Professora: **Mercedes Gonzales Márquez**

Preliminares

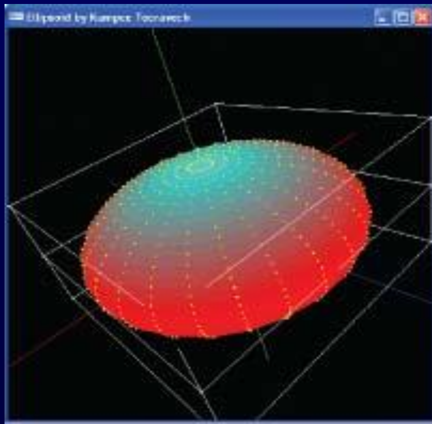
- OpenGL é uma API para criar aplicações interativas que renderizam imagens de alta qualidade compostas de objetos tridimensionais e imagens..
- OpenGL é independente do sistema operacional e do sistema de janelas.

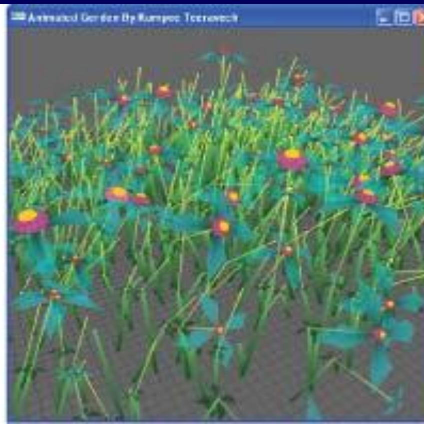
OpenGL

- Alguns programas executáveis em windows mostrando as potencialidades do OpenGL.

http://www.comp.uems.br/~mercedes/disciplinas/2022/CG/SUMANTA GUHA/

- Chapter1/Ellipsoid
- Chapter1/AnimatedGarden
- Chapter1/Dominos





Primeiro Programa

- Usando ambiente Windows, no Dev-C++ é necessário **criar um projeto**.
- Faça download da pasta http://www.comp.uems.br/~mercedes/disciplinas/2022/CG/SUMANTA GUHA/CodigosExperimentos.
- Ao longo das aulas usaremos vários dos programas dessa pasta para exemplificar nosso conteúdo teórico.

Primeiro Programa

- Usando ambiente LINUX direto no terminal.

Instalar a glut:

>sudo apt-get install freeglut3-dev

Gerar executável:

g++ -o nome-progr nome-progr.cpp –lglut –lGL –lGLU -lm

Bibliotecas GLU e GLUT

- GLU (OpenGL Utility Library)
  - parte de OpenGL
  - Simplifica tarefas como renderização de superfícies quádricas como esferas, cones, cilindros, etc.
  - #include <GL/glu.h>
- GLUT (OpenGL Utility Toolkit)
  - Simplifica o processo de criar janelas, trabalhar com eventos no sistema de janelas e manejar animações.
  - Não é oficialmente parte de OpenGL
  - #include <GL/glut.h>

GLUT é orientado a eventos

- GLUT proporciona uma "camada" de software que possibilita o acesso aos recursos básicos de hardware necessários como criação de janelas e interação simples com dispositivos de entrada e saída como o mouse e teclado, enquanto que o programa em OpenGL, que constitui a parte maior do código, fica "livre" de dependências de plataforma.

GLUT é orientado a eventos

- O paradigma de **programação** em **GLUT** é **orientado a eventos** do tipo: o mouse foi clicado; uma tecla foi pressionada, etc. O programador apenas define quais rotinas (callbacks) devem ser chamadas quando um determinado **evento** acontecer e **GLUT** gerencia estes **eventos** num laço de controle infinito.

GLUT é orientado a eventos

- void main(int argc, char **argv) {

    - inicializar o GLUT e a janela

    - registro de funções para processar eventos (callbacks)

    - entrar no ciclo de processamento do GLUT

  }

Estrutura da aplicação OpenGL/Glut

- Configure e inicialize a janela
- Inicialize os estados do OpenGL. Isto pode incluir coisas como cor do fundo, posições de luz e mapas de textura.
- Registre funções callbacks : rotinas que GLUT chama quando uma certa sequencia de eventos ocorre, por exemplo: quando a janela precisa ser reexibida, ou o usuário movimenta o mouse.
- Entre no loop de processamento de eventos.

Funções Callback GLUT

- Callbacks da GLUT: Rotinas que são chamadas quando algo como o seguinte acontece
  - Renderização (display) e animação
  - Redimensionamento ou redesenho da janela
  - Entrada do usuário
- Funções callbacks GLUT
    - **`glutDisplayFunc(` *`display`* `)`** estabelece a função "display" previamente definida como a função callback de exibição. Isto significa que a GLUT chama a função sempre que a janela precisar ser redesenhada. Esta chamada ocorre, por exemplo, quando a janela é redimensionada ou encoberta. É nesta função que se deve colocar as chamadas de funções OpenGL, por exemplo, para modelar e exibir um objeto.

Funções Callback GLUT

- Outras
  - `glutReshapeFunc()` – chamada quando a janela muda de tamanho
  - `glutKeyboardFunc()` – chamada quando uma tecla é pressionada no teclado
  - `glutMouseFunc()` – chamada quando o usuário pressiona um botão do mouse
  - `glutMotionFunc()` – chamada quando o usuário movimenta o mouse enquanto um botão do mesmo está pressionado.
  - `glutPassiveMouseFunc()` – chamada quando o mouse é movimentado sem considerar o estado dos botões.
  - `glutIdleFunc()` – chamada quando nada está sendo realizado. É muito útil para animações.

Programa exemplo

Programa square.c (cria um quadrado preto sobre fundo branco)

```
int main(int argc, char **argv) {
   // Configura e inicializa a GLUT e a janela
   glutInit(&argc, argv);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("square.cpp");
   // Inicialização de estados
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   setup();
   // Registra rotinas callbacks
   // Rotina display
   glutDisplayFunc(drawScene);
   // Rotina reshape.
   glutReshapeFunc(resize);
   // Rotina de teclado.
   glutKeyboardFunc(keyInput);
   // Loop de processamento de eventos
   glutMainLoop();
   return 0;
}
```

Inicialização de OpenGL

- Fixa alguns estados

```
void setup(void) {
   // Set background (or clearing) color.
   glClearColor(1.0, 1.0, 1.0, 0.0);
}
```

Callback de renderização (display)

- O desenho que será apresentado na tela é feito aqui.

```
glutDisplayFunc( display );
```

```
void drawScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(0.0, 0.0, 0.0);
  glBegin(GL_POLYGON);
      glVertex3f(20.0, 20.0, 0.0);
      glVertex3f(80.0, 20.0, 0.0);
      glVertex3f(80.0, 80.0, 0.0);
      glVertex3f(20.0, 80.0, 0.0);
   glEnd();
   glFlush();
}
```

Callbacks de entradas

- Processa uma entrada do usuário, pode ser por teclado, mouse.

```
glutKeyboardFunc( keyboard );
```

```
void keyInput(unsigned char key,int x,int y)
{
   switch(key)
   {
      case 27: // Press escape to exit
         exit(0);
         break;
      default:
         break;
   }
```

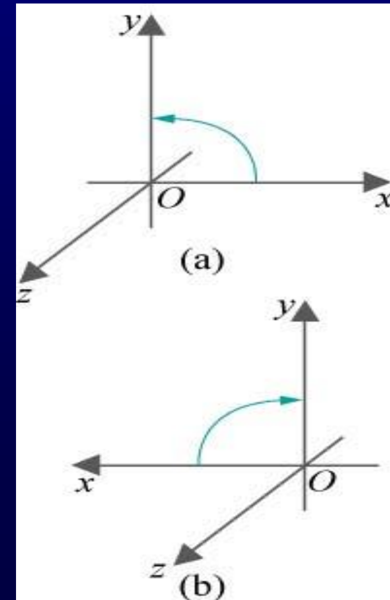
Sistemas coordenados

- Os vértices são especificados no espaço tridimensional.
- OpenGL permite desenhar no espaço 3D e criar cenas realmente tridimensionais. Porém, nos percebemos a cena 3D como uma imagem processada para uma parte 2D da tela do computador, a janela retangular OpenGL.
- O sistema coordenado 3D é o sistema mão direita.

No desenho ao lado

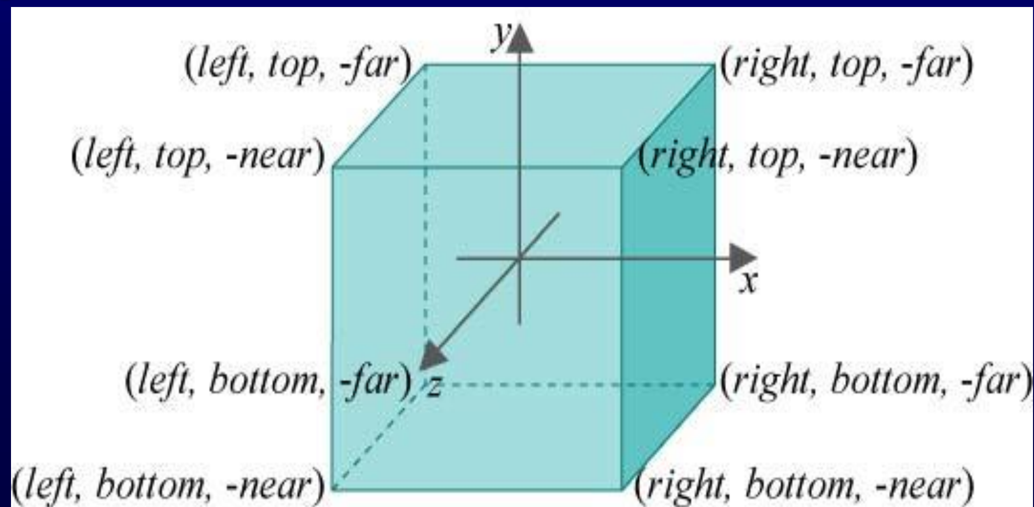
(a) sistema mão direita

(b) sistema mão esquerda



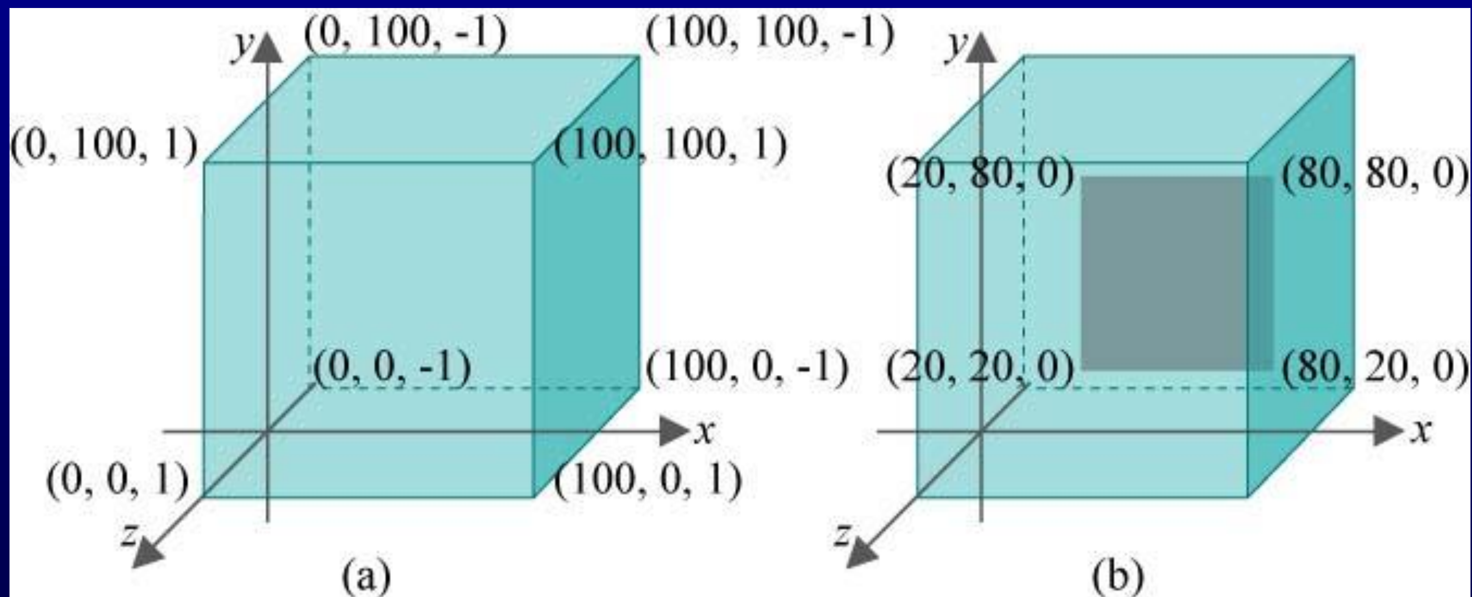
Projeção Ortográfica, Volume de Visualização

- O comando glOrtho (left, right, bottom, top, near, far)
  - especifica o volume de visualização (vi) onde a cena 3D deverá estar contida,
  - projeta perpendicularmente sobre a face da frente do vi (face sobre o plano z=-near)
  - A projeção é proporcionalmente escalonada para ajustar a janela OpenGL.
- Volume de visualização



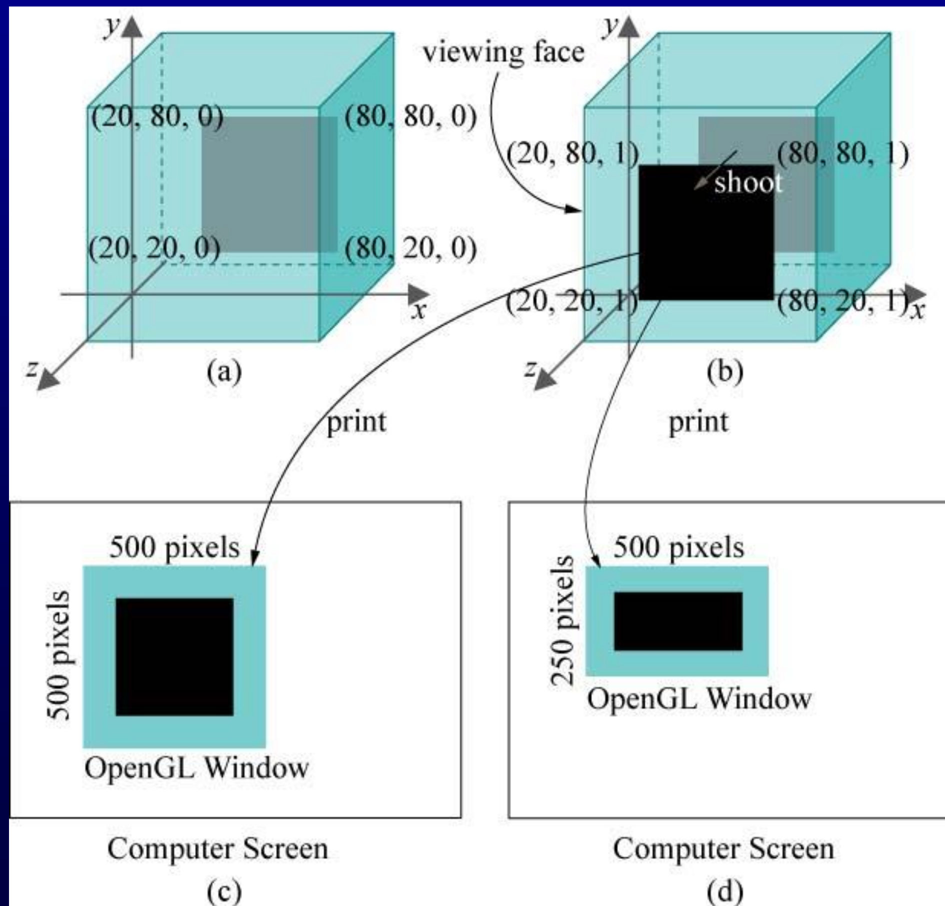
Projeção Ortográfica, Volume de Visualização

- (a) Volume de visualização do programa square.c
- (b) quadrado dentro do vi



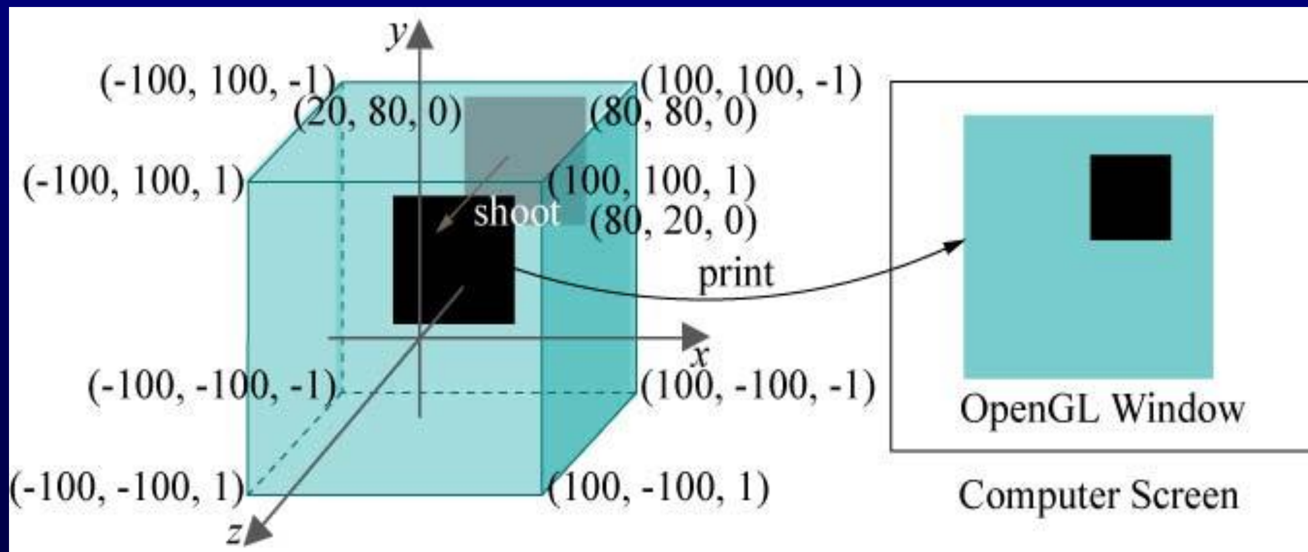
Projeção Ortográfica, Volume de Visualização

- (a) glutInitWindowSize (500,500)
- (b) glutInitWindowSize (500,250) (distorse o quadrado a um retângulo)



Projeção Ortográfica, Volume de Visualização

- Experimento 2.3: Mude o vi fazendo glOrtho(-100.0,100.0,-100.0,100.0,-1.0,1.0), perceba que a localização do quadrado no novo vi é diferente e portanto o resultado da projeção também.



Projeção Ortográfica, Volume de Visualização

- Mude para

(a) glOrtho (0.0,200.0,0.0,200.0,-1.0,1.0)

(b) glOrtho (20.0,80.0,20.0,80.0,-1.0,1.0)

(c) glOrtho (0.0,100.0,0.0,100.0,-2.0,5.0), em todos os casos tente prever o resultado.

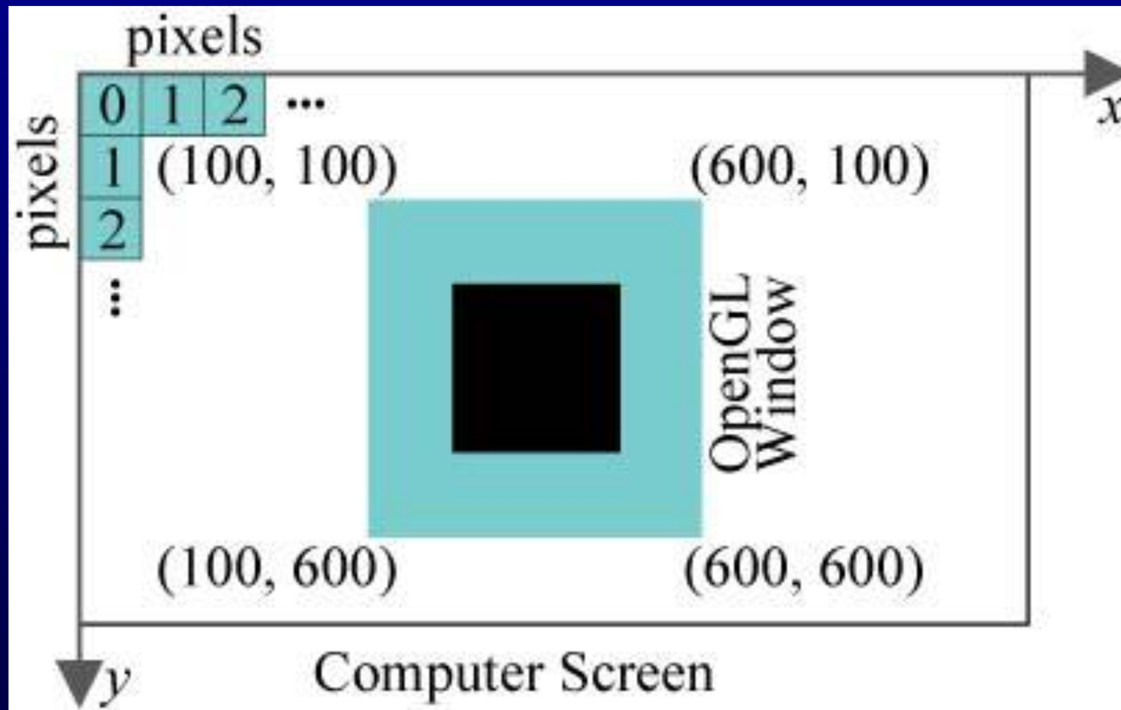
- Altere para

```
    glBegin(GL_POLYGON)
  glVertex3f(20.0,20.0,0.5);
  glVertex3f(80.0,20.0,-0.5);
  glVertex3f(80.0,80.0,0.1);
  glVertex3f(20.0,80.0,0.2);
GlEnd();
```

o resultado muda?

Janela OpenGL

- Mude os parâmetros de glutInitWindowPosition(x,y)



Recorte

- Experimento 2.5: Adicione um outro quadrado

```
glBegin(GL_POLYGON)
  glVertex3f(120.0,120.0,0.0);
  glVertex3f(180.0,120.0,0.0);
  glVertex3f(180.0,180.0,0.0);
  glVertex3f(120.0,180.0,0.0);
glEnd();
```

Ele é visível ou não? Como pode você deixá-lo visível?

Recorte

Experimento 2.6: Substitua agora o quadrado por um triângulo,assim:

```
glBegin(GL_POLYGON)
        glVertex3f(20.0,20.0,0.0);
        glVertex3f(80.0,20.0,0.0);
        glVertex3f(80.0,80.0,0.0);
    glEnd();
```

Então puxe a coordenada z do primeiro vértice mudando-a

(a) glVertex(20.0,20.0,0.5)

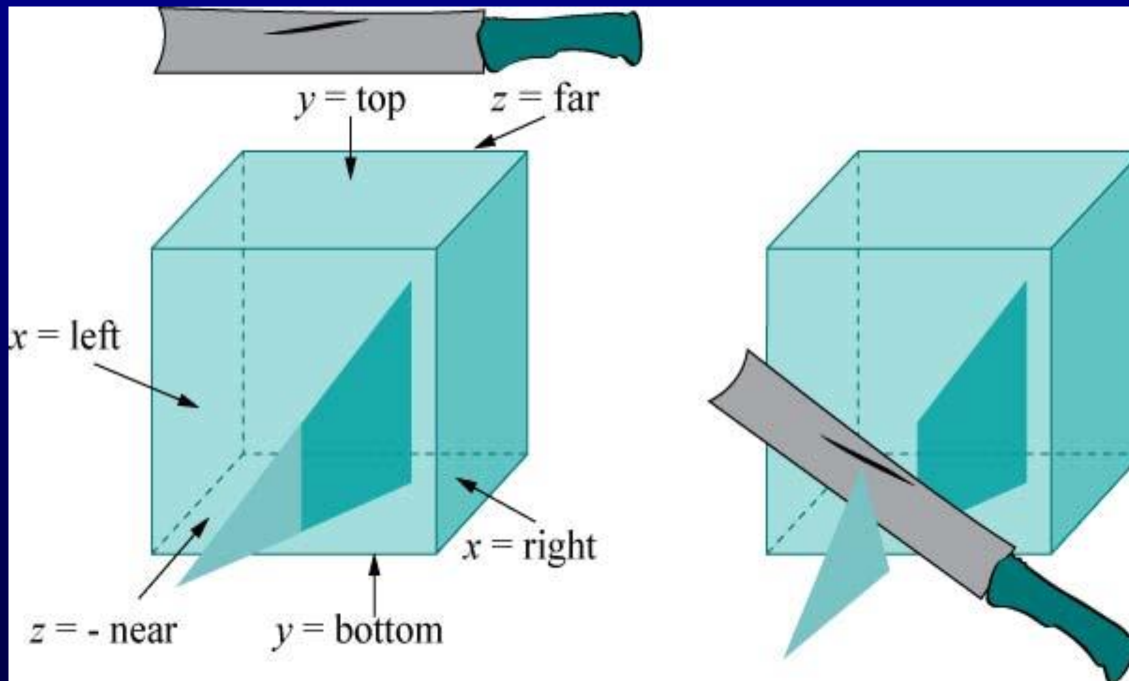
(b) glVertex(20.0,20.0,1.5)

(c) glVertex(20.0,20.0,2.5)

(d) glVertex(20.0,20.0,10.0)

Recorte

- Veja o efeito do recorte



Recorte

- Exercício: Use papel e lapis para deduzir a saída se o trecho de construção do polígono for substituído por

```
glBegin(GL_POLYGON)
        glVertex3f(-20.0,-20.0,0.0);
        glVertex3f(80.0,20.0,0.0);
        glVertex3f(120.0,120.0,0.0);
        glVertex3f(20.0,80.0,0.0);
glEnd();
```

Cor

- A cor é especificada pelos três parâmetros do comando glColor3f(0.0,0.0,0.0) na rotina drawScene(). Cada um deles fornece o valor de uma das componentes primárias: azul, verde e vermelho. Veja a seguinte tabela:

(0.0,0.0,0.0) – Preto

(1.0,0.0,0.0) – Vermelho

(0.0,1.0,0.0) – Verde

(0.0,0.0,1.0) – Azul

(1.0,1.0,0.0) – Amarelo

(1.0,0.0,1.0) – Magenta

(0.0,1.0,1.0) – Ciano

(1.0,1.0,1.0) - Branco

glColor3f(1.0,1.0,1.0)
Equivale
glColor3i(255,255,255)

glColor3f(0.5,0.5,0.5)
Equivale
glColor3i(128,128,128)

Cor

- Geralmente, glColor3f(red,green,blue) especifica a cor do primeiro plano, ou a cor do desenho. O valor de cada componente de cor (que deve estar entre 0 e 1) determina sua intensidade. Por exemplo, glColor3f(1.0,1.0,0.0) é um amarelo mais brilhante do que glColor3f(0.5,0.5,0.0) que é um amarelo mais fraco.
- Exercício: Ambos glColor3f(0.2,0.2,0.2) e glColor3f(0.8,0.8,0.8) são cinzas, tendo intensidades iguais vermelho, verde e azul. Conjecture qual é o mais escuro dos dois. Verifique mudando a cor de primeiro plano de square.c.
- O comando glClearColor (1.0,1.0,1.0,0.0) na rotina setup() especifica a cor do fundo, o cor de limpeza. No momento devemos ignorar o 4o parâmetro. O comando glClear(GL_COLOR_BUFFER_BIT) em drawScene() realmente limpa a janela com a cor de fundo especificada, ou seja cada pixel no buffer de cor é setado a aquela cor.

Máquina de estados

- Experimento 2.8: Adicione o comando glColor3f(1.0,0.0,0.0) depois do já existente comando glColor3f(0.0,0.0,0.0) na rotina de desenho de square.c tal que a cor do primeiro plano mude.
- O quadrado é desenhado em vermelho pois o valor corrente da cor de primeiro plano (ou cor do desenho) é vermelha quando cada um dos seus vértices são especificados.
- Cor de desenho pertence a uma coleção de variáveis, chamadas variáveis de estado, as quais determinam o estado de OpenGL. Outras variáveis de estado são: tamanho de ponto, espessura da linha, pontilhado da linha, propriedades de material da superfície, etc. OpenGL permanece e funciona no seu estado corrente até que uma declaração for feita mudando a variável de estado.

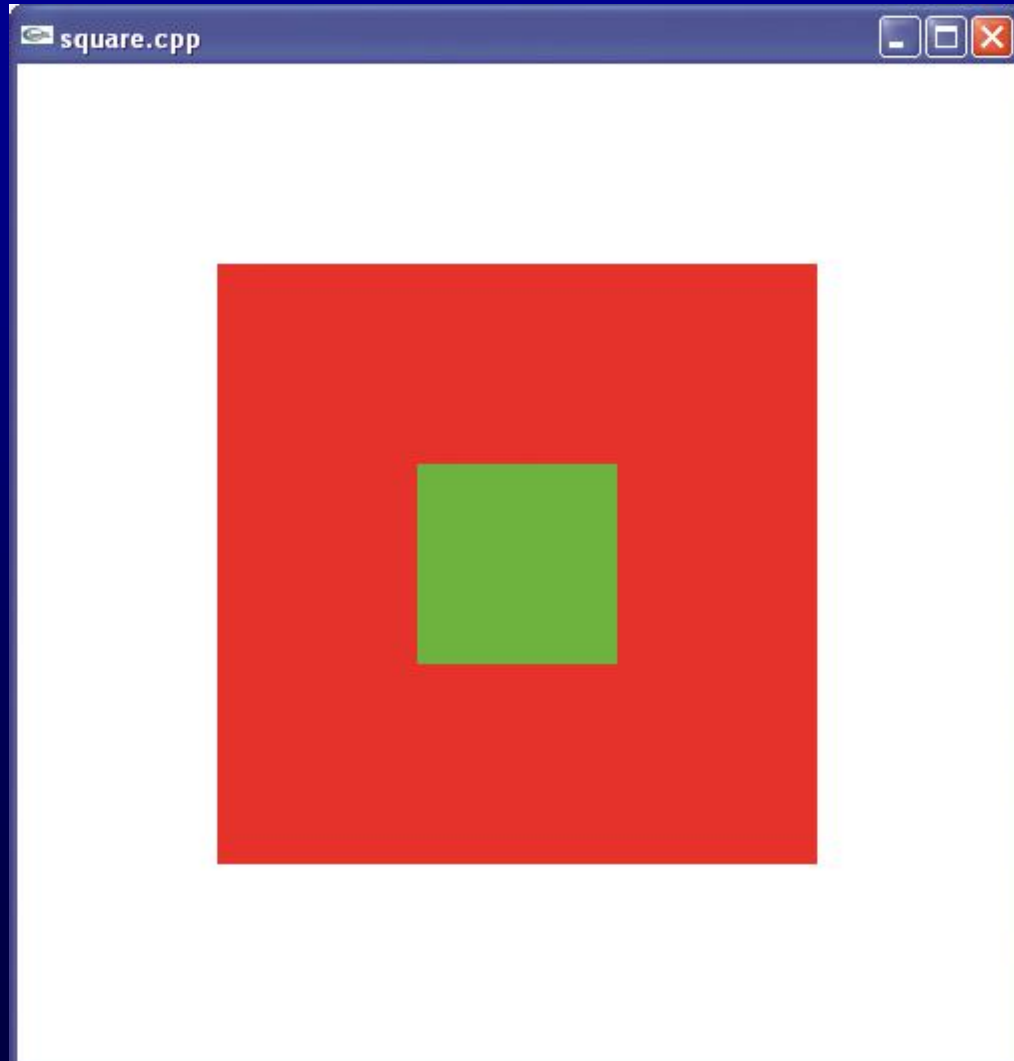
Máquina de estados

- Experimento 2.9: Substitua a parte de desenho do polígono de square.c com a seguinte que desenha dois quadrados.

```
glColor3f(1.0,0.0,0.0);
glBegin(GL_POLYGON)
    glVertex3f(20.0,20.0,0.0);
    glVertex3f(80.0,20.0,0.0);
    glVertex3f(80.0,80.0,0.0);
    glVertex3f(20.0,80.0,0.0);
glEnd();
```

```
glColor3f(0.0,1.0,0.0);
glBegin(GL_POLYGON)
    glVertex3f(40.0,40.0,0.0);
    glVertex3f(60.0,40.0,0.0);
    glVertex3f(60.0,60.0,0.0);
    glVertex3f(40.0,60.0,0.0);
glEnd();
```

Máquina de estados



Máquina de estados

- Mude a ordem no qual os quadrados aparecem cortando os sete comandos que especificam o quadrado vermelho e colando-os depois dos que desenham o quadrado verde. O quadrado verde é sobrescrito pelo vermelho porque OpenGL desenha na ordem do código.

Primitivas Geométricas

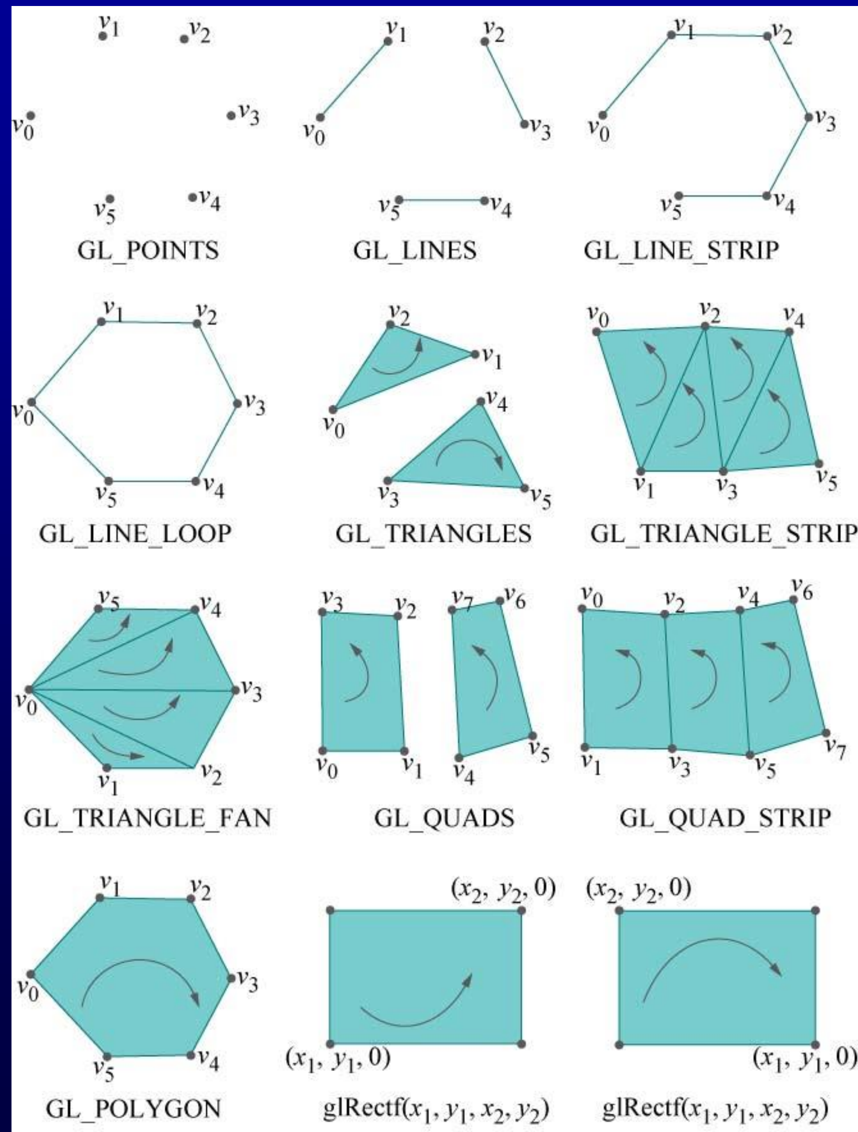
- Experimento 2.11: Substitua glBegin(GL_POLYGON) por glBegin(GL_POINTS) em square.c e faça os pontos maiores com a chamada a glPointSize(5.0), assim:

```
glPointSize(5.0)
glBegin(GL_POINTS)
    glVertex3f(20.0,20.0,0.0);
    glVertex3f(80.0,20.0,0.0);
    glVertex3f(80.0,80.0,0.0);
    glVertex3f(20.0,80.0,0.0);
glEnd();
```

Primitivas Geométricas

- Experimento 2.12: Continue substituindo GL_POINTS com GL_LINES, GL_LINE_STRIP e, finalmente, GL_LINE_LOOP.

Primitivas Geométricas



Primitivas Geométricas

- Experimento 2.13: Substitua a construção do polígono com o seguinte bloco:

```
glBegin(GL_TRIANGLES);
    glVertex3f(10.0, 90.0, 0.0);
    glVertex3f(10.0, 10.0, 0.0);
    glVertex3f(35.0, 75.0, 0.0);
    glVertex3f(30.0, 20.0, 0.0);
    glVertex3f(90.0, 90.0, 0.0);
    glVertex3f(80.0, 40.0, 0.0);
  glEnd();
```

Primitivas Geométricas

- Triângulos são desenhados preenchidos. Porém, podemos escolher um modo diferente de desenho aplicando glPolygonMode(face,mode), onde face pode ser GL_FRONT,GL_BACK ou GL_FRONT_AND_BACK, e mode pode ser GL_FILL, GL_LINE ou GL_POINT. Devemos ter em conta que a primitiva estará de frente o ou não dependendo d sua orientação.

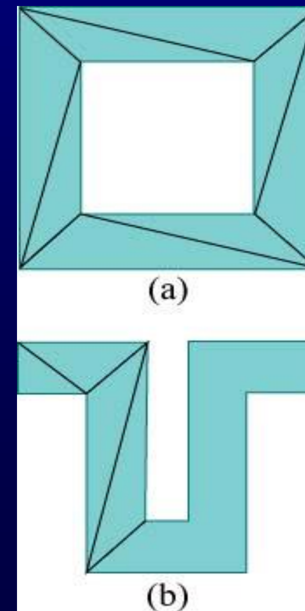
Primitivas Geométricas

- Experimento 2.14: Insira glPolygonMode (GL_FRONT_AND_BACK,GL_LINE) na rotina de desenho e substitua GL_TRIANGLES por GL_TRIANGLE_STRIP, assim

```
glPolygonMode(GL_FRONT_AND_BACK,GL_LINE)
glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(10.0, 90.0, 0.0);
    glVertex3f(10.0, 10.0, 0.0);
    glVertex3f(35.0, 75.0, 0.0);
    glVertex3f(30.0, 20.0, 0.0);
    glVertex3f(90.0, 90.0, 0.0);
    glVertex3f(80.0, 40.0, 0.0);
  glEnd();
```

Primitivas Geométricas

- Exercício: Crie o seguinte anel quadrado usando um único triangle strip. Você deve esboçar o anel em um papel para determinar as coordenadas dos seus oito cantos.
- Exercício: Cria a forma parcialmente triangulada da segunda figura usando um único triangle strip.



Primitivas Geométricas

Experimento 2.15: Substitua a construção do polígono pelo seguinte trecho:

```
glBegin(GL_TRIANGLE_FAN);
    glVertex3f(10.0, 10.0, 0.0);
    glVertex3f(15.0, 90.0, 0.0);
    glVertex3f(55.0, 75.0, 0.0);
    glVertex3f(70.0, 30.0, 0.0);
    glVertex3f(90.0, 10.0, 0.0);
  glEnd();
```

Aplique ambos os modos de desenho preenchido e wireframe.

Primitivas Geométricas

Exercício: Crie o anel quadrado da figura anterior usando dois triangle fans. Primeiro faça o esboço no papel.

Experimento 2.16: Substitua o trecho de construção do quadrad por

```
glBegin(GL_QUADS);
  glVertex3f(10.0, 90.0, 0.0);
  glVertex3f(10.0, 10.0, 0.0);
  glVertex3f(40.0, 20.0, 0.0);
  glVertex3f(35.0, 75.0, 0.0);
  glVertex3f(55.0, 80.0, 0.0);
  glVertex3f(60.0, 10.0, 0.0);
  glVertex3f(90.0, 20.0, 0.0);
  glVertex3f(90.0, 75.0, 0.0);
```

Primitivas Geométricas

Aplique o modo de desenho preenchido e wireframe.

Experimento 2.17: Substitua o trecho de construção do quadrad
por

```
glBegin(GL_QUAD_STRIP);
  glVertex3f(10.0, 90.0, 0.0);
  glVertex3f(10.0, 10.0, 0.0);
  glVertex3f(30.0, 80.0, 0.0);
  glVertex3f(40.0, 15.0, 0.0);
  glVertex3f(60.0, 75.0, 0.0);
  glVertex3f(60.0, 25.0, 0.0);
  glVertex3f(90.0, 90.0, 0.0);
  glVertex3f(85.0, 20.0, 0.0);
glEnd();
```