OpenGL

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 como renderizador

Generalmente, há duas operações que OpenGL faz :

- Desenhar algo através de primitivas geométricas e de imagens.
- Mudar o estado de como OpenGL desenha.

- Primitivas Geometricas
  - Pontos, linhas e polígonos
- Primitivas de Imagens
  - imagens e bitmaps (os pixels que podemos extrair de uma imagem JPEG depois de lida)
  - Adicionalmente, OpenGL pode linkar imagens e primitavas geométricas através de um mapeamento de textura.
- OpenGL funciona como uma máquina de estados. A renderização depende do estado
  - cores, materiais, fontes de luz, etc.

OpenGL

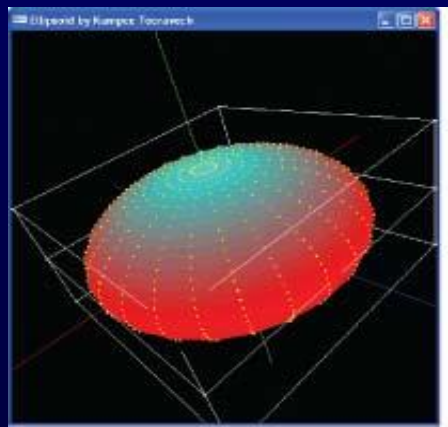
- Alguns programas executáveis em windows mostrando as potencialidades do OpenGL.

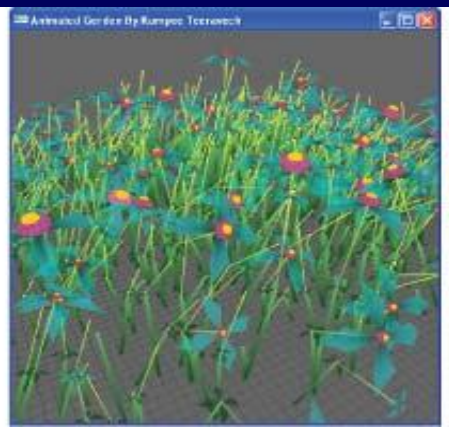
http://www.comp.uems.br/~mercedes/disciplinas/2019/CG/SUMANTA GUHA/

- 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/2019/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

GLU e GLUT

- GLU (OpenGL Utility Library)
  - parte de OpenGL
  - Simplifica tarefas como renderização de superfícies quádricas como esferas, cones, cilindros, etc.
- 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

Preliminares

- Arquivos cabeçalho (headers files): descreve para o compilador, as chamadas de funções, seus parâmetros e valores constantes definidos.
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Bibliotecas : depende do sistema que estamos usando. Cada sistema operacional tem seu próprio conjunto de bibliotecas. Para sistema Unix é `libGL.so` e para Microsoft Windows é `opengl32.lib`.
- Enumerated Types
  - GLfloat, GLint, etc.

GLUT Básico

- Estrutura da aplicação
  - 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.
    - De renderização (display)
    - De redimensionamento de janela
    - De entrada: teclado, mouse, etc.
  - Entre no loop de processamento de eventos : Quando a aplicação recebe eventos, e controla quando as funções callbacks serão chamadas.

Programa exemplo

- Escolheremos como nosso primeiro exemplo, o programa square.c, no qual um quadrado preto sobre fundo branco é criado.

```
int main(int argc, char **argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("square.cpp");
   setup();
   glutDisplayFunc(drawScene);
   glutReshapeFunc(resize);
   glutKeyboardFunc(keyInput);
   glutMainLoop();
    return 0;
}
```

Funções de configuração da janela

Inicalização de estados

Rotinas Callbak

Loop de processamento de eventos

Inicialização de OpenGL

- Fixa alguns estados

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

Funções Callback GLUT

- O usuário não precisa receber e processar cada evento. Callbacks da GLUT simplificam o processo definindo quais ações são suportadas e automaticamente manejando os eventos.
- São rotinas que são chamadas quando algo como o seguinte acontece
  - Redimensionamento ou redesenho da janela
  - Entrada do usuário
  - animação
- Registro de callbacks com GLUT

```
glutDisplayFunc( display )
glutIdleFunc( idle );
glutKeyboardFunc( keyboard );
```

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.

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;
   }
```

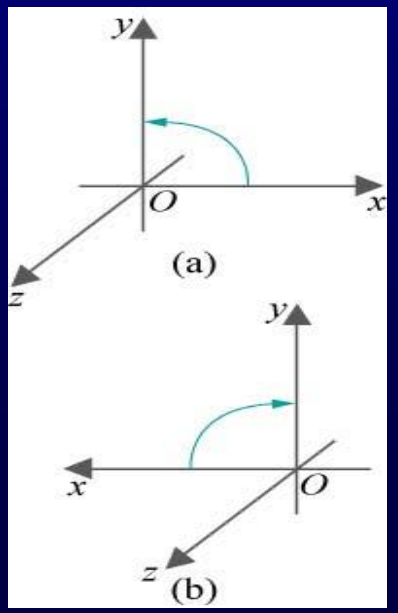
Primeiro Programa

- 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

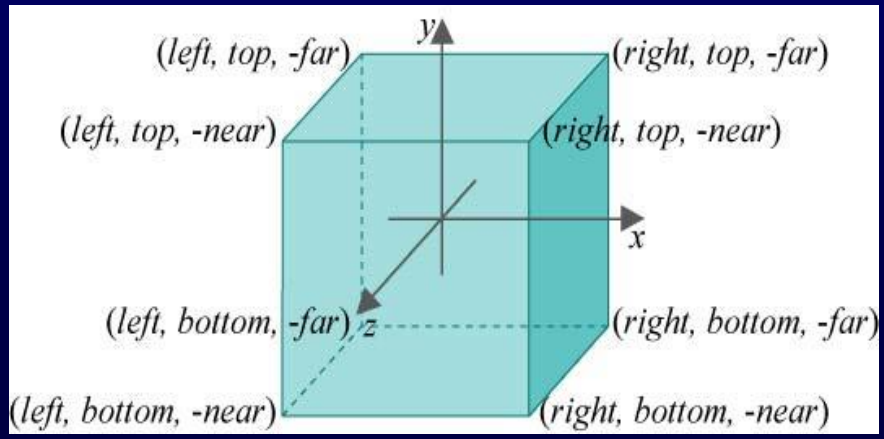
(a) sistema mão direita

(b) sistema mão esquerda



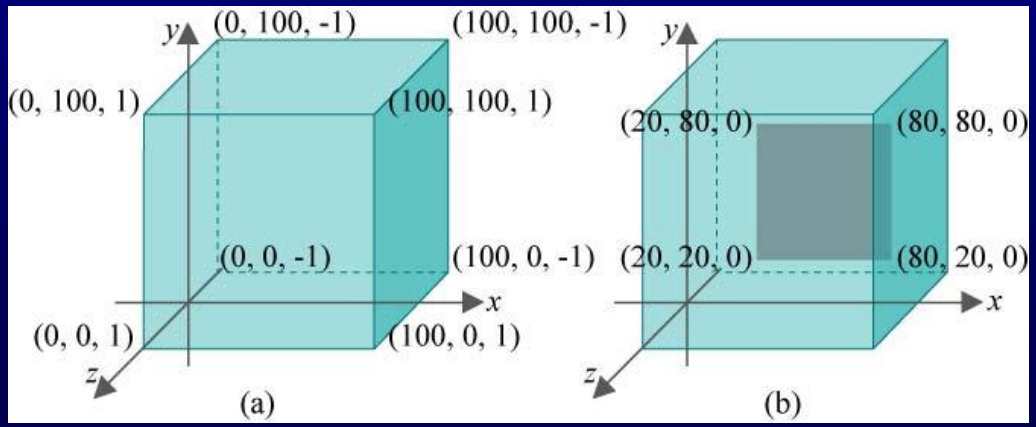
Projeção Ortográfica, Volume de Visualização

- O comando glOrtho (left, right, bottom, top, near, far)
  - especifica o volume de visualização (vi) onde a cena 3D deverá estar contida,
  - a 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



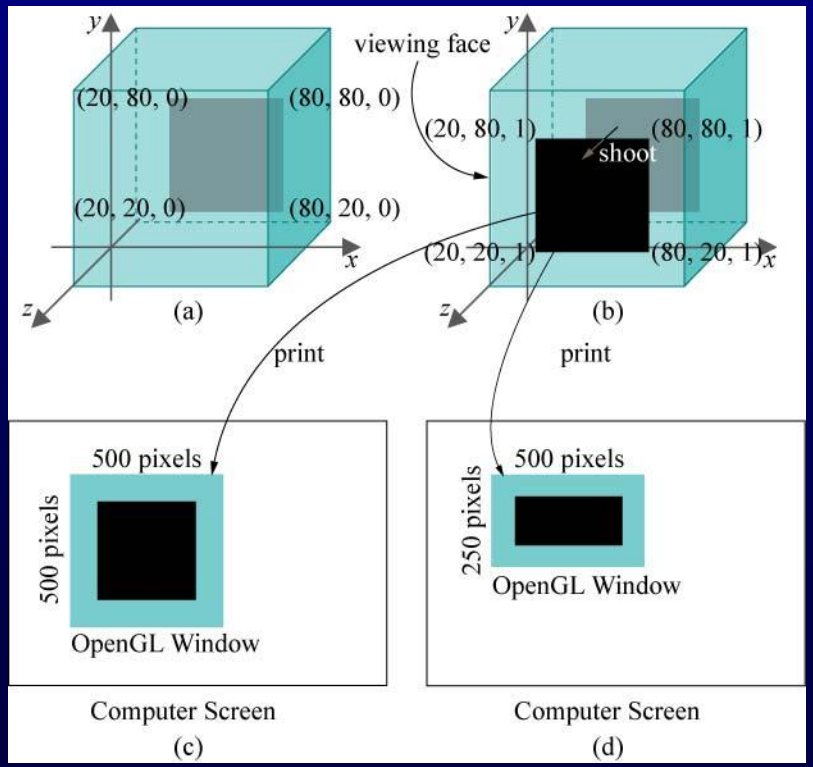
Projeção Ortográfica, Volume de Visualização

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



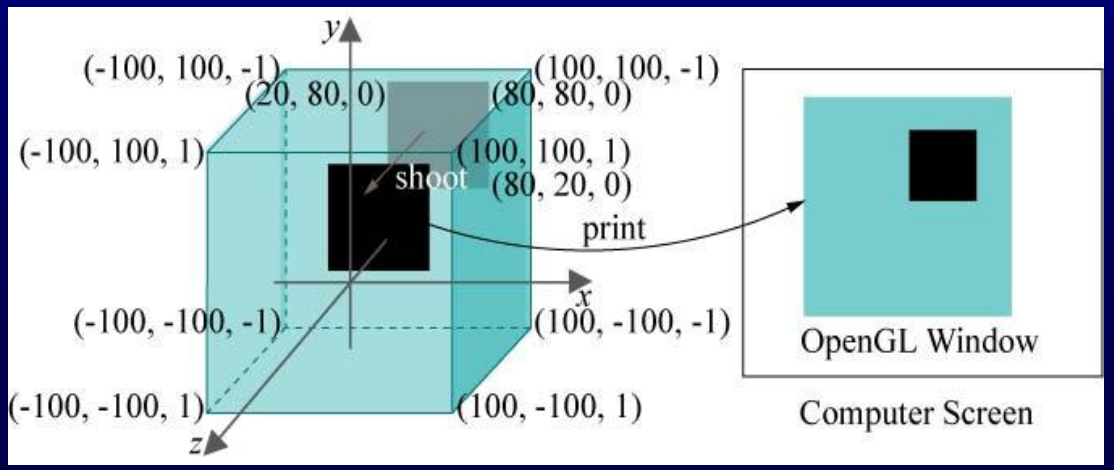
Projeção Ortográfica, Volume de Visualização

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



Projeção Ortográfica, Volume de Visualização

- Experiência: 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)

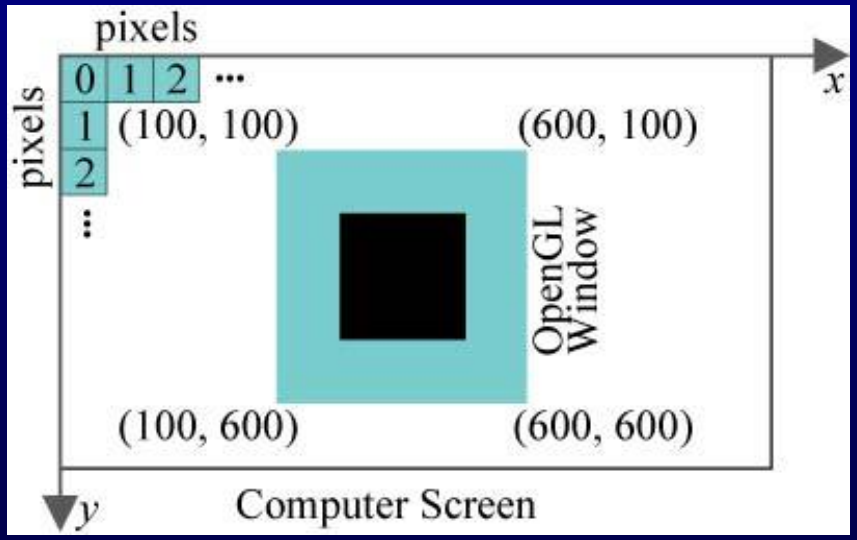
(c) glOrtho (0.0,100.0,0.0,100.0,-2.0,5.0), em todos os casos tente prever o resultado.

- 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

- 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

- 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)

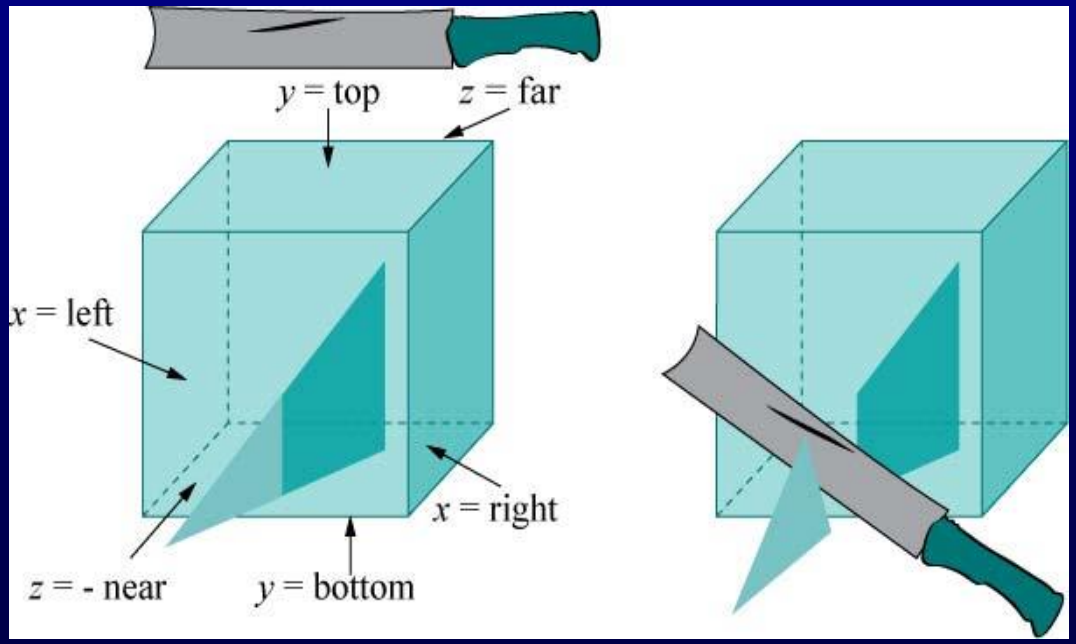
(b) glVertex(20.0,20.0,1.5)

(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 é 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

Cor

- Geralmente, glColor3f(red,green,blue) especifica a cor do primeiro plano, o a cor do desenho. O valor de cada componente de cor (que deve estar entre 0 e 1) determinar 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: 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 punto, espessura da linha, pontilhado da linha, propriedades de material da superfície, etc. OpenGL permanece e funciona no seu estado corrente até que uma declaração é feita mudando a variável de estado.

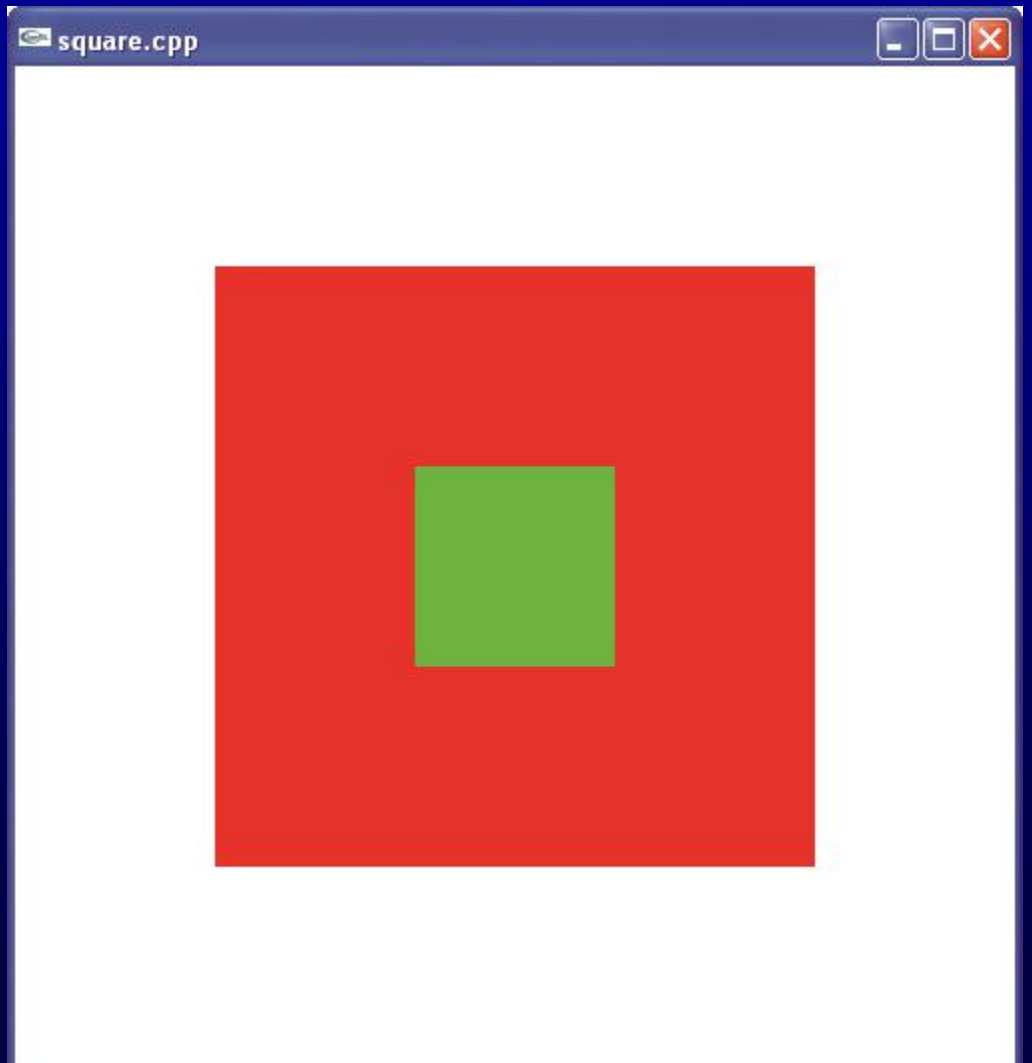
Máquina de estados

- Experimento: 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

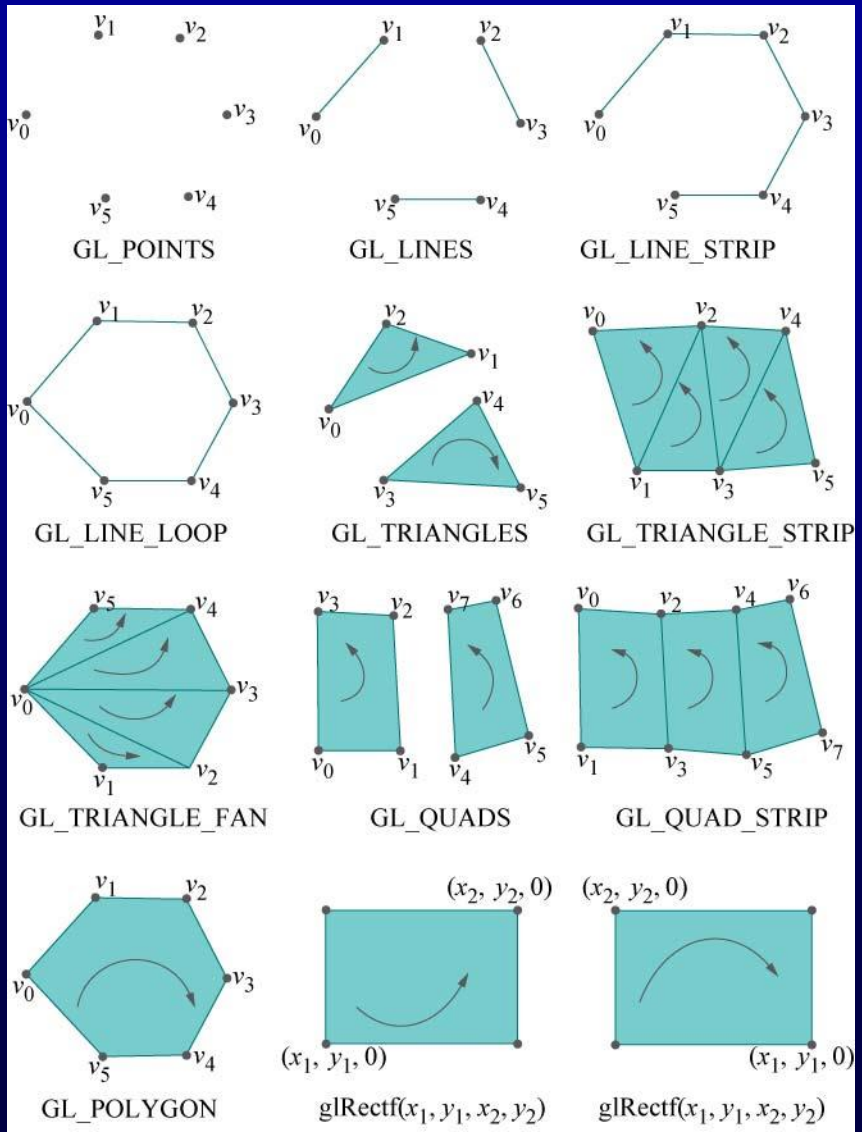
- Experimento: Substitua glBegin(GL_POLYGON) por glBegin(GL_POINTS) em square.c e faça os pontos maiores com a chamada a glPointSize(5.0), assim:

```
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: Continue substituindo GL_POINTS com GL_LINES, GL_LINE_STRIP e, finalmente, GL_LINE_LOOP.

Primitivas Geométricas



Primitivas Geométricas

- Experimento: 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 o sua orientação.

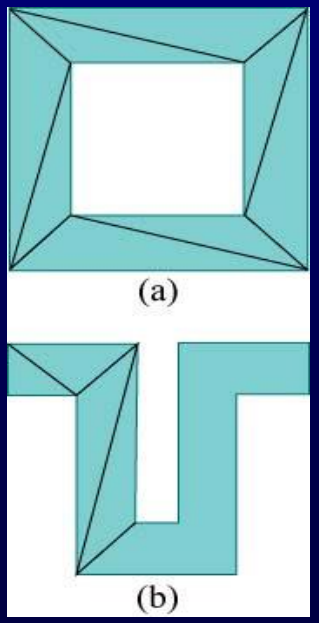
Primitivas Geométricas

- Experimento: 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: 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: Substitua o trecho de construção do quadrado 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);
glEnd();
```

Primitivas Geométricas

Aplique o modo de desenho preenchido e wireframe.

Experimento: Substitua o trecho de construção do quadrado 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();
```

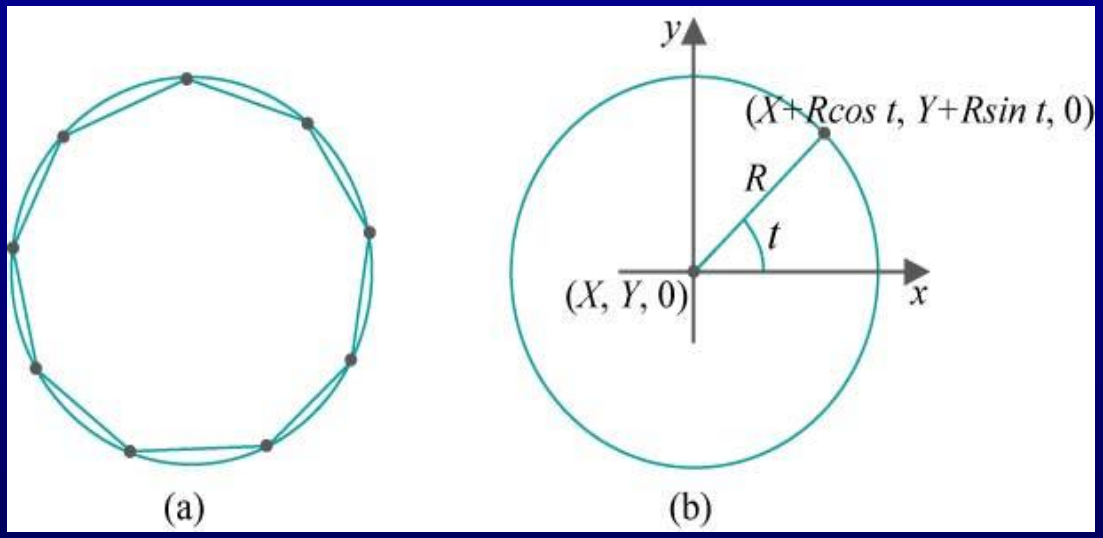

Aplique o modo de desenho preenchido e wireframe.

Objetos curvos aproximados

Até aqui temos visto que as primitivas geométricas do OpenGL são pontos, segmentos de retas e figuras planas como triângulos, quadriláteros e polígonos. Como, então, desenhar objetos como discos, elipses, espirais, etc. A resposta é, aproximando-os com primitivas retas e planas de forma tão suficiente que o observador não note a diferença.

Experimento 2.20: Compile e rode o programa circle.cpp. Incremente o número de vértices do “loop” pressionando “+” até que este se torne um circulo. Pressione “-” para decrementar o número de vértices.

Objetos curvos aproximados



A equação paramétrica do circulo implementado é:

x=X+Rcost, y=Y+Rsint, z=0, 0<=t<=2pi

Onde (X,Y,0) é o centro e R é o raio do círculo.

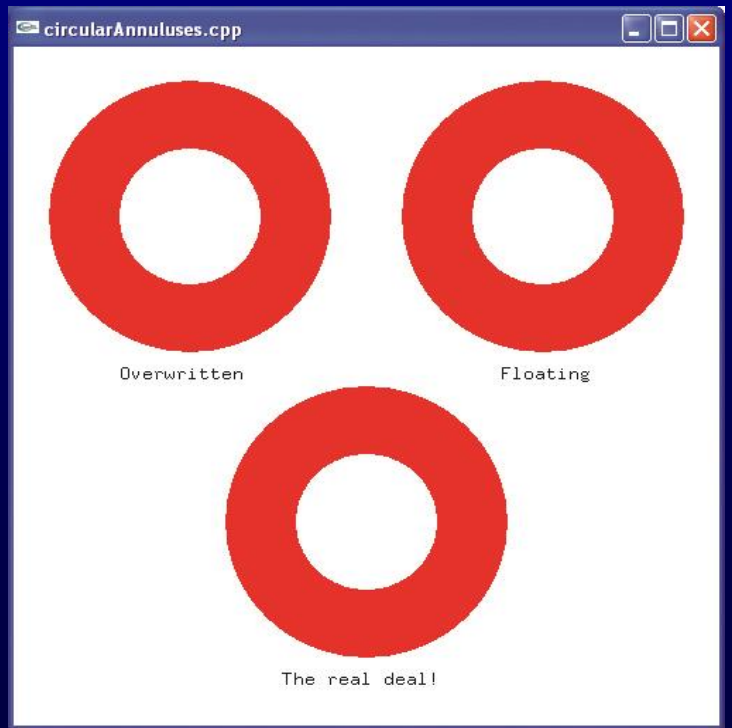
Objetos curvos aproximados

O programa também mostra uma interação via teclado. A rotina keyInput() é registrada como uma rotina de tratamento de teclado em main() pelo comando glutKeyboardFunc(keyInput).

Perceba também as chamadas a glutPostRedisplay() em keyInput() pedindo que o display seja redesenhado depois de cada atualização de numVertices.

Buffer de profundidade

Experimento 2.22: Rode o programa circularAnnuluses.cpp. Três aneis circulares de idêntica aparência são desenhados em três formas diferentes.



Buffer de profundidade

(a) Superior esquerdo: Não há um furo real. O disco branco sobre escreve o disco vermelho em

```
glColor3f (1.0,0.0,0.0);
drawDisc(20.0,25.0,75.0,0.0);
glColor3f (1.0,1.0,1.0);
drawDisc(10.0,25.0,75.0,0.0);
```

O primeiro parâmetro de drawDisc() é o raio e os outros três, as coordenadas do centro.

Buffer de profundidade

(b) Superior direito: Não há um furo real, também. O disco branco é desenhado mais perto ao observador do que o disco vermelho, bloqueando-o na região central.

```
glEnable(GL_DEPTH_TEST);
glColor3f (1.0,0.0,0.0);
drawDisc(20.0,75.0,75.0,0.0);
glColor3f (1.0,1.0,1.0);
drawDisc(10.0,75.0,75.0,0.5);
glDisable(GL_DEPTH_TEST);
```

Veja que o valor z do centro do disco branco é maior que o do disco vermelho.

Buffer de profundidade

(c) Inferior: Um verdadeiro anel circular com um furo real

```
if (isWire) glPolygonMode(GL_FRONT,GL_LINE);
else glPolygonMode(GL_FRONT,GL_FILL);
glColor3f(1.0,0.0,0.0);
glBegin(GL_TRIANGLE_STRIP);
…
glEnd();
```

Pressione a barra de espaço para ver o modo wireframe.

```
olygonMode(GL_FRONT,GL_LINE);
```

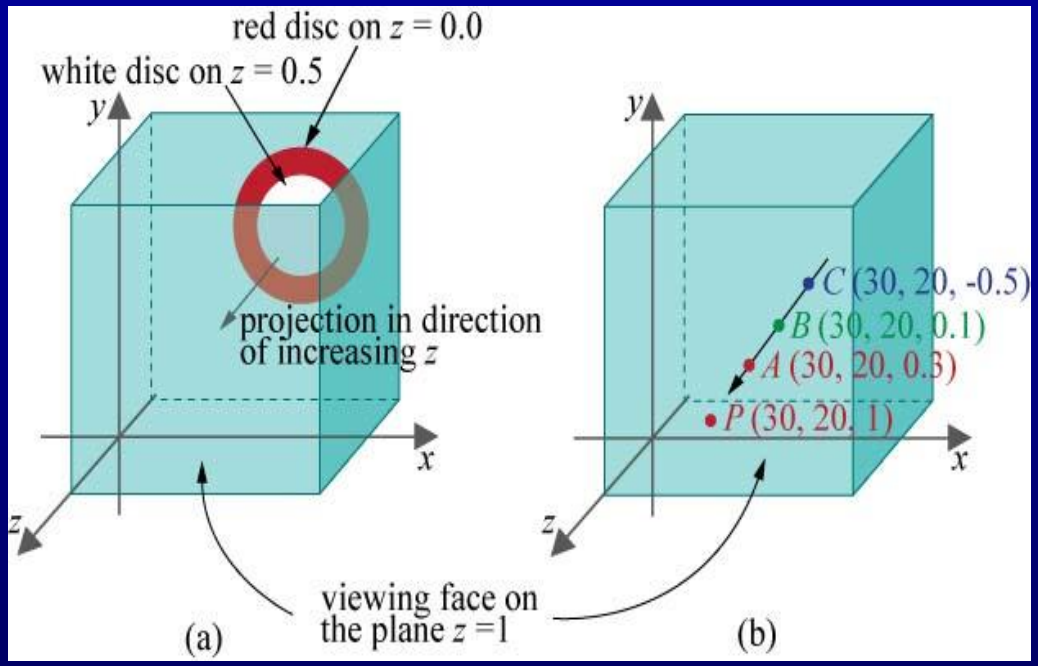
Buffer de profundidade

Exercício: Troque a ordem de desenho dos discos vermelho e branco nos anéis da parte superior. Qual dos dois é afetado e por quê?

O buffer de profundidade faz com que OpenGL elimine partes dos objetos que são ocluídos por outros.

Um ponto de um objeto não é desenhado se sua projeção na face de visualização é obstruída por outro objeto. Esse processo é chamado de remoção de superfícies escondidas ou teste de profundidade ou determinação de visibilidade.

Buffer de profundidade



Os três pontos A,B e C, coloridos de vermelho, verde e azul, respectivamente, compartilham os mesmos valores x e y e todos são projetados ao ponto P na face de visualização. Já que A tem a coordenada z maior que os outros dois, então P é desenhado vermelho.

Buffer de profundidade

Note o uso de três comandos:

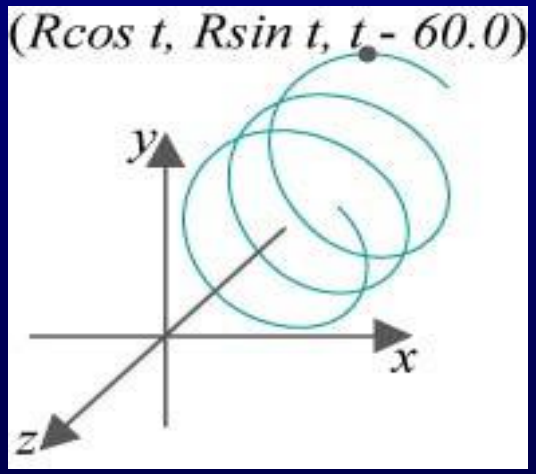
(a) O parâmetro GL_DEPTH_BUFFER_BIT do comando glClear para limpar o buffer.

(b)O comando glEnable(GL_DEPTH_TEST) para habilitar o buffer.

(c) O parâmetro GL_DEPTH do comando glutInitDisplayMode, para inicializar o buffer.

Projeção Perspectiva

Veja o programa helix.cpp que usa as equações paramétricas x=Rcost, y=Rsint, z=t-60.0, -10pi<=t<=10pi.

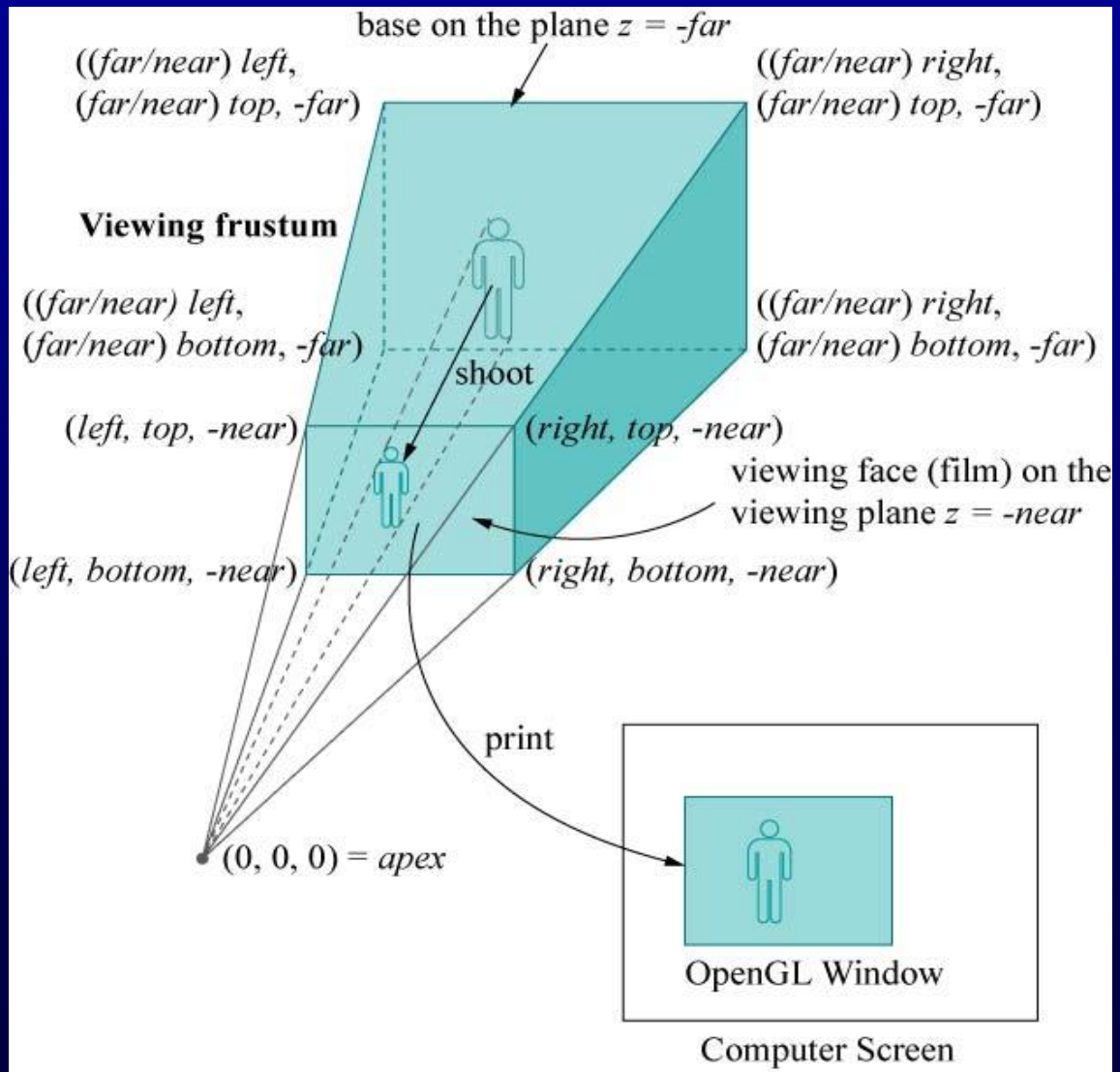


Projeção Perspectiva

- Experimento 2.23: Rode o programa helix.cpp e veja que apenas um círculo é visualizado. A razão é que a projeção ortográfica sobre a face de visualização aplana a hélice e por essa característica, a projeção ortográfica muitas vezes não é adequada para cenas 3D.
- OpenGL fornece outro tipo de projeção chamada projeção perspectiva, mais apropriada para aplicações 3D.

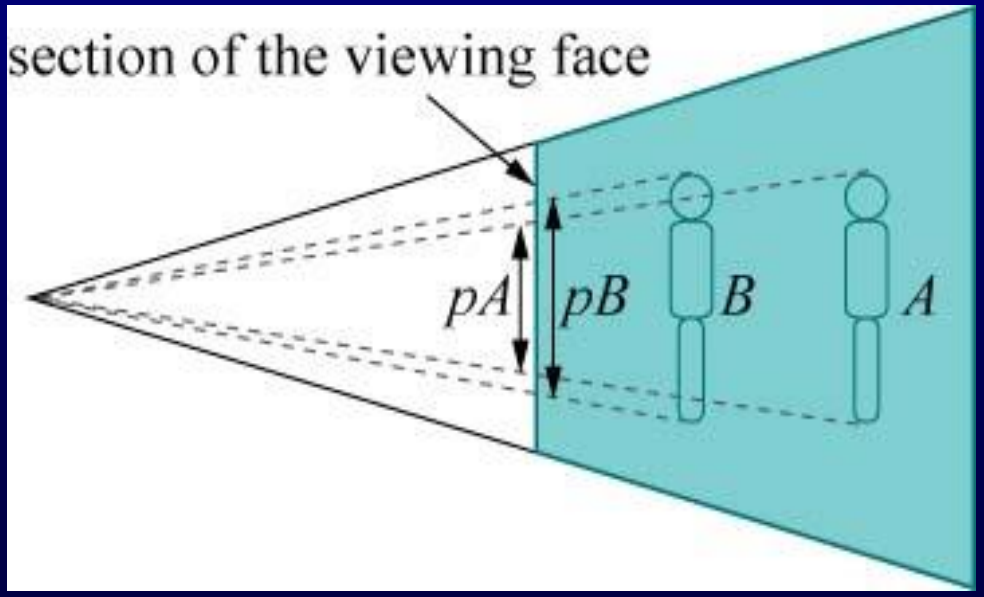
No lugar de uma caixa de visualização, glFrustum(left,right,bottom,top,near,far) configura uma pirâmide truncada cujo topo foi cortado por um plano paralelo a sua base. Right e top devem ser positivos, e left e bottom seus correspondentes negativos. Near e far devem ser positivos e near<far.

Projeção Perspectiva



Projeção Perspectiva

Projeção perspectiva causa encurtamento porque os objetos mais afastados do ápice, aparecem menores. Observe a figura,+ onde A e B são da mesma altura, mas a projeção pA é menor que a projeção pB.



Projeção Perspectiva

Experimento 2.24: No programa helix.cpp, substitua a projeção ortográfica pela projeção perspectiva fazendo

glFrustum(-5.0,5.0,-5.0,5.0,5.0,100.0)

Você pode ver agora uma espiral real!

Projeção perspectiva é mais realística que projeção ortográfica porque ela imita a forma que as imagens são formadas na retina do olho pelos raios de luz viajando em direção a um ponto fixo

Projeção Perspectiva

Exercícios: Desenhe uma curva senoidal entre x=-pi e x=pi. Siga a estratégia do programa circle.cpp.

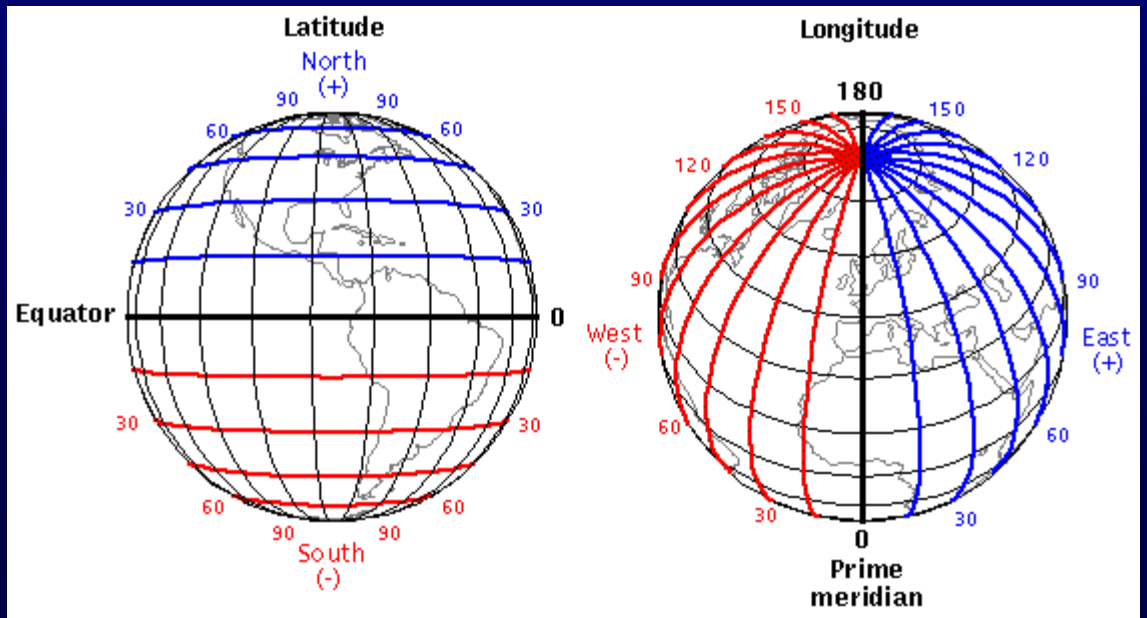
Resolva a lista de exercícios no site da disciplina.

Aproximando Objetos Bi-dimensionais

Uma circunferência ou uma hélice são objetos unidimensionais pois são topologicamente equivalentes a uma linha. Agora veremos como aproximar objetos bi-dimensionais como uma esfera ou um cilindro.

A representação paramétrica de uma esfera segue o modelo de coordenadas geográficas do globo terrestre (latitude e longitude).

https://www.geogebra.org/m/HJs4kEtb



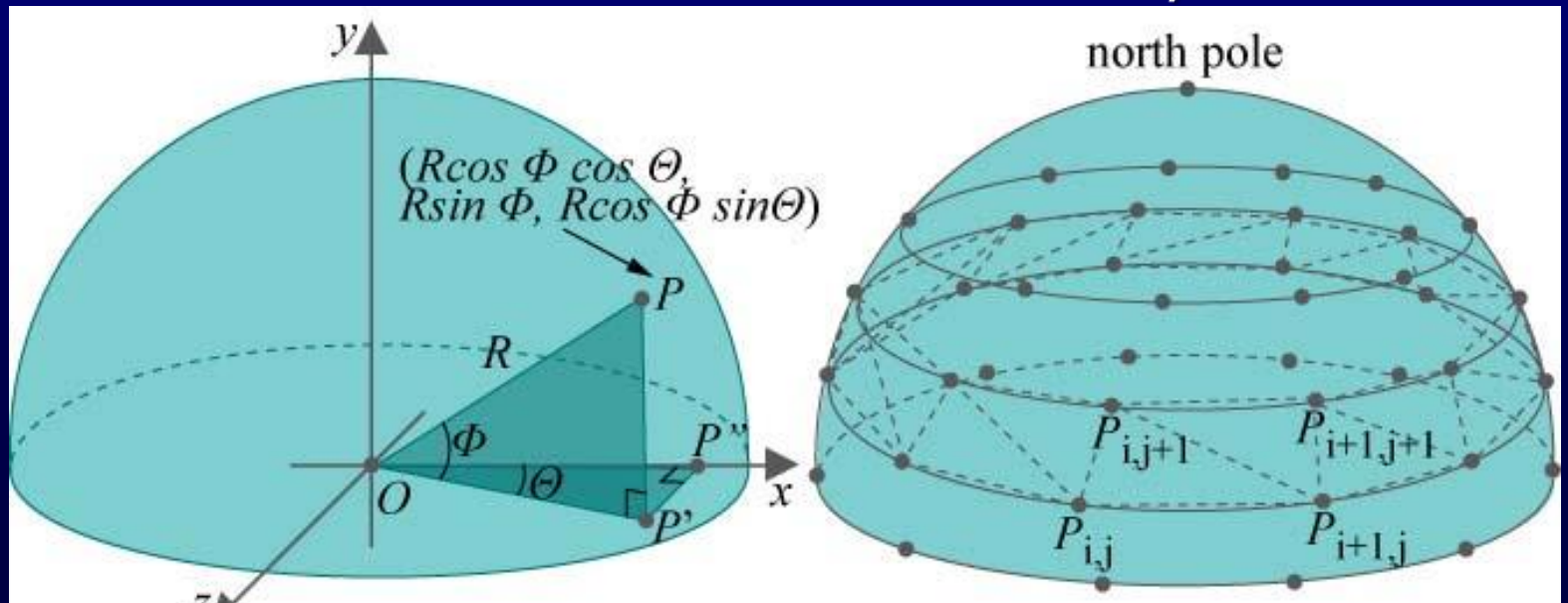
Aproximando Objetos Bi-dimensionais

Considere um hemisfério de raio R, centralizado na origem O e com sua base circular sobre o plano xz. Suponha que as coordenadas esféricas de um ponto P sobre o hemisfério são a longitude theta e a latitude phi.

As coordenadas cartesianas são:

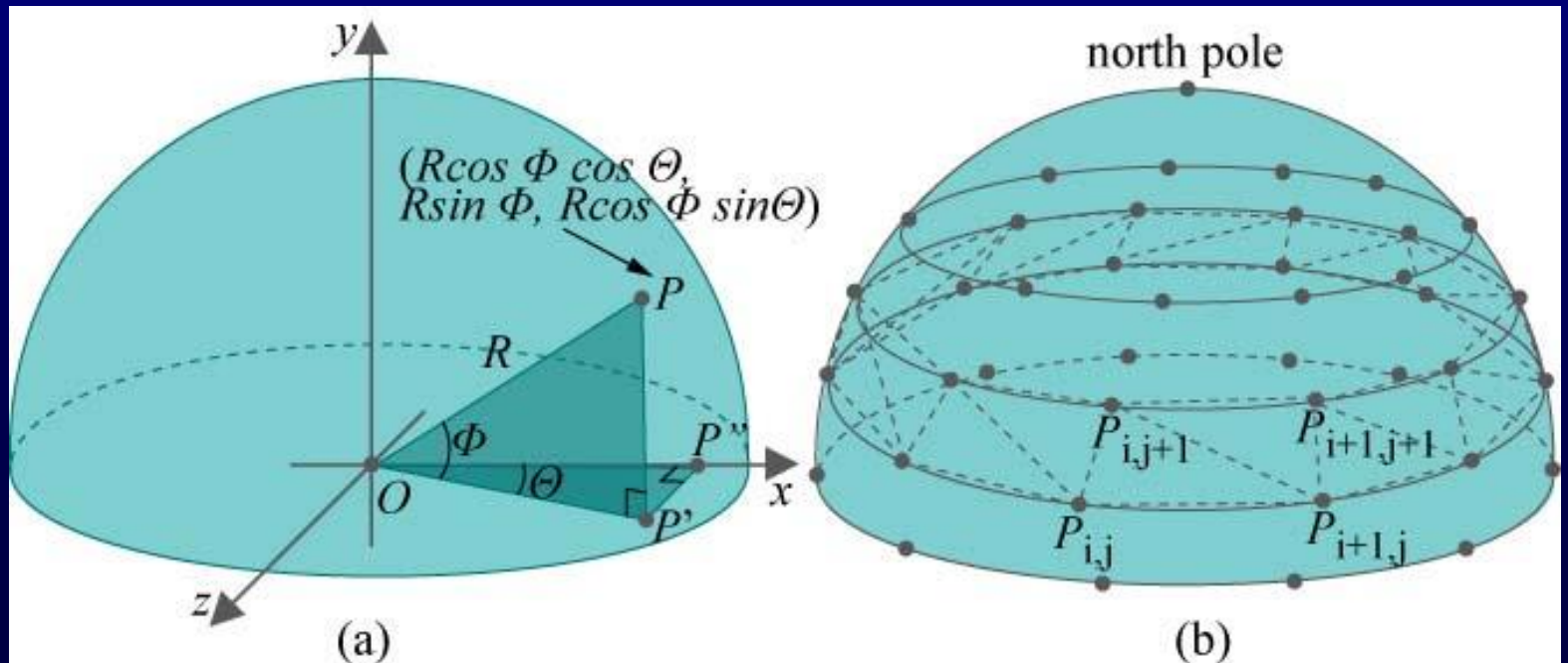
$$(RCos\emptyset Cos\theta, RSin\emptyset, RCos\emptyset Sin\theta),$$

$$0 \leq \theta \leq 2\pi \; e \; 0 \leq \emptyset \leq \pi/2$$



Aproximando Objetos Bi-dimensionais

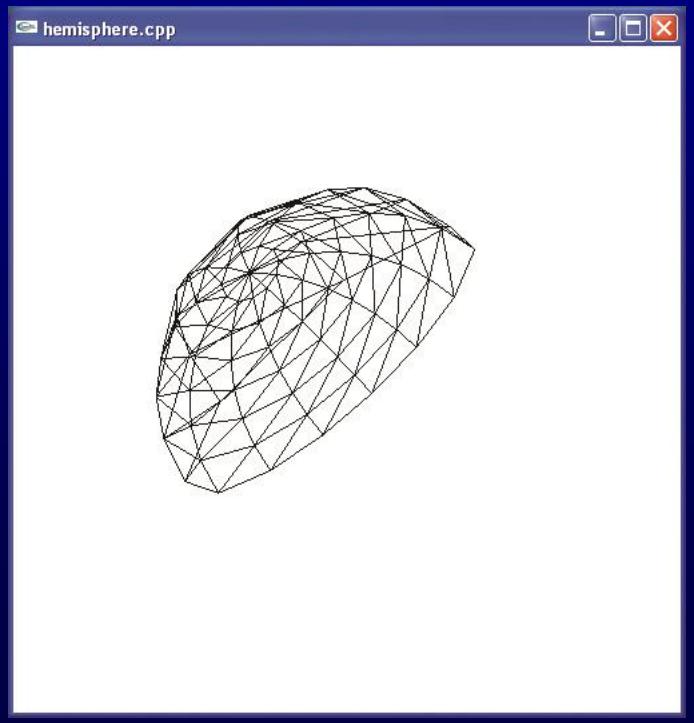
Amostramos o hemisfério em uma malha de (p+1)(q+1) pontos Pij, 0<=i<=p, 0<=j<=q, onde a longitude de Pij é (i/p)*2PI e sua latitude (j/q)*PI/2. Em outros palavras (p+1) pontos longitudinalmente igualmente espaçados são escolhidos entre cada uma das q+1 latitudes igualmente espaçadas. Na figura p=10 e q=4.



Aproximando Objetos Bi-dimensionais

Experimento 2.26. Rode hemisphere.cpp que implementa exatamente a estratégia descrita.

Experimento 2.27.



Aproximando Objetos Bi-dimensionais

even now as the syntax is fairly intuitive. The set of three `glRotatef()`s, particularly, comes in handy to re-align a scene.

Exercise 2.31. (Programming) Modify `hemisphere.cpp` to draw:

(a) the bottom half of a hemisphere (Figure 2.35(a)).

(b) a 30° slice of a hemisphere (Figure 2.35(b)).

Make sure the 'P/p/Q/q' keys still work.

Exercise 2.32. (Programming) Just to get you thinking about animation, which we'll be studying in depth soon enough, guess the effect of replacing `glTranslatef(0.0, 0.0, -10.0)` with `glTranslatef(0.0, 0.0, -20.0)` in `hemisphere.cpp`. Verify.

And, here are some more things to draw.

Exercise 2.33. (Programming) Draw the objects shown in Figure 2.36. Give the user an option to toggle between filled and wireframe renderings.

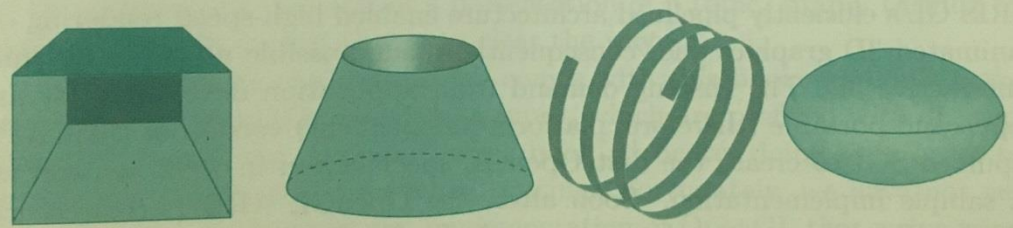


Figure 2.36: More things to draw.

A suggestion for the football, or ellipsoid, is to modify `hemisphere.cpp` to make half of an ellipsoid (a hemi-ellipsoid?). Two hemi-ellipsoids back to back would then give a whole ellipsoid.

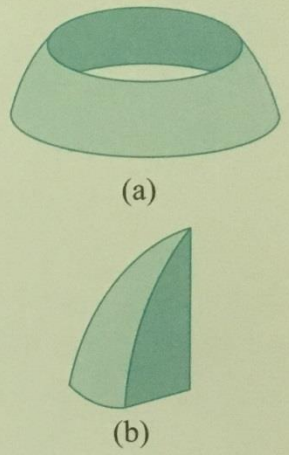


Figure 2.35: (a) Half a hemisphere (b) Slice of a hemisphere.