

OpenGL (2º Parte)

Professora: Mercedes Gonzales Márquez

Aspectos básicos de OpenGL

- Nos slides a seguir continuamos apresentando comandos básicos de OpenGL juntamente com os experimentos do Capítulo 2 e 3 do arquivo `experimenter.pdf`. Falaremos sobre:
 1. Primitivas Geométricas
 2. Buffer de Profundidade
 3. Projeção Perspectiva
 4. Listas, fontes, teclas não ASCII, mouse, fontes, menus e linhas.

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);
```

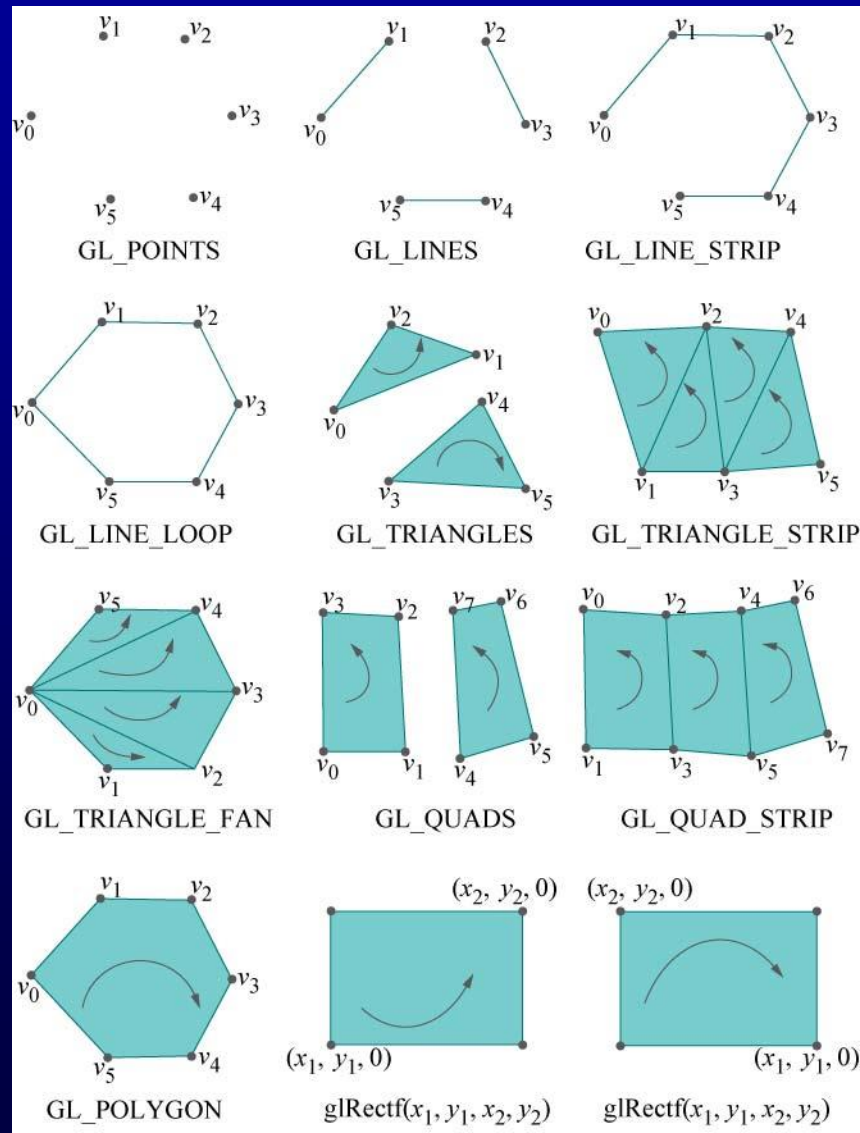
```
    glVertex3f(20.0,80.0,0.0);
```

```
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 da 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);
```

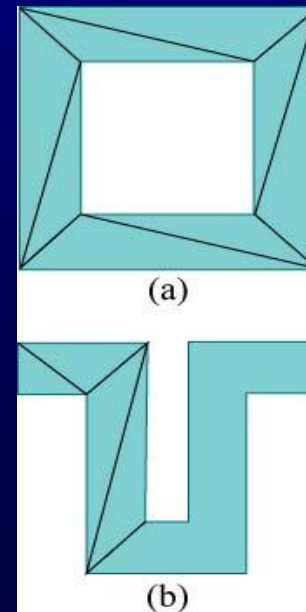
```
    glVertex3f(90.0, 90.0, 0.0);
```

```
    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 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);
```

Primitivas Geométricas

Aplique o modo de desenho preenchido e wireframe.

Experimento 2.17: 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();
```

Primitivas Geométricas

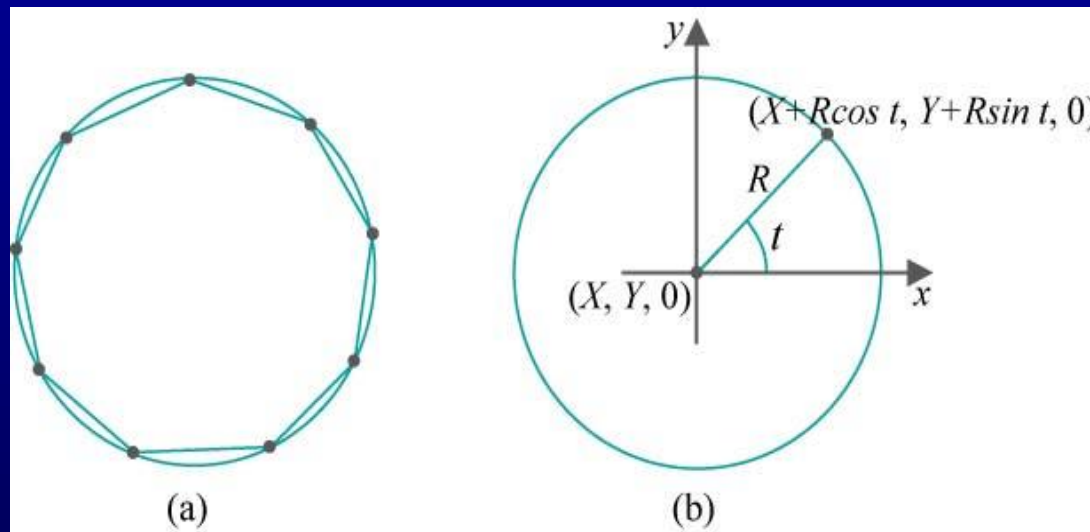
Experimentos 2.18 e 2.19

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 círculo. Pressione “-” para decrementar o número de vértices.

Objetos curvos aproximados



A equação paramétrica do círculo implementado é:

$$x = X + R \cos t, \quad y = Y + R \sin t, \quad z = 0, \quad 0 \leq t \leq 2\pi$$

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

Objetos curvos aproximados

- Explicando a implementação da equação do círculo
- Suponha $N=4$ $t=2*\text{PI}*i/N$
- Para $i=0 \Rightarrow t=2*\text{PI}*0/4 = 0 = 0$
- Para $i=1 \Rightarrow t=2*\text{PI}*1/4 = \text{PI}/2 = 90$
- Para $i=2 \Rightarrow t=2*\text{PI}*2/4 = \text{PI} = 180$
- Para $i=3 \Rightarrow t=2*\text{PI}*3/4 = 3*\text{PI}/2 = 270$

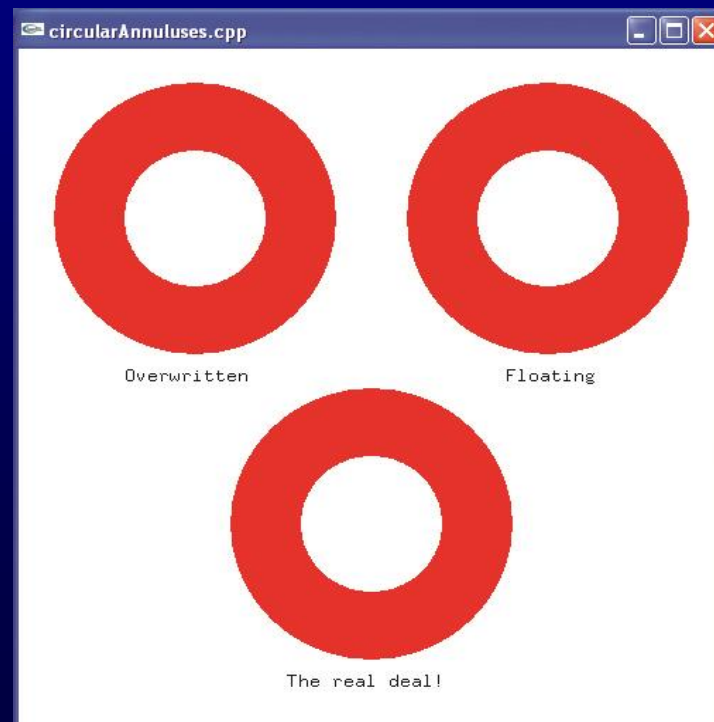
Objetos curvos aproximados

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 anéis 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.

```
PolygonMode(GL_FRONT, GL_LINE);
```

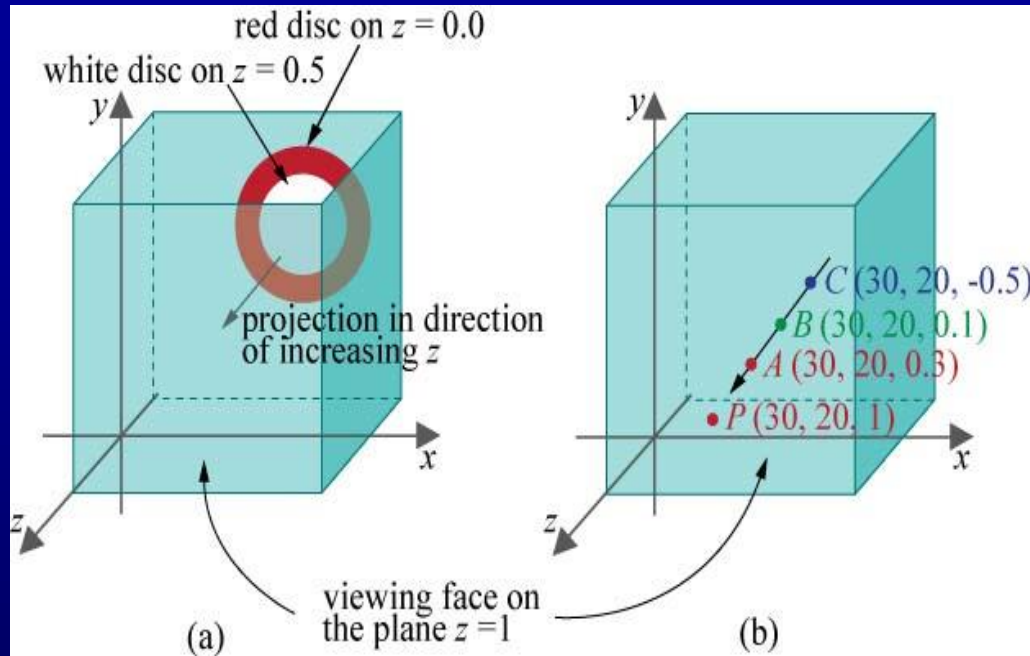
Buffer de profundidade

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:

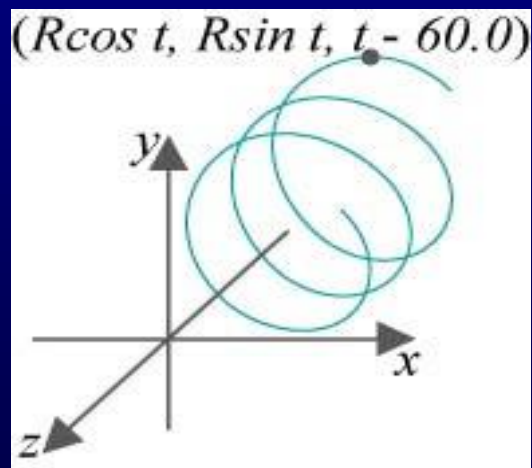
(a) O parâmetro `GL_DEPTH_BUFFER_BIT` do comando `glClear` para limpar o buffer.

(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 $x=R\cos t$, $y=R\sin t$, $z=t-60.0$, $-10\pi \leq t \leq 10\pi$.

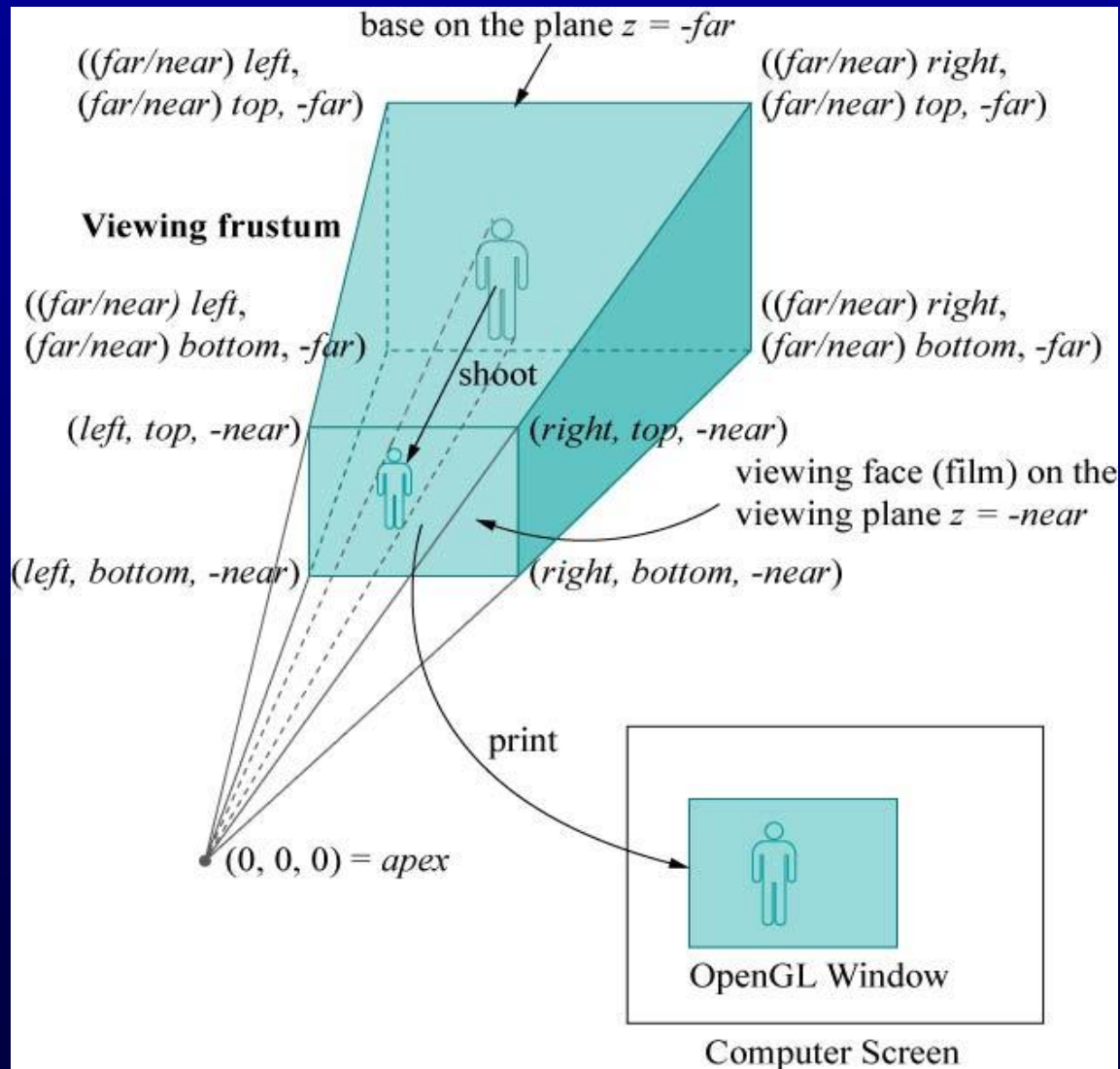


Projeção Perspectiva

- 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 aplanada 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.

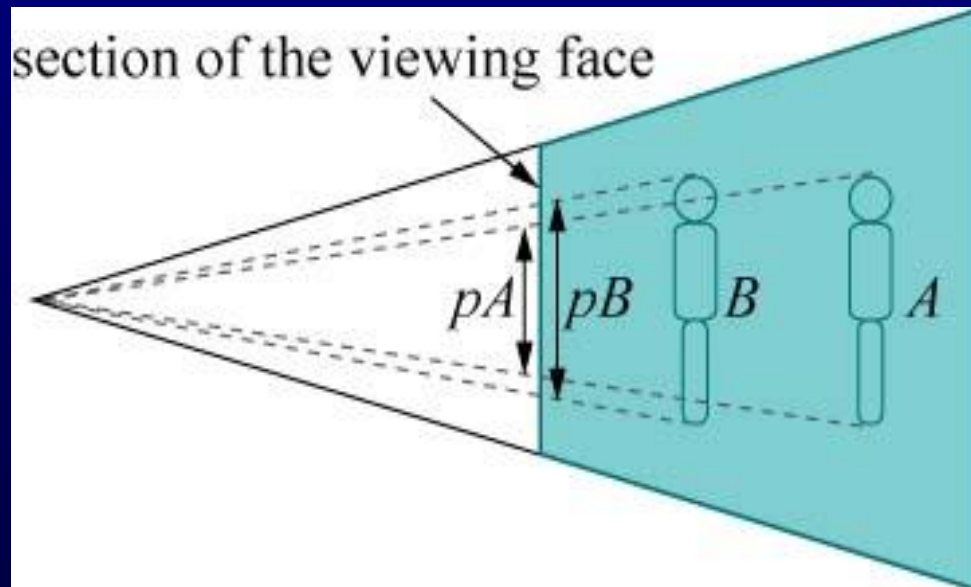
No lugar de uma caixa de visualização, `glFrustum(left,right,bottom,top,near,far)` configura uma pirâmide truncada cujo topo foi cortado por um plano paralelo a sua base. Right e top devem ser positivos, e left e bottom seus correspondentes negativos. Near e far devem ser positivos e `near < far`.

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

Primeira Lista

Resolva a lista de exercícios no site da disciplina.

Listas

- Em OpenGL os dados podem ser conservados em uma lista da exposição para uso corrente ou serem usados mais tarde. (Como alternativa ao modo de processamento 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.

OpenGL - Listas

- As listas de exibição aperfeiçoam o desempenho de aplicações OpenGL.
- As listas de exibição são indicadas para: Operações complexas com matrizes, luzes, propriedades de materiais e modelos de iluminação complexos, texturas.
- Por exemplo o comando `glRotate * ()` poderia mostrar uma significativa melhoria de performance se estivesse em uma lista de exibição, uma vez que os cálculos para produzir a matriz de rotação não são triviais.

OpenGL - Listas

- `GLuint glGenLists(GLsizei range);`

Este comando retorna um inteiro que inicia um bloco de tamanho `range` de índices de lista de display disponíveis.

OpenGL - Listas

- `void glNewList (GLuint lista, GLenum modo);`

Especifica o início de uma lista de display. As rotinas OpenGL chamadas subsequentemente (enquanto o comando `glEndList()` não for executado) são armazenados na lista de display. O parâmetro `lista` é um inteiro positivo maior que zero, que identifica unicamente a lista de display. Os possíveis valores para modo são `GL_COMPILE` e `GL_COMPILE_AND_EXECUTE`.

OpenGL - Listas

- `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.

OpenGL - Listas

- `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.

Veja Experimento 3.6.

Exercício : Faça um anel de círculos concêntricos de múltiplas cores no plano xy usando listas. Use o fator de escala u , assim `glScale (u,u,1.0);`

OpenGL - Fontes

- Texto gráfico pode ser de dois tipos: bitmapped (ou raster) e stroke (ou vetorial).
- Veja o experimento 3.8.
- Exercício: Escreva os textos do programa `circularAnnuluses.cpp` no centro de cada anel (você pode precisar dividir os textos em mais de uma linha).

OpenGL - Mouse

- Botões do mouse podem ser programados para responder a cliques e ao movimento do mesmo.
- Veja o experimento 3.9 e 3.10.
- Veja um programa em C na página da disciplina.
- Exercício: Escreva um programa para desenhar um círculo depois de dois cliques do mouse. O primeiro clique será para a origem do círculo e o segundo clique para definirmos o raio.
- Exercício: Melhore o programa anterior para permitir que usuário veja o círculo mudando à medida que ele arrasta o segundo ponto.

OpenGL – Teclas Não ASCII

- Veja o experimento 3.11.

OpenGL - Menus

- Veja o experimento 3.12.
- Exercício: Melhore o programa `menus.cpp` para adicionar mais dois itens ao topo do menu:
Modo: Aramado ou preenchido
Tamanho com as opções: Largura e altura, os quais tem subopções Pequeno, médio e grande.

OpenGL – Linhas Pontilhadas

- Veja o experimento 3.13.
- Exercícios: Aplicar as diferentes linhas pontilhadas ao círculo no programa `circle.cpp`.
- Exercícios: Aplicar as diferentes linhas pontilhadas ao círculo no programa `circle.cpp`.
- E agora juntando tudo no experimento 3.14 vemos sobre um canvas um desenho com menu e funcionalidades do mouse. Rode o programa `canvas.cpp`