Figuras 3D, Transformações Geométricas 3D e Animação

Profa. Regina Célia Coelho

Matrizes de Trabalho

- Matrizes de trabalho da OpenGL:
  - Imagem Corrente (GL_MODELVIEW);
  - Projeção (GL_PROJECTION);
  - Textura (GL_TEXTURE);

- void glMatrixMode(GLenum mode)

  define matriz de trabalho para as próximas operações;

  mode especifica qual matriz é a alvo das subsequentes operações. Três valores são aceitos: **GL_MODELVIEW**, **GL_PROJECTION**, e **GL_TEXTURE**. O valor inicial é **GL_MODELVIEW**.

Matrizes de Trabalho

- Exemplo:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity() ;
/* GL_MODELVIEW: significa que qualquer manipulação de matriz
   deste ponto em diante afetará a matriz do modelo de visão.
glLoadIdentity(): responsável por limpar a matriz atual a ser
   modificada de quaisquer transformações atribuindo a matriz
   identidade à matriz corrente */
```

Criação de Objetos 3D

- Até agora criamos apenas objetos bidimensionais.
- Podemos criar objetos utilizando *glvertex* da mesma forma que usamos para criar objetos bidimensionais, porém, agora, acrescentando mais uma coordenada:

```
glvertex3f(10,20,10)
```

- As bibliotecas GLU e GLUT possuem uma série de funções para desenhar esferas, cones, cilindros, *teapot*, etc.

Objetos 3D

glutWireSphere(radius: GIdouble; slices, stacks: GLint)
glutWireCube(size: GLdouble)
glutWireCone(radius, height: GIdouble; slices, stacks: GLint)
glutWireTorus(innerRadius, outerRadius: GIdouble; nsides, rings: GLint)
glutWireOctahedron(): cria um octaedro (polígono de 8 lados).
glutWireTetrahedron(): cria um tetraedro (uma pirâmide).
glutWireIcosahedron(): cria um icosaedro (polígono de 20 vértices).
glutWireTeapot(size: GLdoouble): desenha um *teapot* (bule de chá), sendo size o raio aproximado do *teapot*. Uma esfera com este raio irá "envolver" totalmente o modelo.
glutWireDodecahedron(): gera um dodecaedro (polígono de 12 lados).

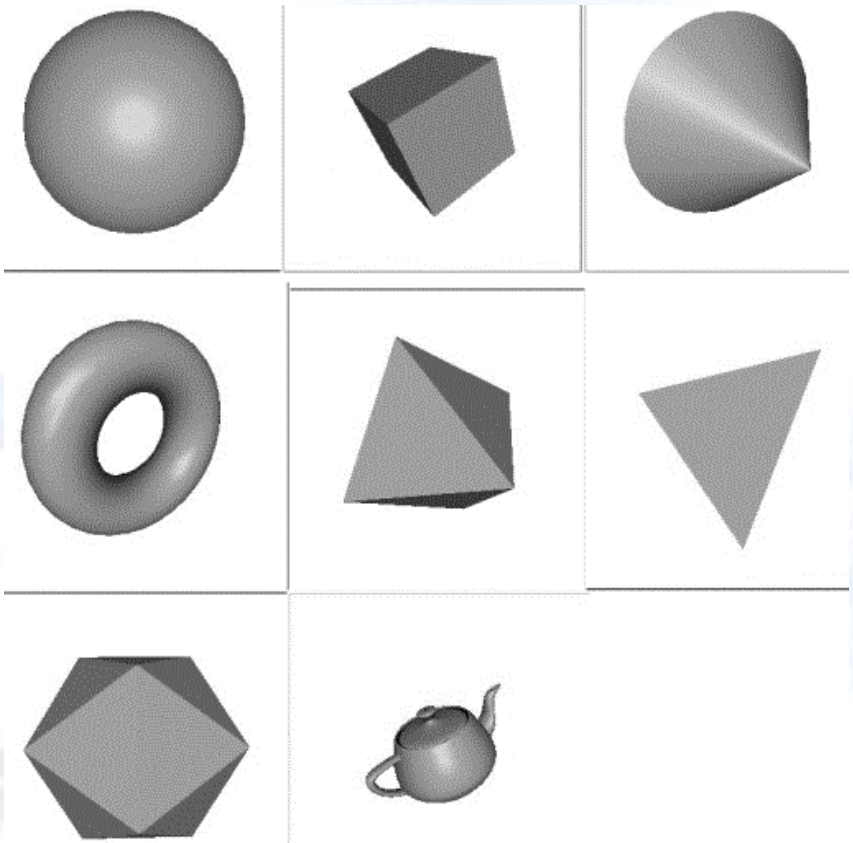
Criação de Objetos 3D

- Os parâmetros *slices* e *stacks* que aparecem no protótipo de algumas funções, significam, respectivamente, o número de subdivisões em torno do eixo z (como se fossem linhas longitudinais) e o número de subdivisões ao longo do eixo z (como se fossem linhas latitudinais).

- Os parâmetros *rings* e *nsides* correspondem, respectivamente, ao número de seções que serão usadas para formar o torus, e ao número de subdivisões para cada seção.

- Todas estas funções também podem ser usadas para desenhar objetos sólidos (*Solid*).

Objetos 3D



esfera, cubo, cone, torus, octaedro, tetraedro, icosaedro e *teapot*.

Transformações

- Sempre é utilizado o sistema de coordenadas da “regra da mão direita” (z saindo do papel)

- Translação:
  - glTranslatef(tx, ty, tz: GLdouble)

- Rotação:
  - glRotatef(Angulo, x, y, z: GLdouble)

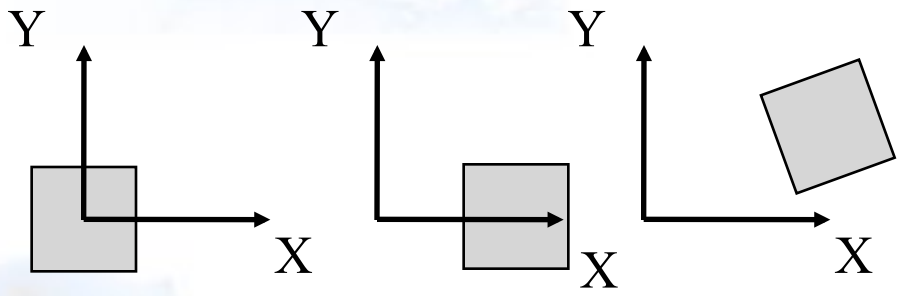
- Escala:
  - glScalef(ex, ey, ez: GLdouble)

Transformações

- São acumulativas, ou seja, podem ser aplicadas umas sobre as outras.

- Ordem inversa para especificação.

```
...
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(30,0,0,1);
glTranslatef(10,0,0);
...
```



Exercício

- Teste estas transformações para visualizar a forma resultante de cada uma das primitivas 3D apresentadas. Para visualizar melhor as primitivas, utilize o teclado (teclas x, y e z) para realizar rotações da sua figura em relação aos eixos x, y, z. Deixe apenas uma figura de cada vez na janela.

- Para as funções que desenham a esfera, o torus e o cone, crie e inicialize as seguintes variáveis globais:

  GLint rings = 6, nsides = 20, slices = 20, stacks = 10;

- Acrescente no seu código os comandos necessários para incrementar e decrementar o valor destas variáveis. Por ex, F1 incrementa e F2 decrementa a variável *rings, F3 e F4* controlam a variável *nsides,* e assim por diante.

Transformações

- Para permitir que a transformação atual seja reinicializada há o comando glLoadIdentity().

```
DesenhaObjeto();
glScalef(1.0,0.5,1.0);
DesenhaObjeto(); //diminui a altura do objeto pela metade

glLoadIdentity(); // reinicializa as transformações
glScalef(1.0,2.0,1.0);
DesenhaObjeto(); // desenha o outro objeto com o dobro
                 //  do tamanho original
```

Transformações

- Para permitir que uma transformação valha somente em um certo trecho de programa e assim não altere o que está sendo desenhado depois, há os comandos glPushMatrix() e glPopMatrix().
- A ideia é que o glPushMatrix armazene as transformações atuais em um pilha interna do OpenGL e que estas transformações possam ser retiradas depois por um glPopMatrix.

Transformações

- Manipulação de pilhas de matrizes de transformação

```
…
glMatrixMode(GL_MODELVIEW);
glLoadIdentity( );
glPushMatrix( );
glTranslatef(10.0,0.0,0.0);
glRotatef(30.0,0.0,0.0,1.0);
draw_object_1( );
glPopMatrix( );
…
```

Transformações

```
DesenhaObjeto();
glPushMatrix();  // salva as transformações
                 //  atuais na pilha
glScalef(1.0,0.5,1.0);
DesenhaObjeto();  // diminui a altura do objeto à
                  // metade do original
glPopMatrix();  // restaura as transformações
                // anteriores
glScalef(1.0,2.0,1.0);
DesenhaObjeto(); // desenha o outro objeto com
                 // o dobro do tamanho original
```

Exemplo

```
void display(void){
   glClearColor (0.0, 0.0, 0.0, 0.0);
   glClear (GL_COLOR_BUFFER_BIT);

// a escala será aplicada aos dois cubos
   glScalef(0.5,0.5,0.5);
   glColor3f(1.0, 0.0, 0.0);
   glutWireCube (0.5);

   glColor3f(0.0, 1.0, 0.0);
   glTranslatef (0.5, 0.0, 0.0);
   glRotatef (45, 0.0, 0.0, 1.0);
   glutWireCube (0.5);
   glFlush();
}
```

Teste este exemplo verificando o que ocorre quando é realizada uma rotação e uma translação em sequências diferentes e, a seguir, utilizando o **glPushMatrix( )** e o **glPopMatrix( )**

Animação

- Em uma animação a primeira coisa que temos que pensar é na rápida atualização da cena.

- Para isso, utilizaremos a opção *double-buffer* no parâmetro da função `glutInitDisplayMode`. Isso possibilita que uma imagem seja mostrada apenas após a sua criação e permitirá que as sucessivas renderizações sejam feitas de modo suave, sem o efeito indesejável de "piscar" entre cada atualização da janela de visualização.

- A troca de *buffers* é feita simplesmente chamando a função `glutSwapBuffers()` depois da criação de uma cena (não precisa de nenhum parâmetro).

Manipulação da área de projeção

- gluOrtho2D: responsável por estabelecer a área de projeção bidimensional da janela (ou a janela de recorte). A matriz de projeção *default* é a ortogonal 3D com limites de –1 a 1 em todas as 3 dimensões. Esta função é usada para transformar a projeção para bidimensional (estabelecendo os limites de *z* para –1 e 1).
- Para utilizar funções da biblioteca glu inclua –lGLU (maiúsculo) Linker stttings → Other linker options.

Transformação Janela – Porta de Visão

- Define um retângulo de pixel na janela na qual a imagem final será desenhada.

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

  - x, y especifica o canto inferior esquerdo da viewport;
  - width, height são a largura e altura do viewport.
  - Os valores de viewport iniciais por padrão são (0, 0, winWidth, winHeight), sendo que winWidth e winHeight são o tamanho da janela.

Manipulação da área de projeção

- Uma função que permite a manipulação da área de projeção é a função:

  glutReshapeFunc(name).

- Ela especifica a função que será chamada se a janela for movida ou alterada de tamanho. Ela também é usada para iniciar o tipo de projeção.

- É interessante também definirmos nesta função o sistema de coordenadas e o limite de desenho na janela.

Manipulação da área de projeção

- Modificando levemente a função *main* afim de incluir a chamada da subrotina glutReshapeFunc (reshape) (com o argumento reshape) depois da chamada para a glutDisplayFunc, os resultados do programa será o aparecimento da figura contida na função desenha.

- No caso de um quadrado, por exemplo, para que não haja distorção para um retângulo quando a janela tiver seu tamanho alterado pode ser incluído o comando *if* na função reshape para escolher os parâmetros da função glViewport().

Manipulação da área de projeção

```
void reshape(int w, int h)
{
   if (w >= h)
     glViewport(0, 0, (GLsizei)h, (GLsizei)h) ;
   else
     glViewport(0, 0, (GLsizei)w, (GLsizei)w) ;

   glMatrixMode(GL_PROJECTION) ;
   glLoadIdentity() ;
   gluOrtho2D(0.0,1.0,0.0,1.0) ;
   glMatrixMode(GL_MODELVIEW) ;
   glLoadIdentity() ;
}
```

- Quando for chamada, a GLUT passará dois argumentos que são a largura e altura da janela em *pixels*.

Exercício

- Crie uma cena contendo pelo menos 2 objetos diferentes (o bule e o cubo, por exemplo) e faça uma animação utilizando transformações geométricas 3D. Use teclado e/ou mouse para disparar a animação. Utilize as 3 transformações no seu programa. Insira nesta animação o uso da função glutReshapeFunc.

Exercício

- Faça um programa que crie um cachorro 3D que andará no seu Canvas (janela de desenho). Faça um ambiente em que o cachorro sempre seguirá o mouse. Conforme movimentamos o mouse, ele deverá segui-lo, sem deixar que ele saia da janela. Se o mouse sair da janela, ele deverá ficar parado. O cachorro deverá andar somente de frente e as rotações necessárias deverão ser suaves (deverá parecer que ele girou para ir na direção do mouse, não que ele tenha pulado).

Exercício - continuação

- A tecla ‘S’ ou ‘s’ deverá escalar o cachorro, sendo ‘S’ para aumentar e ‘s’ para diminuir. Coloque um limite máximo para aumentar e diminuir, de forma que o cachorro não fique excessivamente grande ou pequeno.

- **Lembre-se de que o cachorro possui 4 patas**. Use sua criatividade neste trabalho para tentar conseguir até um ponto extra. Não é necessário que os movimentos das patas sejam totalmente realistas e que possuam várias partes, pois ainda não foi vista a aula de hierarquia que permite um movimento realista às patas.

- **Entrega: 20/09 até às 23h55 no horário do Moodle.**