Modelo Hierárquico & Transformações Geométricas

Universidade de São Paulo – USP
Disciplina de Computação Gráfica
Profª Maria Cristina
PAE: Thiago Silva Reis Santos
Setembro de 2010

Sumário

- Modelagem
- Objetos Predefinidos da GLUT
- Modelo Hierárquico
- Robô
- Pilha de Matrizes
- Estrutura de Dados: Árvore
- Travesia - Pré-ordem
- Movimento
- Bibliografia

Modelagem

- Modelos são abstrações do mundo.
- Estamos acostumados com modelos matemáticos, que são usados em várias áreas da ciência e das engenharias.
- Esses modelos usam equações matemáticas para modelar o fenômeno físico o qual se pretende estudar.

Modelagem

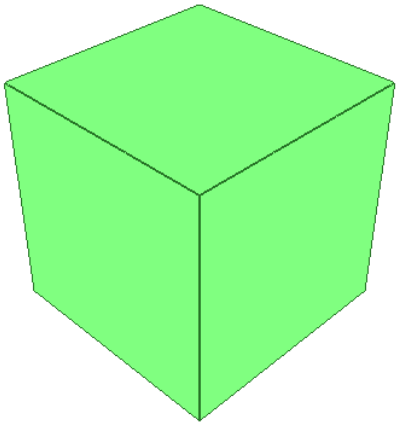
- Em Ciência da Computação, nós usamos tipos de dados abstratos para modelar organizações de objetos, e em computação gráfica, nós modelamos o mundo com objetos geométricos.
- Nós escolhemos cuidadosamente as primitivas que irão compor nosso modelo e o relacionamento entre elas.

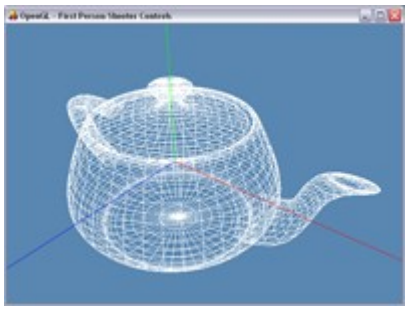
Modelo Hierárquico

- Na criação de modelos hierárquicos temos que resolver duas questões: Como definimos um objeto mais complexo do que os usados até aqui? E Como podemos armazenar esse modelo que pode conter muitos objetos?
- A resposta da 1ª pergunta diz respeito a criação do modelo em si, o qual, geralmente é composto por vários objetos geométricos, tipicamente as funções predefinidas da GLUT.

Objetos Predefinidos da GLUT

- Void glutWireCube(GLdouble size)
  - Desenha um cubo *wireframe* (aramado) com o tamanho de size.

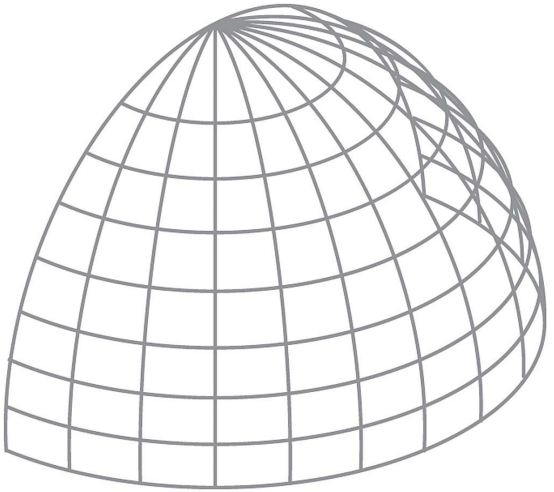


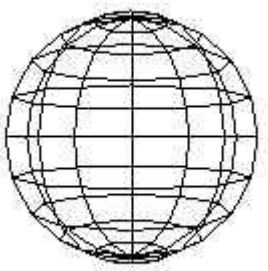


- glutWireTeapot(GLdouble size)
  - Desenha um bule de chá *wireframe* com o tamanho de size.

Objetos Predefinidos da GLUT

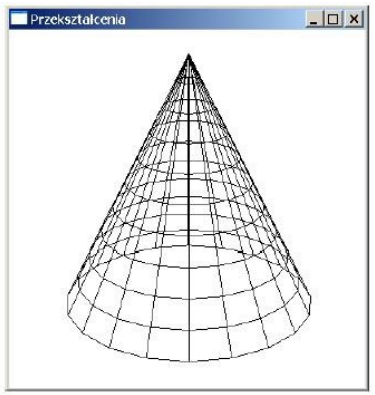
- void glutWireSphere(GLdouble radius, Glint slices, Glint stacks)
  - Desenha uma esfera wireframe. Parâmetros:
  - Radius: O raio da esfera
  - Slices: linhas longitudinais (horizontais)
  - Stacks: linhas latitudinais (vertivais)

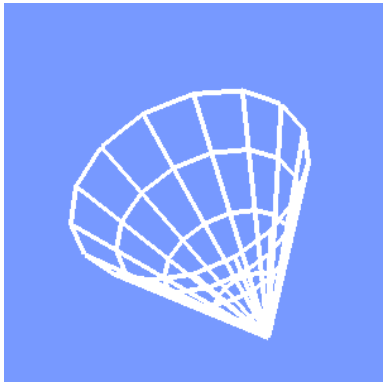




Objetos Predefinidos da GLUT

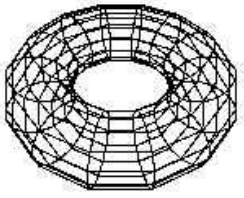
- void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks)
  - Desenha um cone *wireframe*. Parâmetros:
  - Radius: O raio da esfera
  - Height: altura do cone
  - Slices: linhas longitudinais (horizontais)
  - Stacks: linhas latitudinais (vertivais)

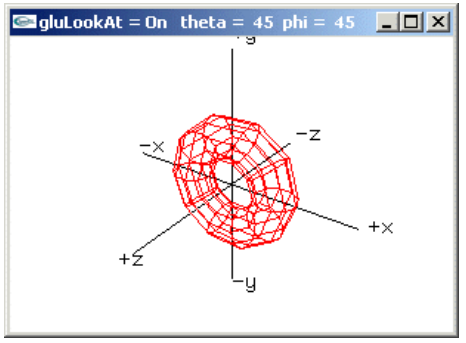




Objetos Predefinidos da GLUT

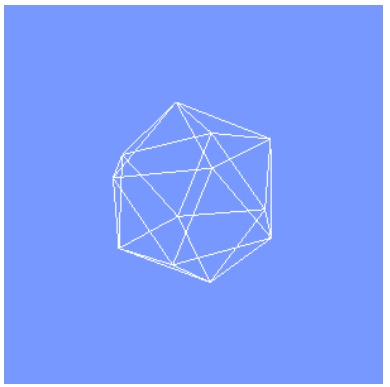
- void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nslices, GLint rings)
  - Desenha um *torus wireframe*. Parâmetros:
  - InnerRadius: Raio interno
  - OuterRadius: Raio externo
  - Rings: Número de seções para formar o torus
  - NSlices: Número de divisões de cada seção



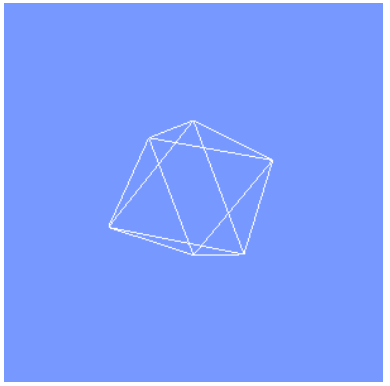


Objetos Predefinidos da GLUT

- void glutWireIcosahedron(void)
  - Desenha um icosaedro *wireframe* de tamanho predefinido.

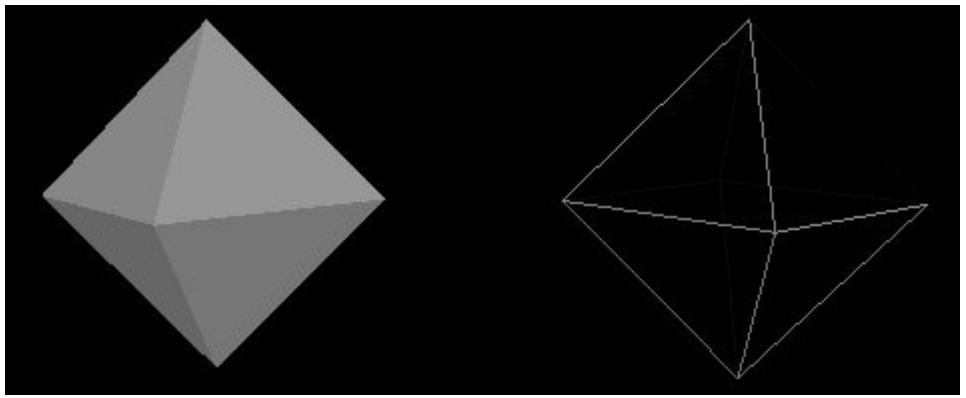


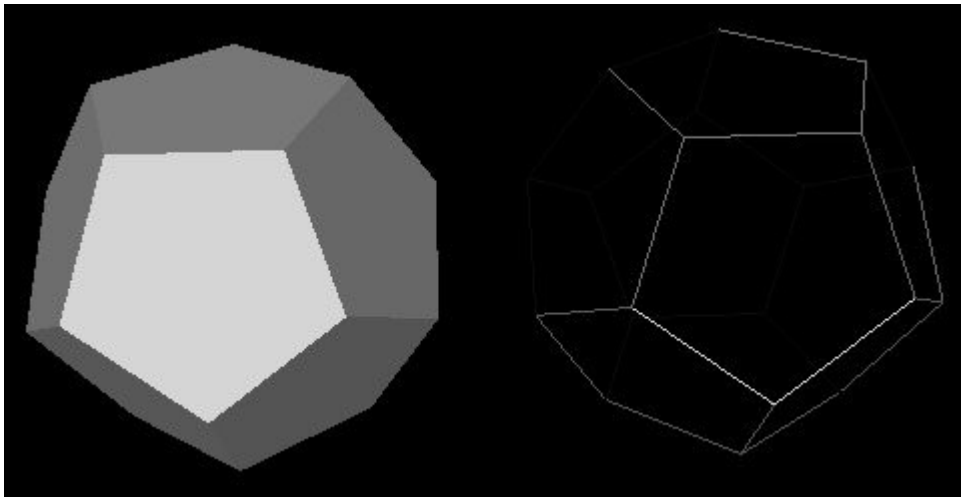
- void glutWireOctahedron(void)
  - Desenha um octaedro *wireframe* de tamanho predefinido.



Objetos Predefinidos da GLUT

- void glutWireTetrahedron(void)
  - Desenha um tetraedro *wireframe* de tamanho predefinido.
- void glutWireDodecahedron(void)
  - Desenha um dodecaedro *wireframe* de tamanho predefinido.





Objetos Predefinidos da GLUT

- Todas essas funções também podem ser usadas para desenhar objetos sólidos. Basta substituir a palavra "*Wire"* no nome da função por "*Solid"*. Ex:
  - gultSolidTorus
  - glutSolidCone

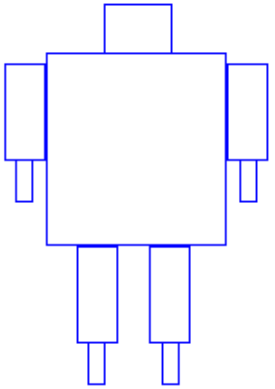
Modelo Hierárquico

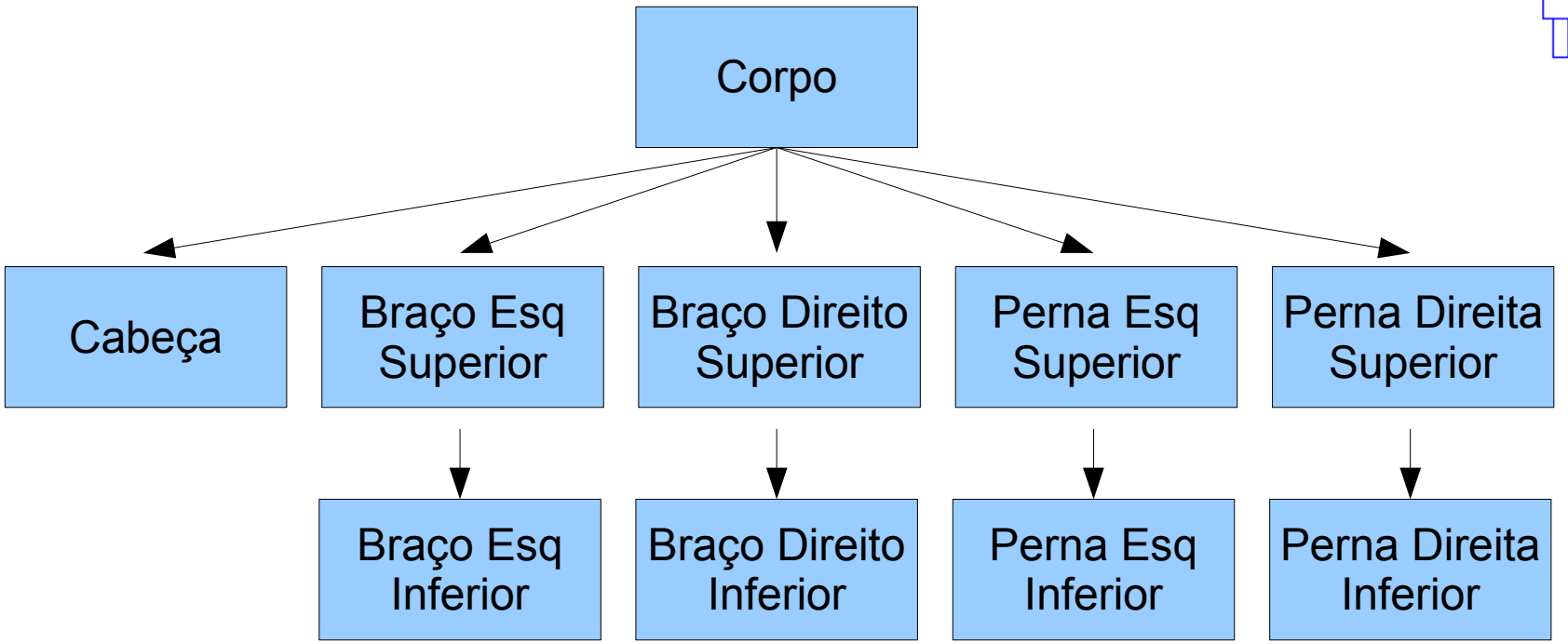
- A resposta da 2ª pergunta diz respeito ao modelo ao modelo hierárquico de dados que iremos utilizar para compor o modelo.

- A idéia do modelo hierárquico é modelar o objeto em questão numa estrutura hierárquica, tipicamente uma árvore.

Robô

- Vamos pegar como exemplo um robô. Como fazer uma estrutura hierárquica de um modelo de robô?





Pilha de Matrizes

- Agora que está feito a estrutura do nosso modelo hierárquico fica mais fácil sua implementação.
- Cada nó da árvore é responsável por desenhar um pequeno pedaço do robô.
- Entretanto, cada nó deve possuir uma matriz de transformação a qual irá posicionar corretamente a sua pequena contribuição do robô.

Pilha de Matrizes

- Iniciamos pelo corpo, o qual será aplicado a matriz M que estará no topo da pilha *model-view*, depois de feita a devida transformação é chamado o método que desenha torso().
- Em seguida vamos para a cabeça, a qual será aplicada uma matriz de transformação MM_{head} que posicionará corretamente a cabeça que será desenhada pelo método head(). Após esse nó desempilha-se a matrix MM_{cab} e deixa-se apenas M.

Pilha de Matrizes

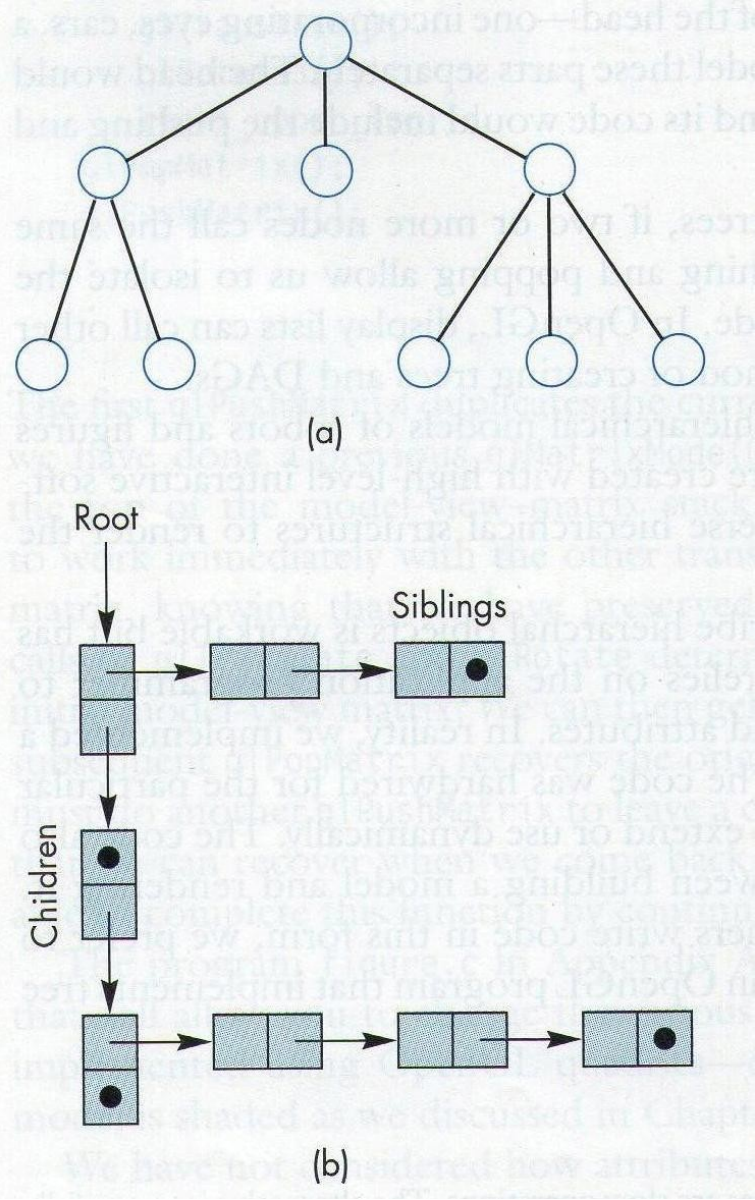
- O próximo passo será o braço superior esquerdo aplicando a matrix MM_{lua} e depois chamando o método que desenha o braço que é left_upper_arm().
- Em seguida será o braço inferior esquerdo, aplicando a matrix $MM_{lua}M_{lla}$ e depois chamando o método que desenha o braço que é left_lower_arm(). Após esse nó iremos desempilhar a matrix $MM_{lua}M_{lla}$ e deixar apenas M.

Pilha de Matrizes

```
glPushMatrix();   //Duplica o valor do topo da pilha
    torso();
    glTranslate( xxxx );
    glRotate3( xxxx );
    head();
glPopMatrix();   // Recupera o valor anterior da pilha
glPushMatrix();   //Duplica o valor do topo da pilha
    glTranslate( xxxx );
    glRotate3( xxxx );
    left_upper_arm();
    glTranslate( xxxx );
    glRotate3( xxxx );
    left_lower_arm();
glPopMatrix();   // Recupera o valor anterior da pilha
glPushMatrix();   //Duplica o valor do topo da pilha
     ...
```

Estrutura de Dados: Árvore

- Para implementar o modelo hierárquico iremos utilizar uma estrutura de árvore otimizada, conforme a imagem ao lado.



Estrutura de Dados: Árvore

- Utilizaremos um struct com 4 campos:
  - A matriz de transformação → m[16] // 4x4
  - Um apontador para a função que desenha → f
  - Um apontador para o irmão (a direita) → sibling
  - Um apontador para o filho (a esquerda) → child

```
type struct treenode {
    GLfloat m[16];
    void (*f)();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
```

Estrutura de Dados: Árvore

- Quando for renderizar um nó, primiero mutiplicamos a matriz do topo da *model-view* a matriz armazenada em m. Depois chamamos a função f que contém as primitivas gráficas.
- Em seguida renderizamos os filhos e por útimo os irmãos.
- No nosso exemplo do robô definimos 10 nós da árvore.
  - torso_node, head_node, lua_node, rua_node, lll_node, rll_node, lla_node, rla, rul e lul_node

Estrutura de Dados: Árvore

- Definimos o torso_node assim:
  - O corpo do robô pode se movimentar em torno do eixco-y, então a matriz de transformação m será:
    - glLoadIdentity();
    - glRotatef(angulo, 0.0, 1.0, 0.0);
    - glGetFloatv(GL_MODELING_MATRIX, torso_node.m);
  - A função com as primitivas gráficas
    - torso_node.f = torso;
  - Os apontadores
    - torso_node.sibling = NULL; //Não tem irmãos
    - torso_node.child = &head_node;

Estrutura de Dados: Árvore

- Definimos o nó do braço esquerdo assim:
  - O braço do robô pode se movimentar em torno do eixco-x, porém o braço é deslocado em relação ao corpo:

```
glLoadIdentity();
glTranslatef(-(TORSO_RADIUS+UPPER_ARM_RADIUS),
                    0.9*TORSO_HEIGHT, 0.0);
glRotatef(theta[3], 1.0, 0.0, 0.0);
glGetFloatv(GL_MODELVIEW_MATRIX,lua_node.m);
lua_node.f = left_upper_arm;
lua_node.sibling =  &rua_node;
lua_node.child = &lla_node;
```

Travesia - Pré-ordem

- Uma vez pronta a árvore o rendering do modelo é feito por uma travesia na árvore, como no algoritmo abaixo.

```
void traverse(treenode* root) {
   if(root==NULL) return;
   glPushMatrix();
   glMultMatrixf(root->m);
   root->f();
   if(root->child!=NULL) traverse(root->child);
   glPopMatrix();
   if(root->sibling!=NULL) traverse(root->sibling);
}
```

Travesia - Pré-ordem

- O algoritmo de travesia é completamente independente do modelo (da árvore), então pode ser usado na função callback display() para gerar a visualização do objeto.

```
void display(void){

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glColor3f(1.0, 0.0, 0.0);
    traverse(&torso_node);
    glutSwapBuffers();
}
```

Movimento

- O robô possui 9 pontos de junção que podem ser movimentados e mais o corpo do robô.
- O movimento aconteçe com o botão direito e esquerdo do mouse. Para selecionar o qual ponto de junção se pretende mover basta utilizar o terceiro butão do mouse e escolher no menu.
- O movimento se dá com a alteração do valor do ângulo da matrix m, onde cada nó da árvore está associado a um ângulo.

Movimento

- O movimento está condicionado aos clicks do mouse e a seleção do ponto de junção ao qual se pretende movimentar.

```
void mouse(int btn, int state, int x, int y) {
      if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
       theta[angle] += 5.0;
       if( theta[angle] > 360.0 ) theta[angle] -= 360.0;
       }
      if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
       theta[angle] -= 5.0;
       if( theta[angle] < 360.0 ) theta[angle] += 360.0;
       }
   ....
   //aplicação do novo ângulo ao ponto de junção selecionado
}
```

Bibliografia

- ANGEL, Edward. Interactive Computer Graphics: A Top-Down Approach Using OpenGL. 4ª edição. Editora: Addison Wesley - Boston, San Francisco, New York, 2005.