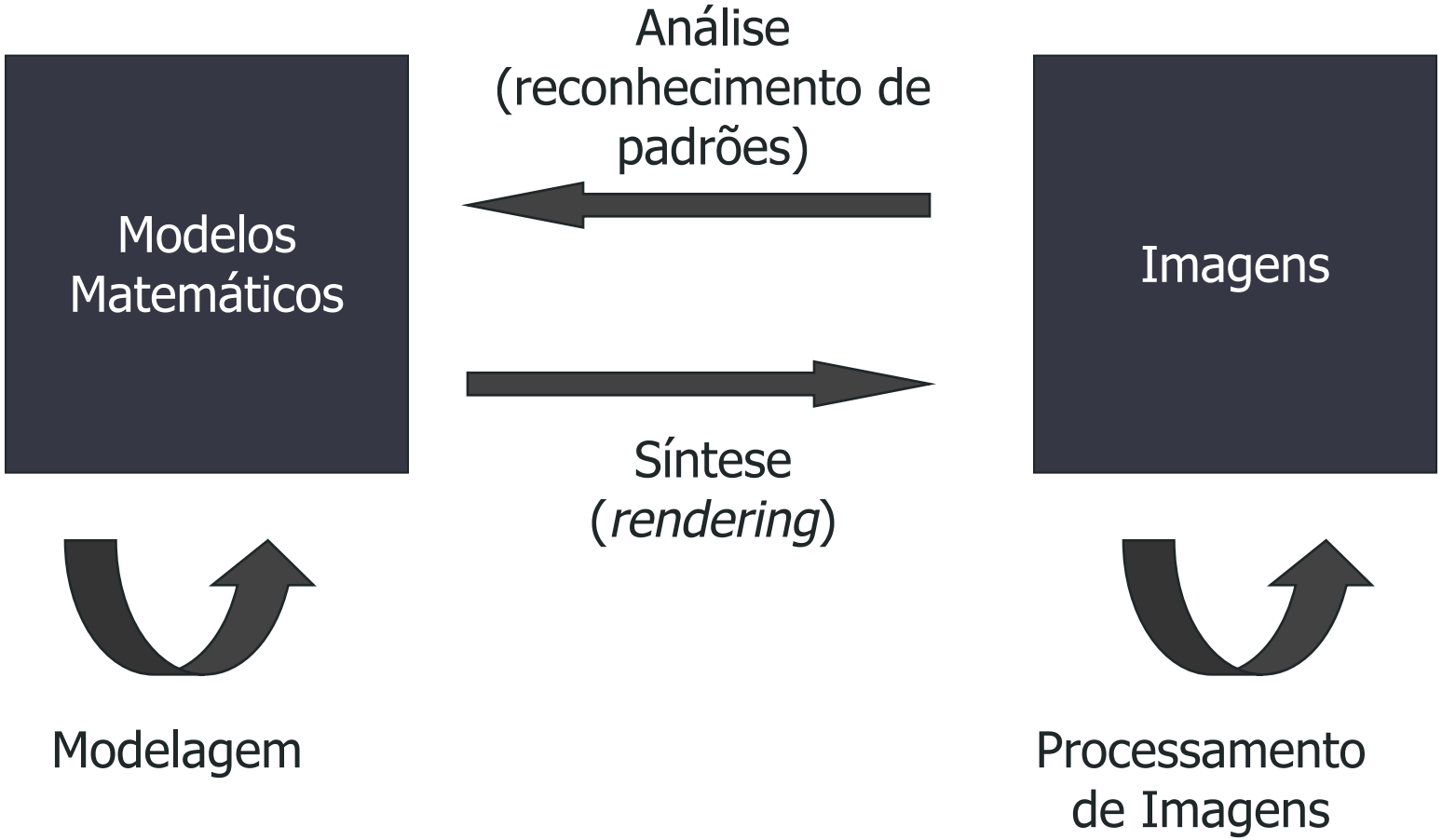
Plano de Ensino e Introdução

COMPUTAÇÃO GRÁFICA

A disciplina



Disciplinas Relacionadas

- Computação
  - Algoritmos
  - Estruturas de Dados
  - Métodos Numéricos
- Matemática
  - Geometria
  - Álgebra Linear
- Física
  - Ótica
  - Mecânica
- Psicologia
  - Percepção
- Artes

Aplicações

- Desenho Assistido por Computador (CAD)
- Sistemas de Informações Geográficas (GIS)
- Visualização Científica
- Visualização Médica
- Educação
- Entretenimento

Plano de Ensino - Representações

- Gráficos Vetoriais - Representados por coleções de objetos geométricos:
  - Pontos
  - Retas
  - Planos
  - Polígonos

Plano de Ensino - Representações

- Gráficos Matriciais - Amostragem em grades retangulares:
  - Imagens Digitais:
    - Matrizes de pixels
    - Cada pixel representa uma cor
  - Dados Volumétricos:
    - Imagens médicas
    - Cada pixel representa densidade ou intensidade de algum campo

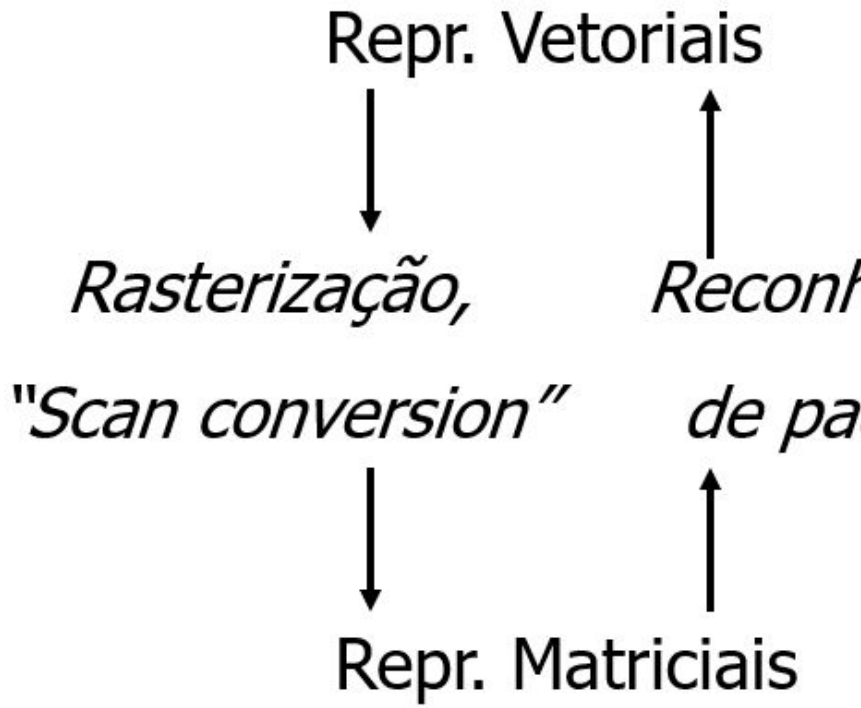
Plano de Ensino - Representações Vetoriais

- Permitem uma série de operações sem (quase) perda de precisão
  - Transformações lineares
  - Deformações
- Por que “quase”? Estruturas de dados utilizam pontos e vetores cujas coordenadas são números reais
  - É necessário usar aproximações
  - Representação em ponto flutuante
  - Números racionais
- Exibição em dispositivos matriciais (requer amostragem, rasterização)

Plano de Ensino - Representações Matriciais

- Representação flexível e muito comum
- Complexidade de processamento linear (numéro de pixels)
- Muitas operações implicam em perda de precisão (reamostragem)
  - Ex.: rotação, escala
- Técnicas para lidar com o problema
  - Ex.: técnicas anti-serrilhado (anti-aliasing)
- Exibição em dispositivos vetoriais (requer uso de técnicas de reconhecimento de padrões)

Plano de Ensino - Conversão de Representações



Plano de Ensino - Conteúdo

- Conceitos Básicos
- Operações Básicas
- Representação 3D

Plano de Ensino - Conteúdo

- Matrizes e Modelos Hierárquicos
- Arquivos .Obj
- Iluminação
- Normais e Texturas

Plano de Ensino - Conteúdo

- Pipeline Gráfico
- Shaders
- Normal Maps
- Voxels

Aulas

- Ênfase na parte prática
- Avaliação através de trabalhos de implementação
  - C++
  - OpenGL com GLUT
- Pesquisas sobre assuntos que vão além da prática
- Trabalho final sobre artigos publicados na área

Avaliações

- M1:
  - Trabalho 1.1 - Operações Básicas - Peso 5
  - Trabalho 1.2 - Obj e Matrizes - Peso 5
- M2:
  - Trabalho 2.1 - Iluminação - Peso 5
  - Trabalho 2.2 - Shaders - Peso 5
- M3:
  - Pesquisa M3 - Ray Tracing - Peso 3
  - Seminário M3 - Tópicos Especiais Peso 7

OpenGL

- OpenGL é uma interface de software para dispositivos de hardware. É uma biblioteca gráfica de modelagem e exibição tridimensional, bastante rápida e portável para vários sistemas operacionais. Seus recursos permitem ao usuário criar objetos gráficos, além de incluir recursos avançados de animação, tratamento de imagens e texturas.

OpenGL

- A biblioteca OpenGL (Open Graphics Library) foi introduzida em 1992 pela Silicon Graphics (http://www.sgi.com/), no intuito de conceber uma API (Interface de Programação de Aplicação) gráfica independente de dispositivos de exibição. Com isto, seria estabelecida uma ponte entre o processo de modelagem geométrica de objetos, situadas em um nível de abstração mais elevado, e as rotinas de exibição e de processamento de imagens implementadas em dispositivos (hardware) e sistemas operacionais específicos. A função utilizada pelo OpenGL para desenhar um ponto na tela, por exemplo, possui o mesmo nome e parâmetros em todos os sistemas operacionais nos quais OpenGL foi implementada, e produz o mesmo efeito de exibição em cada um destes sistemas.

OpenGL

- Todas as rotinas do OpenGL são implementadas em C, tornando fácil sua utilização em qualquer programa escrito em C ou C++.
- Entre os recursos gráficos disponíveis pelo OpenGL, podem ser destacados os seguintes:
    - Modos de desenho de pontos;
    - Ativação/desativação de serrilhamento (aliasing);
    - Mapeamento de superfícies com textura;
    - Seleção de janela de desenho;
    - Manipulação de fontes/tipos de iluminação e sombreamento;
    - Transformação de sistemas de coordenadas.
    - Transformações em perspectiva

Glut

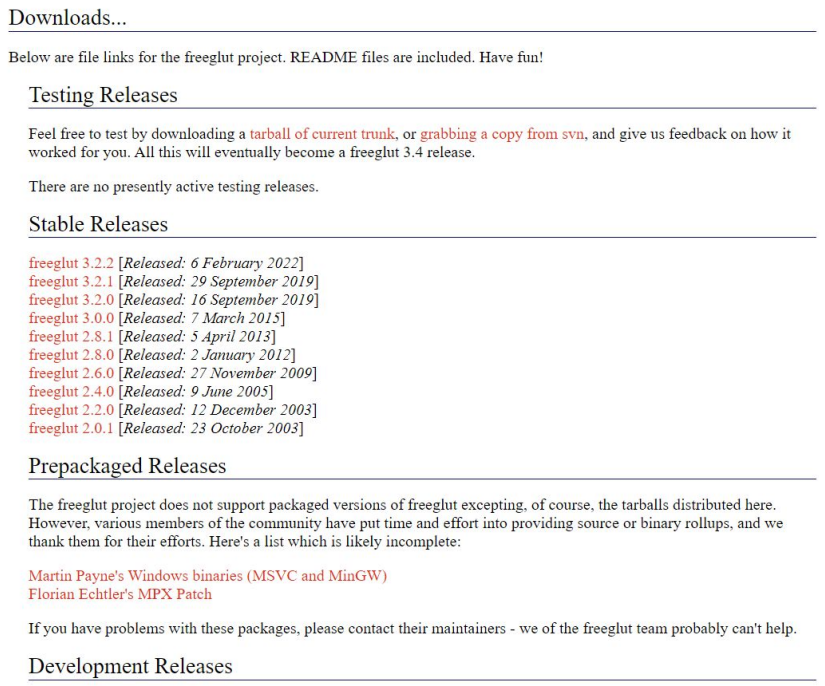
- GLUT (OpenGL Utility Toolkit) é uma biblioteca de funcionalidades para OpenGL cujo principal objetivo é a abstração do sistema operacional fazendo com que os aplicativos sejam multiplataforma. A biblioteca possui funcionalidades para criação e controle de janelas, e também tratamento de eventos de dispositivos de entrada (mouse e teclado). Também existem rotinas para o desenho de formas tridimensionais pré-definidas (como cubo, esfera, bule, etc).

Freeglut

- Vamos utilizar oficialmente o freeglut, o GLUT é uma simplificação da biblioteca OpenGL para o gerenciamento de janelas e afins ser independente de sistema operacional, porém o projeto original está parado. O freeglut é uma proposta open source de manter esse projeto vivo.

Freeglut

- O primeiro passo é acessar:
  - https://freeglut.sourceforge.net/index.php#download



Downloads...

Below are file links for the freeglut project. README files are included. Have fun!

Testing Releases

Feel free to test by downloading a tarball of current trunk, or grabbing a copy from svn, and give us feedback on how it worked for you. All this will eventually become a freeglut 3.4 release.

There are no presently active testing releases.

Stable Releases

freeglut 3.2.2 [*Released: 6 February 2022*]
freeglut 3.2.1 [*Released: 29 September 2019*]
freeglut 3.2.0 [*Released: 16 September 2019*]
freeglut 3.0.0 [*Released: 7 March 2015*]
freeglut 2.8.1 [*Released: 5 April 2013*]
freeglut 2.8.0 [*Released: 2 January 2012*]
freeglut 2.6.0 [*Released: 27 November 2009*]
freeglut 2.4.0 [*Released: 9 June 2005*]
freeglut 2.2.0 [*Released: 12 December 2003*]
freeglut 2.0.1 [*Released: 23 October 2003*]

Prepackaged Releases

The freeglut project does not support packaged versions of freeglut excepting, of course, the tarballs distributed here. However, various members of the community have put time and effort into providing source or binary rollups, and we thank them for their efforts. Here's a list which is likely incomplete:

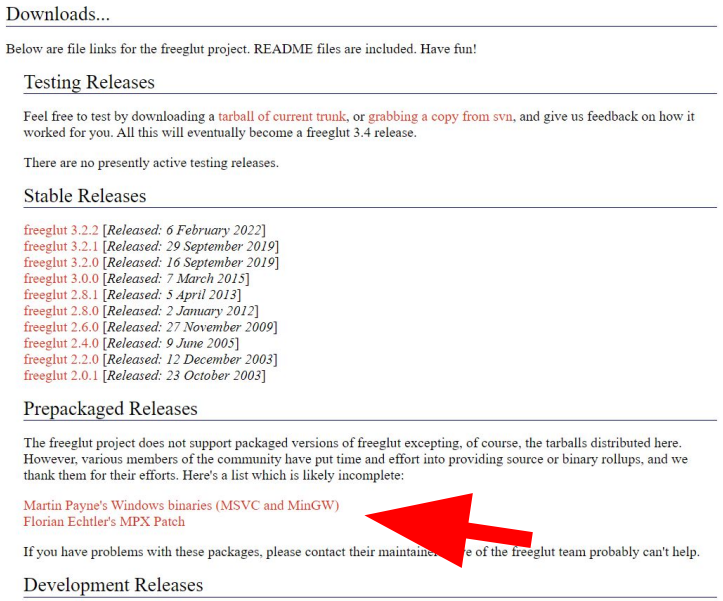
Martin Payne's Windows binaries (MSVC and MinGW)
Florian Echtler's MPX Patch

If you have problems with these packages, please contact their maintainers - we of the freeglut team probably can't help.

Development Releases

Freeglut

- Como estamos fazendo um projeto no Visual Studio, utilizaremos os binários para Windows do compilador MSVC:
  - https://www.transmissionzero.co.uk/software/freeglut-devel/



Downloads...

Below are file links for the freeglut project. README files are included. Have fun!

Testing Releases

Feel free to test by downloading a tarball of current trunk, or grabbing a copy from svn, and give us feedback on how it worked for you. All this will eventually become a freeglut 3.4 release.

There are no presently active testing releases.

Stable Releases

freeglut 3.2.2 [*Released: 6 February 2022*]
freeglut 3.2.1 [*Released: 29 September 2019*]
freeglut 3.2.0 [*Released: 16 September 2019*]
freeglut 3.0.0 [*Released: 7 March 2015*]
freeglut 2.8.1 [*Released: 5 April 2013*]
freeglut 2.8.0 [*Released: 2 January 2012*]
freeglut 2.6.0 [*Released: 27 November 2009*]
freeglut 2.4.0 [*Released: 9 June 2005*]
freeglut 2.2.0 [*Released: 12 December 2003*]
freeglut 2.0.1 [*Released: 23 October 2003*]

Prepackaged Releases

The freeglut project does not support packaged versions of freeglut excepting, of course, the tarballs distributed here. However, various members of the community have put time and effort into providing source or binary rollups, and we thank them for their efforts. Here's a list which is likely incomplete:

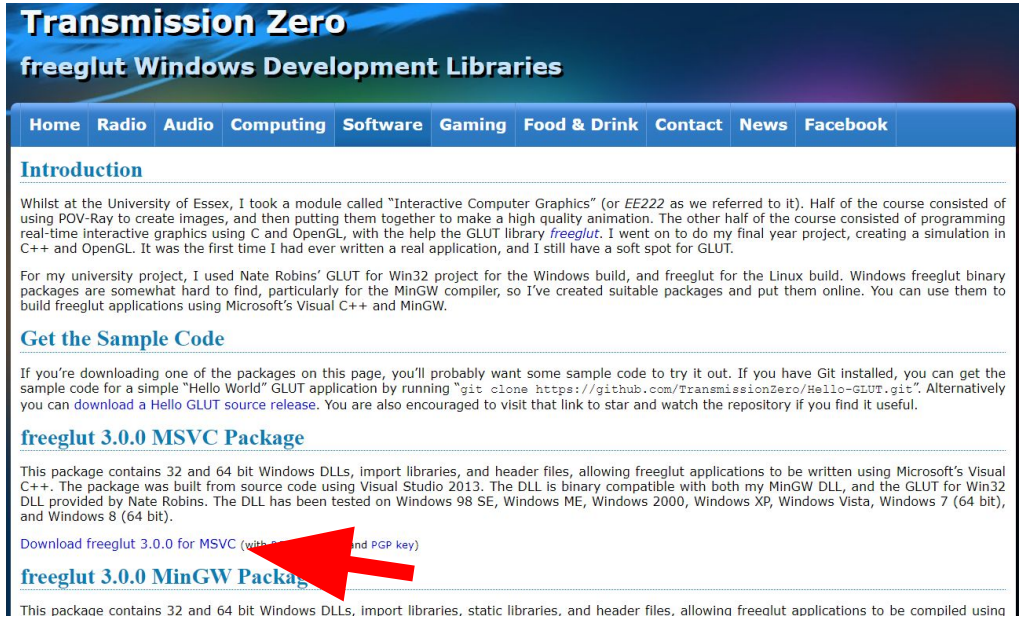
Martin Payne's Windows binaries (MSVC and MinGW)
Florian Echtler's MPX Patch

If you have problems with these packages, please contact their maintainer[illegible]e of the freeglut team probably can't help.

Development Releases

Freeglut

- Dentro da página, acessamos os binários específicos para MSVC:
  - https://www.transmissionzero.co.uk/files/software/development/GLUT/freeglut-MSVC.zip
- Esse link faz download de um .zip que pode ser descompactado, dentro dele temos a pasta freeglut



Criando um Projeto

- Agora vamos fazer a parte do Visual Studio, o primeiro passo é criar um projeto em C++ como Console Application, para isso é necessário adicionar as dependências do C++ no Visual Studio, selecionando elas na instalação ou modificando depois de instalado pelo Installer.
- Os computadores da UNIVALI já vêm com as dependências instaladas.
- Utilizem o Visual Studio 2019 ou superior para facilitar na criação do projeto.

Criando um Projeto

- Agora abram a pasta do projeto, ela pode ser acessada pelo Visual Studio com o botão direito no projeto e “Abrir Pasta no Gerenciador de Arquivos”.
- Dentro dessa pasta coloque a pasta freeglut que estava no .zip do download.

Criando um Projeto

- Agora vamos adicionar essa dependência ao projeto, acessando as propriedades (pelo botão direito):
- Primeiro na aba C/C++ -> Geral, em Diretórios de Inclusões Adicionais acrescente:
    - $(ProjectDir)freeglut\include
- Depois na aba Vinculador -> Geral, em Diretórios de Bibliotecas Adicionais acrescente:
    - $(ProjectDir)freeglut\lib\x64

Criando um Projeto

- Depois, na aba Vinculador -> Entrada, em Dependências Adicionais acrescente:
    - freeglut.lib

- $(ProjectDir) é uma variável que aponta para a pasta do projeto atual, assim podemos modificar a pasta do projeto sem precisar modificar o caminho.

Criando um Projeto

- Por último, precisamos fazer uma última ação, ou copiar manualmente a pasta freeglut/bin/x64 para o projeto, ou colocar um comando para que essa cópia aconteça depois da compilação para não mantermos 2 cópias da dll o tempo todo.
- Para isso, na aba Compilar Eventos -> Evento de Pós-Compilação, em linha de comando acresente:
  - copy $(ProjectDir)freeglut\bin\x64\freeglut.dll
- Pronto, agora podemos trabalhar com o OpenGL

Programa Teste

- Baixem o arquivo C01 - Exemplo.cpp no material didático e copiem o conteúdo para o main.cpp do projeto.

Programa Teste

- Define o tamanho inicial da janela, 256x256 pixels, e a posição inicial do seu canto superior esquerdo na tela, (x, y)=(100, 100). Esses comandos são opcionais.

```
glutInitWindowSize (256, 256);
glutInitWindowPosition (100, 100);
```

Programa Teste

- Cria uma janela e define seu título como "Desenhando uma linha".

```
glutCreateWindow ("Desenhando uma linha");
```

Programa Teste

- Define display() como a função de desenho (display callback) para a janela corrente. Quando GLUT determina que esta janela deve ser redesenhada, a função de desenho é chamada.
- A função de desenho deve possuir o seguinte protótipo:
  - void function(void);

```
glutDisplayFunc(display);
```

Programa Teste

- Indica que sempre que uma tecla for pressionada no teclado, GLUT deverá chama a função keyboard() para tratar eventos de teclado (keyboard callback).
- A função de teclado deve possuir o seguinte protótipo:void
  - function(unsigned char key, int x, int y);

```
glutKeyboardFunc(keyboard);
```

Programa Teste

- Inicia o loop de processamento de desenhos com GLUT. Esta rotina deve ser chamada pelo menos uma vez em um programa que utilize a biblioteca GLUT.

```
glutMainLoop();
```

Programa Teste

- Especifica as intensidade de vermelho (RED), verde (GREEN) e azul (BLUE) utilizadas para limpar a janela. Cada parâmetro pode varia de 0 a 1, o equivalente a uma variação de 0 a 255, usada convencionalmente no sistema de janelas. O último argumento é o canal alfa, usado para compor superfícies transparentes. Como estes conceitos ainda não foram apresentados, o canal alfa deve ser mantido com valor igual a 1.

```
glClearColor(1.0, 1.0, 1.0, 1.0);
```

Programa Teste

- A função glOrtho define as coordenadas do volume de recorte (clipping volume), possuindo o seguinte protótipo:void glOrtho()(GLdouble left , GLdouble right , GLdouble bottom , GLdouble top , GLdouble zNear , GLdouble zFar);

```
glOrtho (0, 256, 0, 256, -1 ,1);
```

Programa Teste

- A função glClear() serve para limpar buffers utilizados pelo OpenGL com valores pré-definidos. A máscara utilizada neste exemplo, (GL_COLOR_BUFFER_BIT, diz à função glClear() que apenas o buffer de desenho deve ser limpo. Após a execução desta função, a tela ficará branca, uma vez que a init() define (R, G, B)=(1.0, 1.0, 1.0) como cor de limpeza de tela.

```
glClear(GL_COLOR_BUFFER_BIT);
```

Programa Teste

- Especifica (R, G, B)=(0, 0, 0), preto, como a cor de desenho. Todos os objetos desenhados a partir daqui terão cor preta.

```
glColor3f (0.0, 0.0, 0.0);
```

Programa Teste

- As funções glBegin() e glEnd() delimitam os vértices de uma primitiva de desenho ou de um grupo de primitivas. O parâmetro passado para a função especifica o tipo de primitiva a ser desenhado. Neste exemplo, o parâmetro GL_LINES indica que os vértices especificados devem ser tratados como pares de pontos que comporão segmentos de reta independentes. A função glVertex2i() define as coordenadas de um vértice.

```
glBegin(GL_LINES);
glVertex2i(40,200);
glVertex2i(200,10);
glEnd();
```

Programa Teste

- Aqui a função de callback é chamada quando uma tecla é pressionada, nesse casa ele finaliza o programa quando a tecla 27(esc) é pressionada.

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