclusão

O resultado final deste trabalho, a publicação do conteúdo do mesmo na Internet, tornou-se uma nova referência de consulta e utilização das funções de OpenGL.

Com a publicação inicial tendo sido feita em setembro de 2003, o número de acessos no website, alcançou em novembro de 2003 o número de 2978 acessos. Este número só vem a apresentar a carência e a necessidade deste tema.

Apesar de estar abordando uma grande gama de assuntos referentes à computação gráfica e ao OpenGL, a presente obra não abrange todos os tópicos que a API fornece. Assim a continuidade deste trabalho se faz necessária, para que toda a comunidade, tanto acadêmica quanto profissional possa estar munida de um referencial ainda mais poderoso em nossa língua de origem. Podendo este mesmo tema ser sugerido como trabalho futuro, de forma a abordar temas como : Sombras Volumétricas, Fog, Antialiasing, dentre outras.

Finalmente, como contribuição acadêmica e profissional, espera-se que este trabalho, como referência sobre OpenGL, possa despertar o interesse pela pesquisa em computação gráfica.

**Referências bibliográficas**


JAMSA, Kris. **Programando em C/C++ a Bíblia**;Ed. Makron Books;1999; SP

KILGARD, M. J.; **OpenGL and X, Part 1:An Introduction. Technical report**; SGI; 1994; Disponível em <http://www.sgi.com/software/opengl/glandx/intro/intro.html>.

MOLOFEE, Jeff. **Néon Helium Productions Open GL Tutorial**; Disponível em: <http://nehe.gamedev.net/>

SEGAL, M. ; AKELEY, K.; **The Design of the OpenGL Graphics Interface. Technical report**, Silicon Graphics Inc.;1997; Disponível em <http://www.opengl.org/developers/documentation/white_papers/opengl/index.html>.

SEGAL, M. ; AKELEY, K.; **The OpenGL Graphics Interface. Technical report;** Silicon Graphics Inc.;Disponível em <http://www.opengl.org/developers/documentation/white_papers/oglGraphSys/opengl.html>.

WESLEY, Addison Pub Co. **OpenGL Programming Guide – The official Guide to learn OPENGL VERSION 1.2 (RED BOOK)** ; 3º Edição, Ago/1999

WESLEY, Addison Pub Co. **OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.2 (Blue BOOK)**; 3º Edição; Dez/1999

WRIGHT, Richard. **OpenGL Super Bible – Second Edition** ; Editora: Waite Group Press, 2000

MASON, WOO; SHREINER, DAVE; ANGEL, ED; **An Interactive Introduction to OpenGL Programming,** SIGGRAPH 1999; Los Angeles

Centro Universitário de Belo Horizonte

Departamento de Ciências Exatas e Tecnologia

Termo de Responsabilidade

O texto da monografia intitulada **TUTORIAL DE UTILIZAÇÃO DE OPENGL** é de minha inteira responsabilidade.

Belo Horizonte, 11 de dezembro de 2003

_____

Marcionílio Barbosa Sobrinho