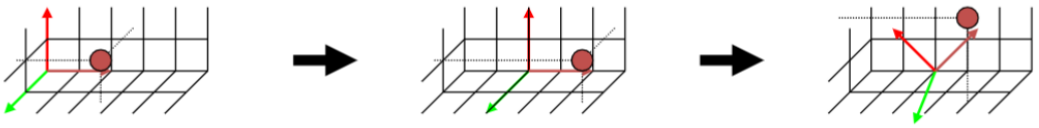
CPT205W08_层次化建模

本地&世界坐标参考系（frames of reference）

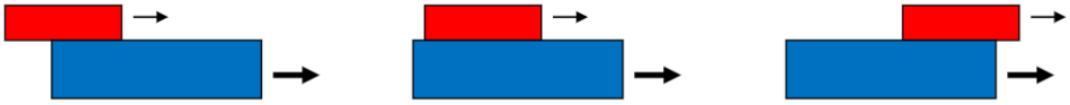
我们可以移动/旋转原点(0,0,0)，从而移动/旋转 相对于该点定义的所有点



> Local basis = Local basis = Local / model frame of reference

每个对象都有一个与世界坐标系（world frame）相关的本地参考系。

如果物体相对于局部原点发送运动，则产生相对运动。

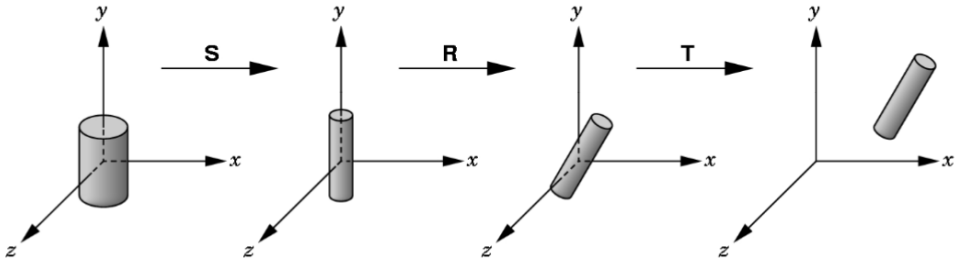


本地原点当然可以相对于更大的参考系移动。

线性建模（Linear modelling）

对象在场景中的每次出现都是一个实例，必须对其进行缩放（scale），定向（orient），定位（position）来定义实例的变换。

$$M = T \cdot R \cdot S$$



在Opengl中，我们将模型坐标model frame设置合适的变换到世界坐标world frame。

在执行代码前应用MODELVIEW矩阵：

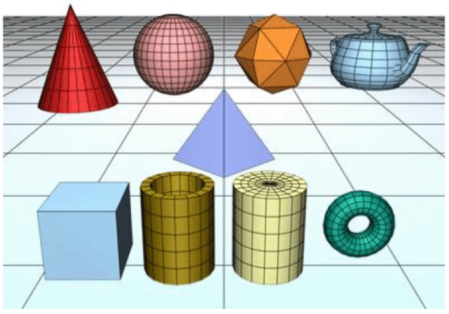
```
glMatrixMode(GL_MODELVIEW); // M = T·R·S
glLoadIdentity();
glTranslatef();
```

```
glRotatef();
glScalef();
glutSolidCylinder()          // or other symbol
```

例：生成圆柱体

```
glBegin(GL_QUADS);
For each A = Angles{
    glVertex3f(R*cos(A), R*sin(A), 0);
    glVertex3f(R*cos(A+DA), R*sin(A+DA), 0);
    glVertex3f(R*cos(A+DA), R*sin(A+DA), H);
    glVertex3f(R*cos(A), R*sin(A), H);
}
glEnd();
// Make Polygons for Top/Bottom of cylinder
```

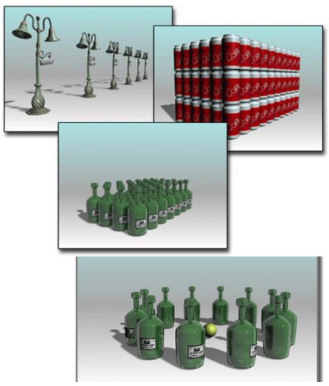
- 符号（Symbol，Primitives）：圆锥，球体，GeoSphere，茶壶，盒子，管状体，圆柱体，圆环等（Cone, Sphere, GeoSphere, Teapot, Box, Tube, Cylinder, Torus, etc.）



- 拷贝（Copy）：创建与原始物体完全独立的克隆clone，对拷贝的修改不会影响原物体。
- 实例（Instance）：创建与原始物体可完全互换的（interchangeable）克隆，对实例的修改等于修改原始物体。
- 阵列（Array）：一系列克隆

  线性：选择对象，定义轴axis，定义距离distance，定义数量number

  径向：选择对象，定义轴axis，定义半径radius，定义数量number



模型储存在表中：为每个符号分配一个编号并储存实例转换的参数。

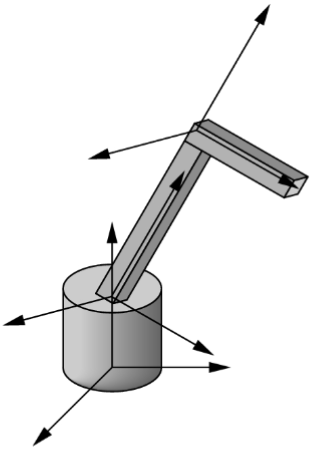
包含平面信息Flat Information，但没有实际结构的信息。

如何使用约束表示复杂的结构：每个部分有自己的坐标系模型框架，但没有关系的信息。

如何使用子结构（substructures）进行操作?

Symbol	Scale	Rotate	Translate
1	s_x, s_y, s_z	u_x, u_y, u_z	d_x, d_y, d_z
2			
3			
1			
1			
.			
.			

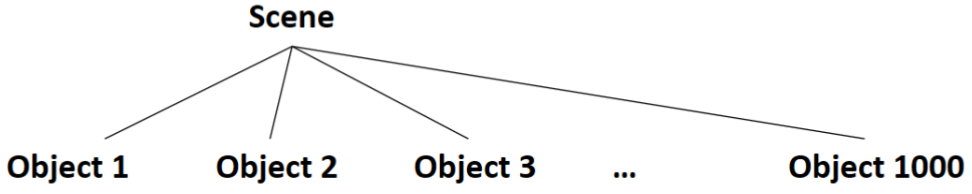
Linear model table



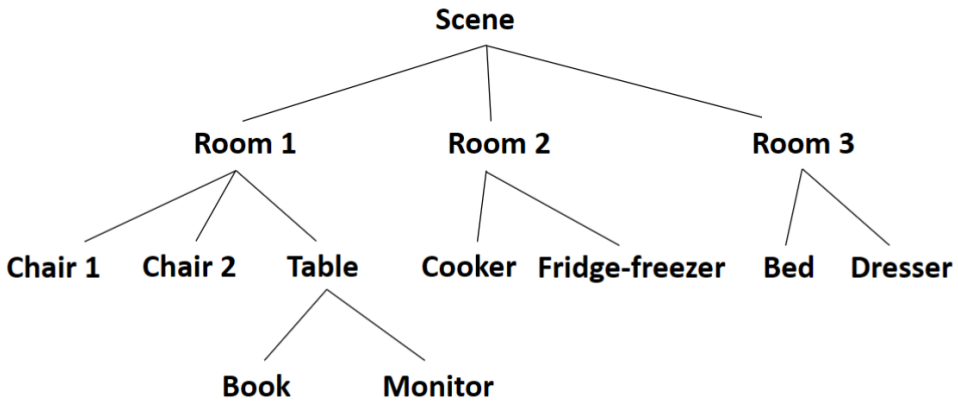
Model with constraints

场景层次结构（Scene hierarchy）

一个含有大量对象的场景，除了简单的单层组织结构，还可以有层次结构



简单单层结构



层次分组（hierarchical grouping）

在一个场景中，一些对象可能以某种方式组合在一起。例如，关节图形可能包含多个以指定方式连接在一起的刚性组件。

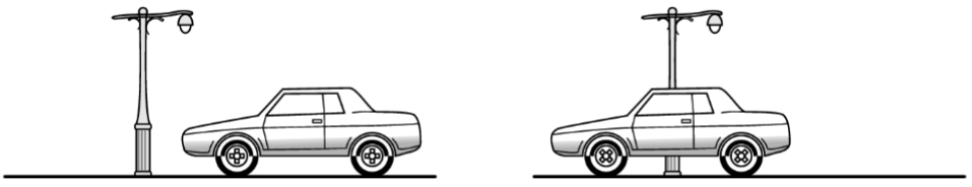
如：几个物体放在托盘上，围绕太阳系运行的行星卫星，有200间客房的酒店，每间客房都有床桌子椅子等

在这些情况下，当考虑物体位置和方向时，物体位置更容易描述

分层模型（Hierarchical models）

汽车的速度实际上是由车轮的转速决定的，或反之亦然。

```
void main ( );{
    float s = ...; // speed
    float d[3] = {...}; // direction
    draw_right_front_wheel(s,d);
    draw_left_front_wheel(s,d);
    draw_right_rear_wheel(s,d);
    draw_left_rear_wheel(s,d);
    draw_chassis(s,d);
} // WE DO NOT WANT THIS!
```



Two frames of reference for animation

分层树（Hierarchical tree）

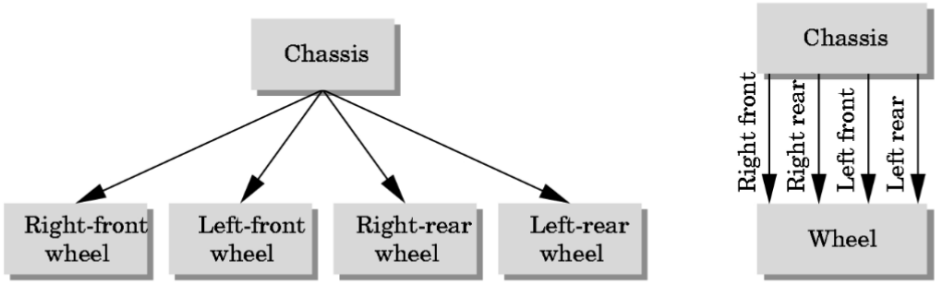
各个对象被分组到一个层次结构中，该层次结构由树结构（倒置树）表示。

- 每个移动部件都是树中的单个节点。

- 顶部的节点是根节点。
- 每个节点（根除外）只有一个父节点，该父节点位于其正上方。
- 一个节点下面可能有多个子节点。
- 具有相同父节点的节点称为同级节点。
- 树底部没有子节点的节点称为叶节点。

直接无环图 （Direct Acyclic Graph， DAG） 存储每个轮子的位置。

树和 DAG：分层方法表示关系。



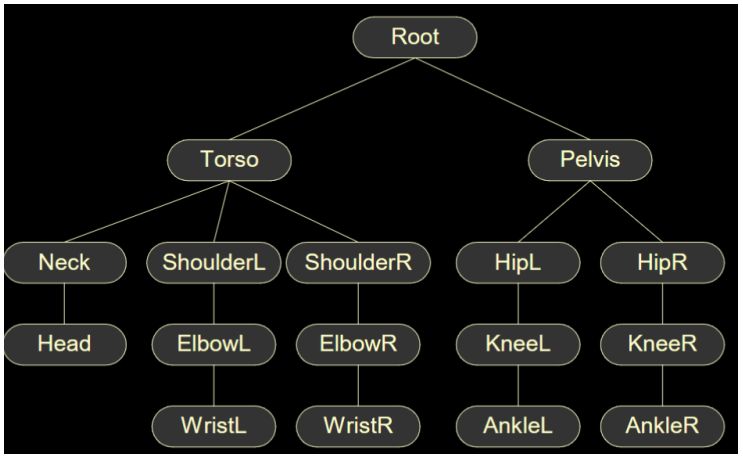
铰接式模型（Articulated model）

铰接模型是由刚性部件和连接接头组成的分层模型的一个例子。

如果我们选择某个特定的部分作为 'root'，那么移动的部分可以被安排成一个树状的数据结构。

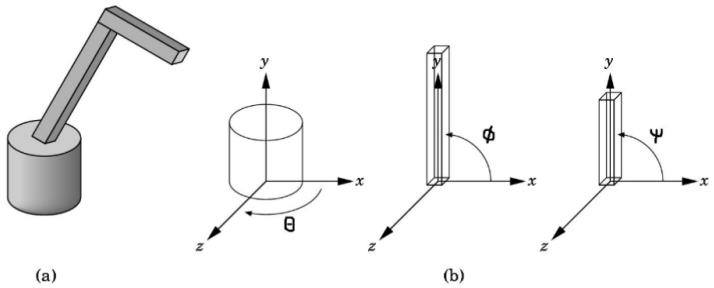
对于关节模型（如两足动物角色），我们通常选择根部位于躯干中心附近的某个位置。

图中的每个关节都有特定的允许自由度 （degrees of freedom DOF），用于定义模型的可能姿势范围。



例子：机械臂

可以通过指定所有关节角度来指定模型的状态。



(a) (b)

Robot arm **Parts in their own frames of reference**

分析自由度：底座水平旋转；下臂相对底座平移，围绕关节旋转；上臂相对下壁平移，围绕关节旋转

- Base transformation
  - Rotate the base: $\mathbf{R_b}$
  - Apply $\mathbf{M_{b\text{-}w} = R_b}$ to the base
- Lower arm transformation
  - Translate the lower arm relative to the base: $\mathbf{T_{la}}$
  - Rotate the lower arm around the joint: $\mathbf{R_{la}}$
  - Apply $\mathbf{M_{la\text{-}w} = M_{b\text{-}w} \cdot M_{la} = R_b \cdot T_{la} \cdot R_{la}}$ to the lower arm
- Upper arm transformation
  - Translate the upper arm relative to the lower arm: $\mathbf{T_{ua}}$
  - Rotate the upper arm around the joint: $\mathbf{R_{ua}}$
  - Apply $\mathbf{M_{ua\text{-}w} = M_{la\text{-}w} \cdot M_{ua} = R_b \cdot T_{la} \cdot R_{la} \cdot T_{ua} \cdot R_{ua}}$ to the upper arm

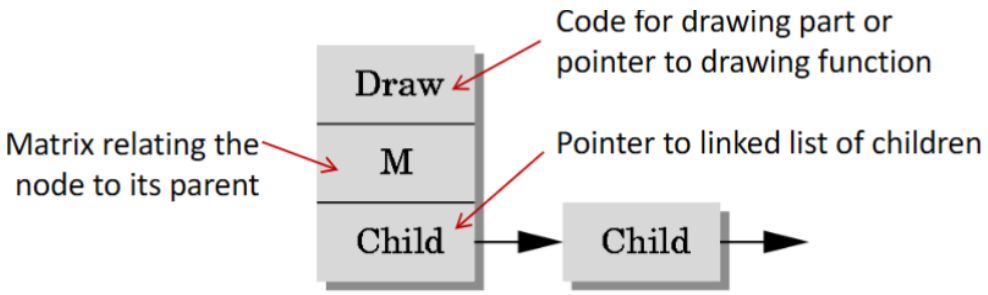
3个部分每个都有一个DOF，即关节角度

```
void display(){
    glRotatef(theta, 0.0, 1.0, 0.0);
    base();
    glTranslatef(0.0, h1, 0.0);
    glRotatef(phi, 0.0, 0.0, 1.0);
    lower_arm();
    glTranslatef(0.0, h2, 0.0);
    glRotatef(psi, 0.0, 0.0, 1.0);
    upper_arm();
}
```

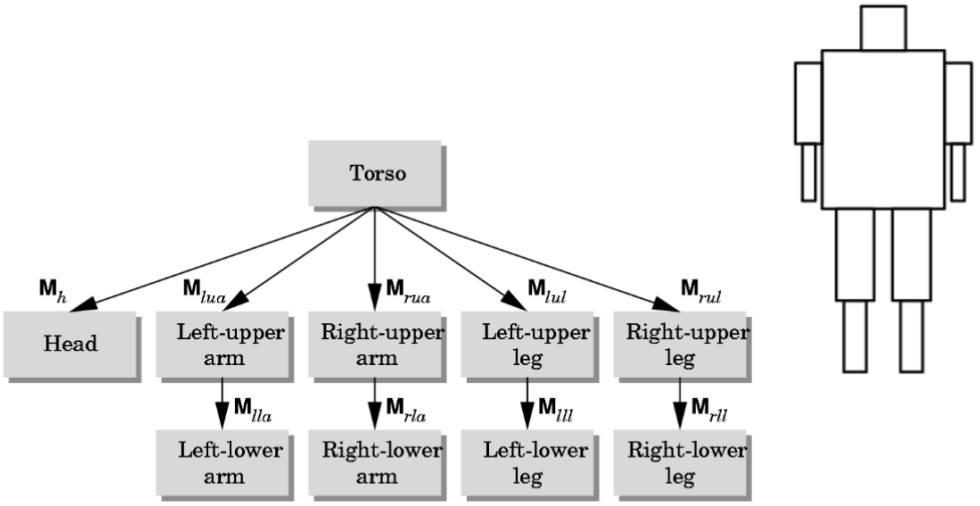
上臂的MODELVIEW矩阵： $\mathbf{M_{ua\text{-}w} = R_b(\theta) \cdot T_{la}(h1) \cdot R_{la}(\phi) \cdot T_{ua}(h2) \cdot R_{ua}(\psi)}$

如果信息存储在节点中（而不是边），则每个节点必须至少存储：

- 指向绘制节点所表示对象的函数的指针
- 一个矩阵，用于相对于节点的父对象（包括其子对象）进行定位、定向和缩放。
- 指向其子项的指针。



例子：人体模型



遍历与显示：

图形的位置由 11 个关节角度决定。树的显示可以被认为是一个图遍历。

访问每个节点一次。在描述与节点关联的零件的每个节点上执行 display 函数，为位置和方向应用正确的变换矩阵。

所有矩阵都是增量矩阵，可以使用任何遍历算法（深度优先或广度优先）。我们可以先从左到右深度遍历。

在代码中执行显式遍历，使用堆栈来存储所需的矩阵和属性。递归遍历代码更简单，矩阵和属性的存储是隐式的。

```
void figure() {
    torso();

    glPushMatrix(); // save present MODELVIEW matrix
    glTranslatef(); // update MODELVIEW matrix for the head
    glRotate3();
    head();
    glPopMatrix(); // recover MODELVIEW matrix for the

    // torso and save the state
    glPushMatrix();
    glTranslatef(); // update MODELVIEW matrix for
```

```
    glRotate3(); // the left upper leg
    left_upper_leg();

    glTranslatef();
    glRotate3(); // incremental change for
    left_lower_leg(); // the left_lower_leg
    glPopMatrix(); // recent state recovery
    ...;
}

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

// for the torso
glRotatef(theta[0], 0.0, 1.0, 0.0);
glGetFloatv(GL_MODELVIEW_MATRIX, torso_node.m);
// matrix elements copied to the M of the node

// the torso node has no sibling; and
// the leftmost child is the head node

// rest of the code for the torso node
torso_node.f = torso;
torso_node.sibling = NULL;
torso_node.child = &head_node;

// for the upper-arm node
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);
// matrix elements copied to the m of the node

lua_node.f = left_upper_arm;
lua_node.sibling = &rua_node;
lua_node.child = &lla_node;

// assumption MODELVIEW state
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);
} // traversal method is independent of the
// particular tree!
```

我们必须先保存模型视图矩阵 （glPushMatrix），然后再将其乘以节点矩阵。更新的矩阵适用于节点的子节点但不是它的兄弟姐妹，它们包含自己的矩阵;因此，在遍历同级之前，我们必须返回到以前的状态（glPopMatrix） 。如果我们要更改节点中的属性，我们可以在渲染函数中推送 （glPushAttrib） 和 pop （glPopAttrib） 属性，或者在推送模型视图矩阵时推送属性。

```
// generic display callback function
void display(void){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    traverse(&torso_node);
    glutSwapBuffers();
}
```

然后可以通过鼠标或键盘控制关节角度（即递增或递减）来实现动画。

总结：

线性建模没有提供有效的方法来保持模型对象之间的关系。通过分层建模，可以有效地创建和操作实际应用的复杂模型。

分层结构树用于在计算机图形学中实现分层模型。人形图形的代码是用 C 语言编写的（使用 Struct），用 C++ 编写会更有效（使用 Class）。