

2018-2019年度第二学期 00106501

计算机图形学



童伟华 管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学 数学科学学院

<http://math.ustc.edu.cn/>





第四节 层次建模

计算机图形学中的模型



■ 模型是世界的抽象

- 这里的世界既可以是真实的，也可以是虚拟的
- 模型通常是用它的数学方程表示的

■ 在计算机科学中应用抽象数据类型模拟对象的组织

■ 在计算机图形学中的建模主要包括

- 几何对象：物体的几何表示等
- 材质模型：反射系数或BRDF等
- 光源模型：光源颜色、光源的几何属性等
- 虚拟照相机

模型的数学表示

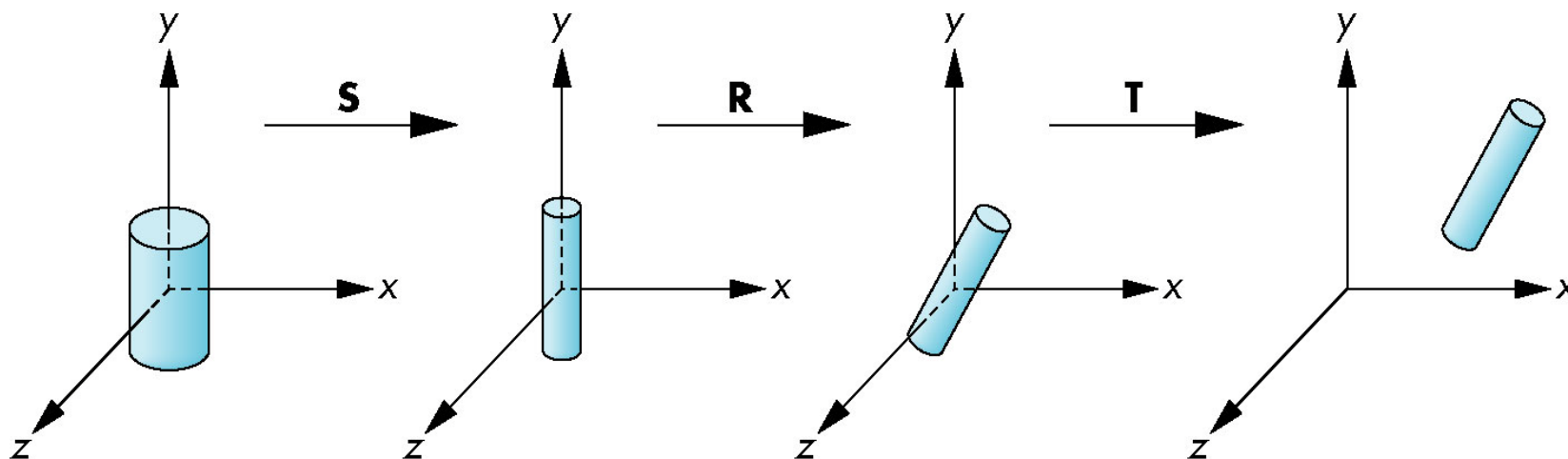


- 当建立一个数学模型时，需要仔细确定采用哪种数学表示
 - 根据所需要模拟的现象进行选择
 - 例：常微分方程适用于模拟弹簧质点系统的行为；而偏微分方程则适用于模拟流体的行为
- 在计算机图形学中关键在于几何对象的表示方式以及对象之间关系的反映

实例变换



- 从一个原型对象（称为一个符号, symbol）开始
- 在模型中对象的出现称为一个实例 (instance)
 - 需要进行放缩、定向、定位
 - 定义实例的变换



符号-实例表



- 存贮模型的方法是：给每个符号赋一个编号，并存贮实例变换的参数

符号	放缩	旋转	平移
1	$s_{x'}, s_{y'}, s_z$	$u_{x'}, u_{y'}, u_z$	$d_{x'}, d_{y'}, d_z$
2			
3			
1			
1			
.			
.			

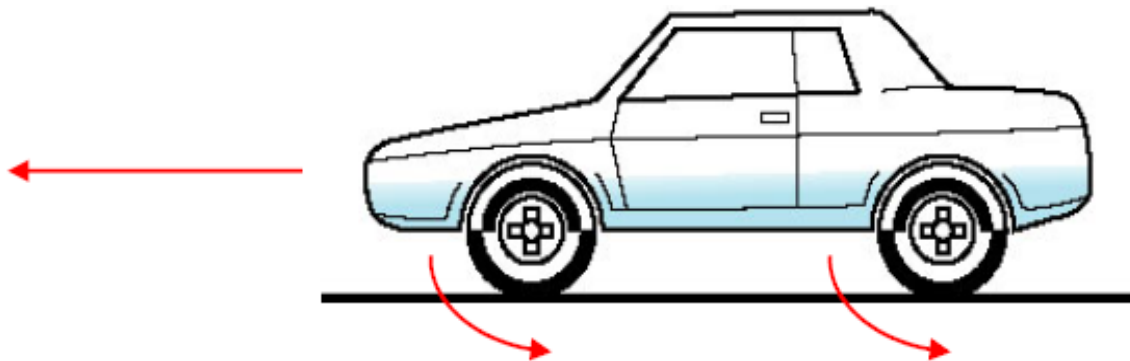
汽车模型的关系



■ 符号-实例表并没有显示出来模型各部分之间的关系

■ 考虑汽车的模型

- 车身 + 四个一样的车轮
- 两个符号



■ 向前运动的速度由车轮的旋转速度确定的

由函数调用反映结构

- 线性结构：没有很好地反映各部分之间的关系

```
car(speed) {  
    chassis();  
    wheel(right_front);  
    wheel(left_front);  
    wheel(right_rear);  
    wheel(left_rear);  
}
```


层次关系

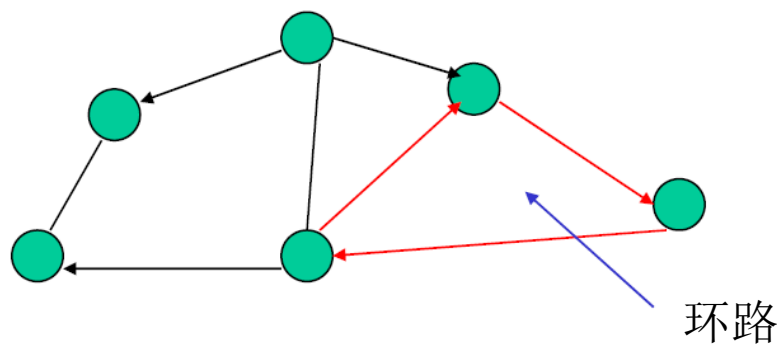


- 层次关系 (hierarchical) 的主要表示方法：
 - 树 (tree)
 - 有向无环图 (DAG)
- 面向对象的编程

图结构



- 图 (graph) 由节点 (node) 和边 (edge, 也称为link) 组成
- 边连接两个节点
 - 有向图 (directed graph)
 - 无向图
- 环路 (cycle) : 构造一个循环的有向路径

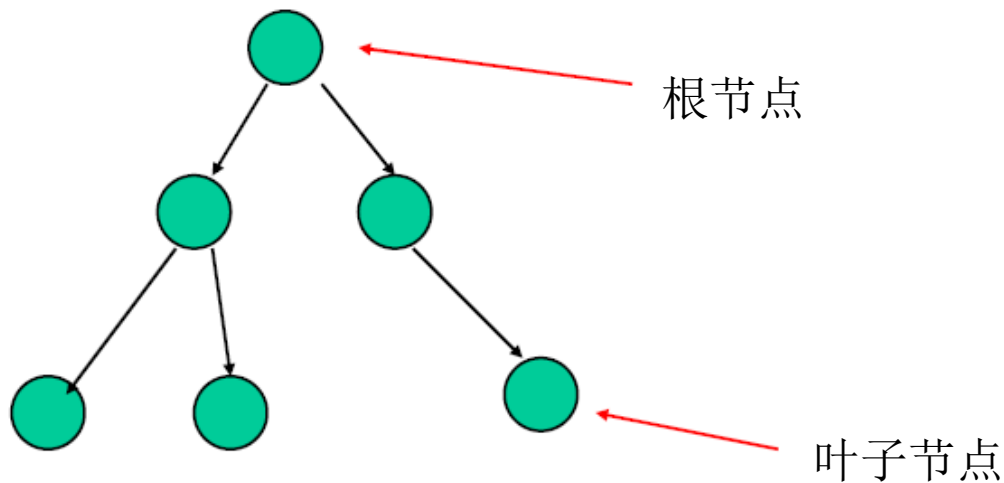


树结构

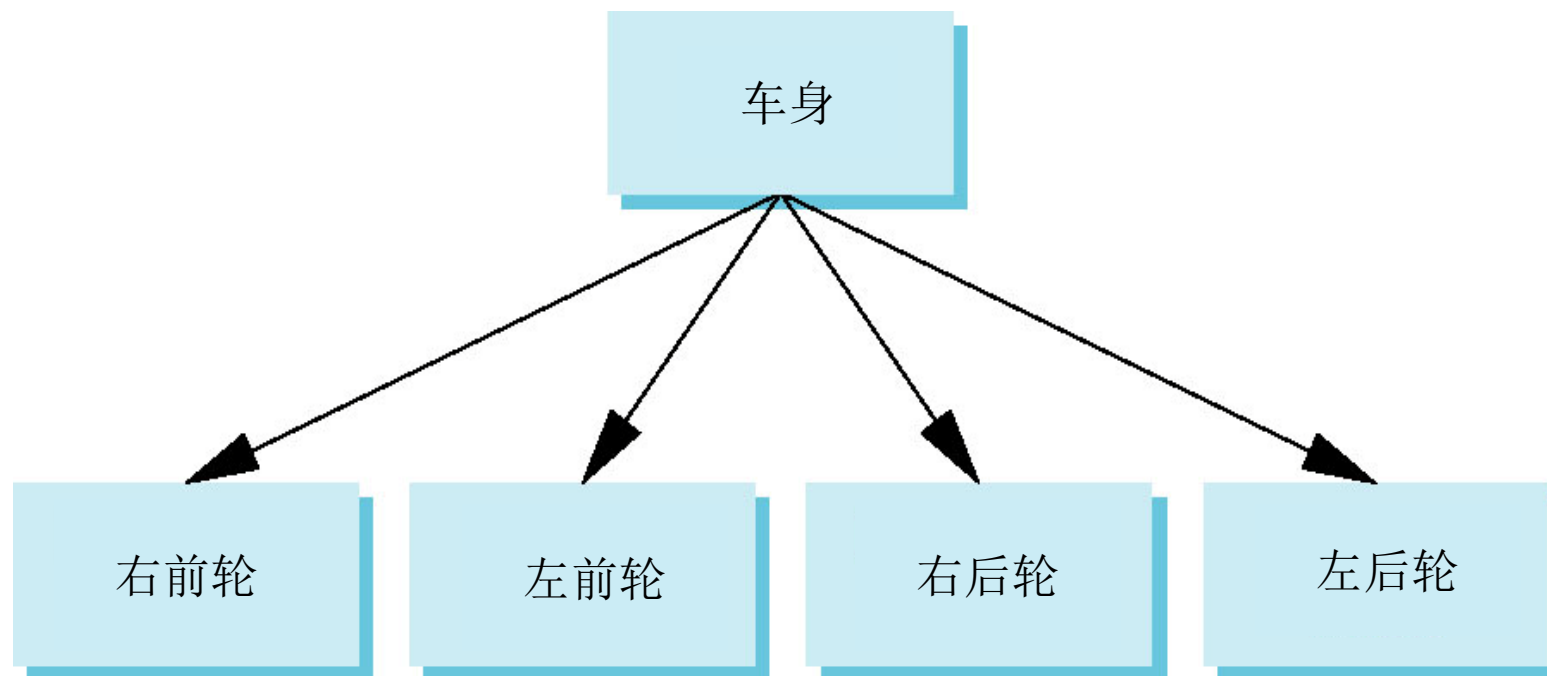


■ 一种特殊的图结构，其中每个节点（除了根节点）都有一个父节点

- 可以有多个子节点
- 叶子：没有子节点的节点



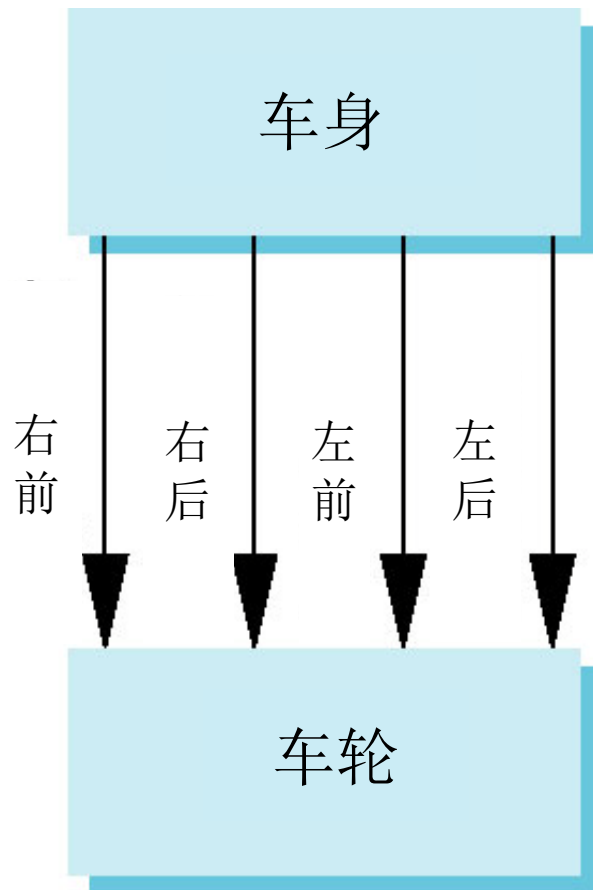
汽车的树结构模型



DAG结构模型



- 如果注意到所有轮子相同的事实，那么就得到有向无环图（directed acyclic graph, DAG）
 - 与树结构的处理没有很大的不同

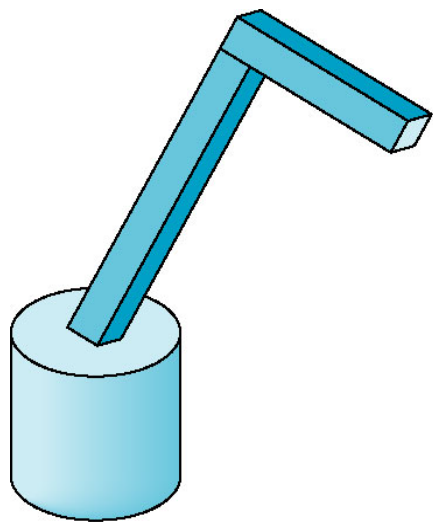


应用树结构建模

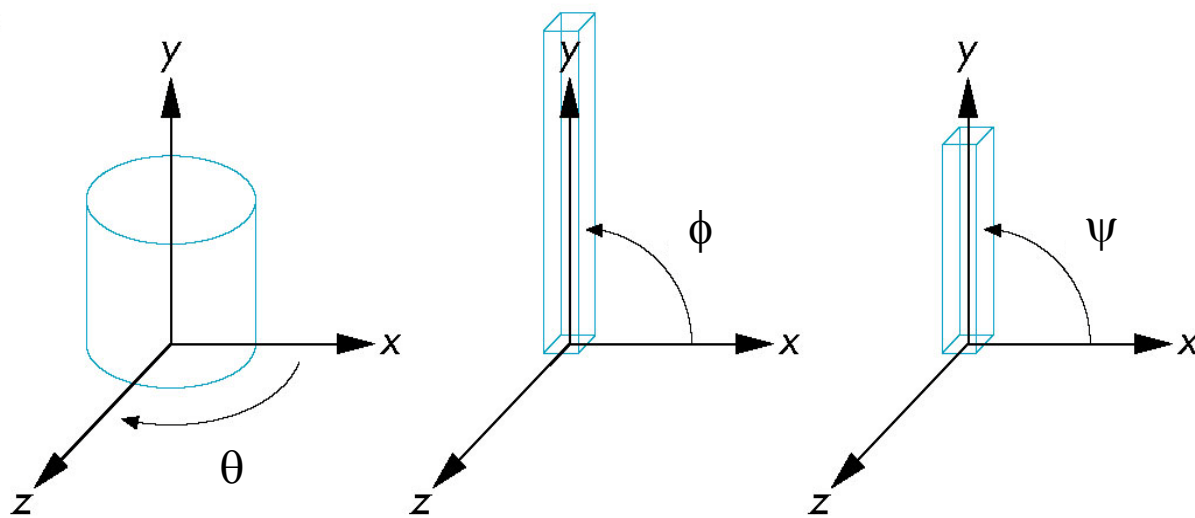


- 需要确定把哪种信息放在节点上，哪种信息放在边上
- 节点
 - 要绘制的内容
 - 指向子节点的指针
- 边
 - 可以包含变换矩阵改变的信息（也可以保存在节点处）

机器人手臂



机器人手臂

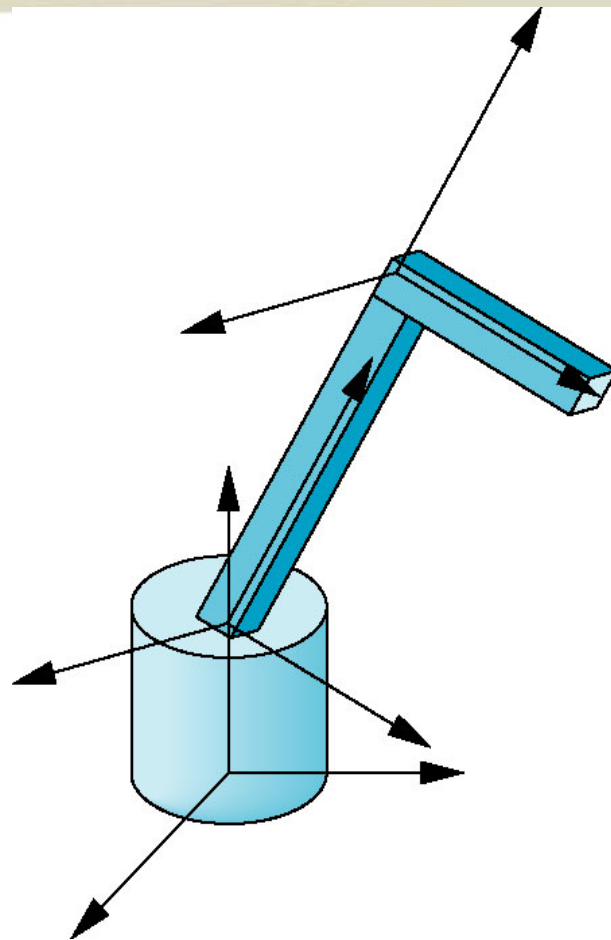


每部分在自己坐标系中的形状

关联的模型



- 机器人手臂就是一个关联的模型 (articulated model) 的例子
- 部分与部分之间在关节处连接在一起
- 可以通过给定关节角指定模型的状态



机器人手臂中的关系



■ 支架部分独立旋转

- 单个角度确定它的位置

■ 下臂与支架部分相连

- 它的位置与支架的旋转相关
- 必须相对于支架平移，并且绕关节点旋转

■ 上臂与下臂相连

- 它的位置与支架和下臂的位置有关
- 相对于下臂部分平移，并且绕与下臂相连的关节处旋转

所需要的矩阵

- 支架的旋转: R_b
 - 把 $M = R_b$ 应用到支架上
- 下臂相对于支架部分部分平移: T_{lu}
- 下臂绕关节旋转: R_{lu}
 - 把 $M = R_b T_{lu} R_{lu}$ 应用到下臂上
- 上臂相对于下臂平移: T_{uu}
- 上臂绕关节旋转: R_{uu}
 - 把 $M = R_b T_{lu} R_{lu} T_{uu} R_{uu}$ 应用到上臂上

机器人手臂的OpenGL代码

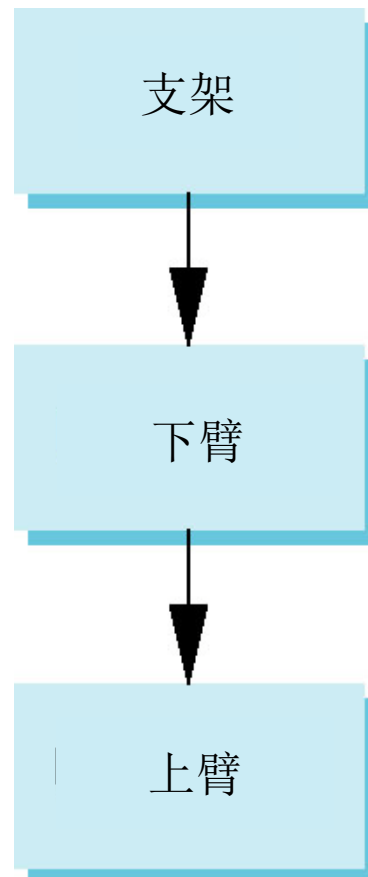


```
robot_arm() {  
    glRotated(theta, 0.0, 1.0, 0.0);  
    base();  
    glTranslated(0.0, h1, 0.0);  
    glRotated(phi, 0.0, 1.0, 0.0);  
    lower_arm();  
    glTranslated(0.0, h2, 0.0);  
    glRoated(psi, 0.0, 1.0, 0.0);  
    upper_arm();  
}
```

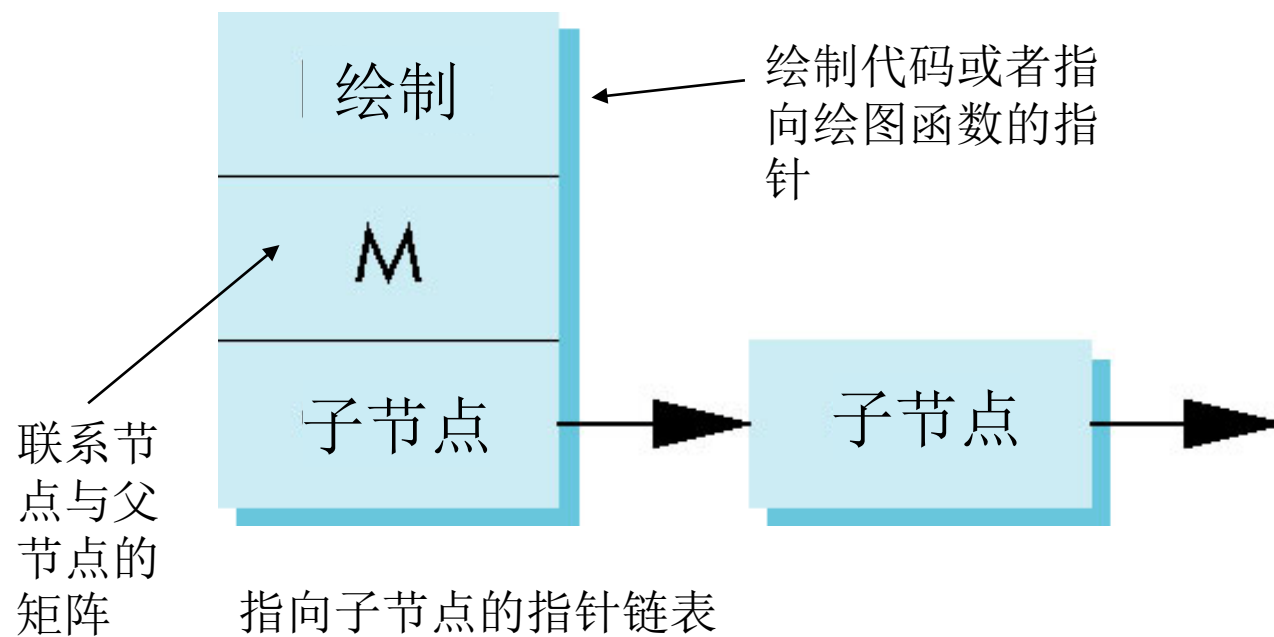
机器人手臂的树结构模型



- 注意上述代码显示了模型中各部分的关系
 - 这样可以很容易改变各部分的样子，而不改变它们相互之间的关系
- 右图为机器人手臂的树结构模型
- 希望为节点建立起一个一般的节点结构



可能的节点结构



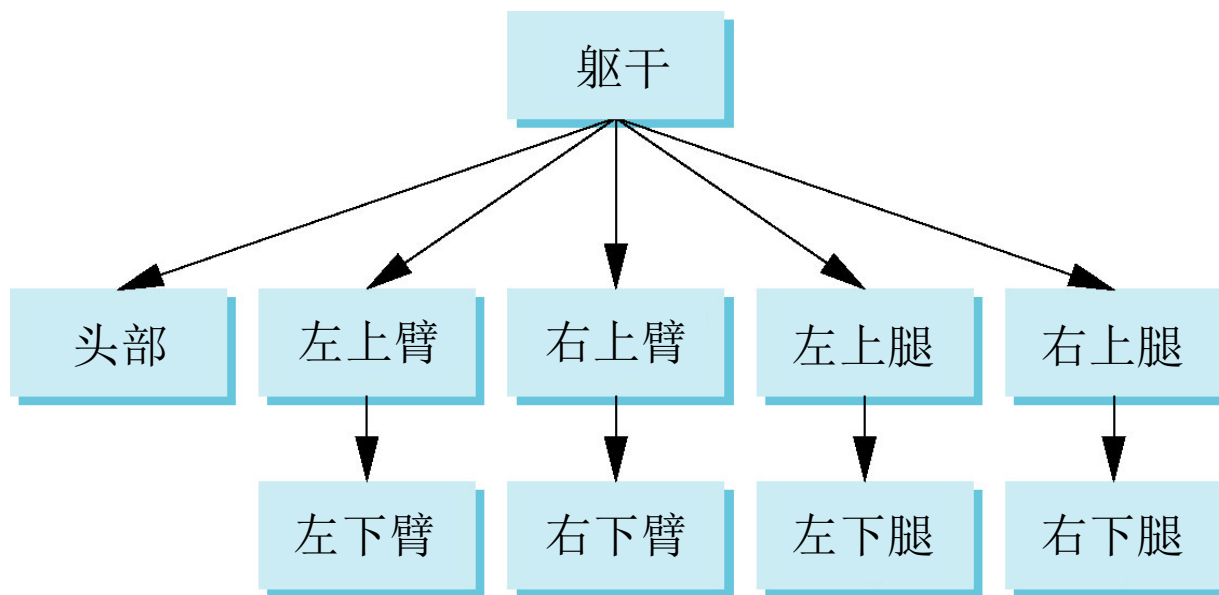
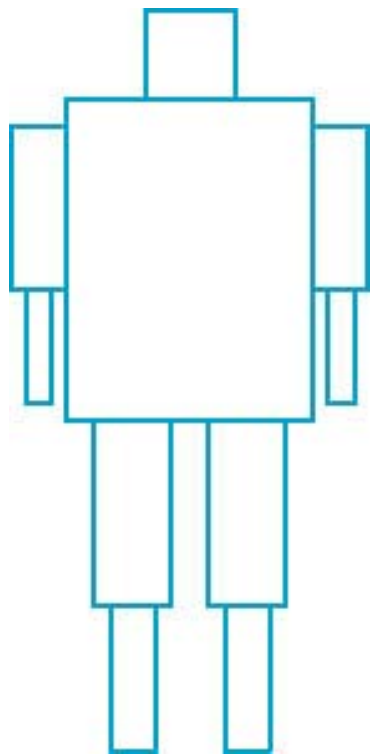
■ 需要处理多个子节点的情形

- 如何表示一个更一般的树结构？
- 如何遍历这样的数据结构？

■ 动画

- 如何动态应用？
- 能否在执行期间创建和删除节点？

人体示意图

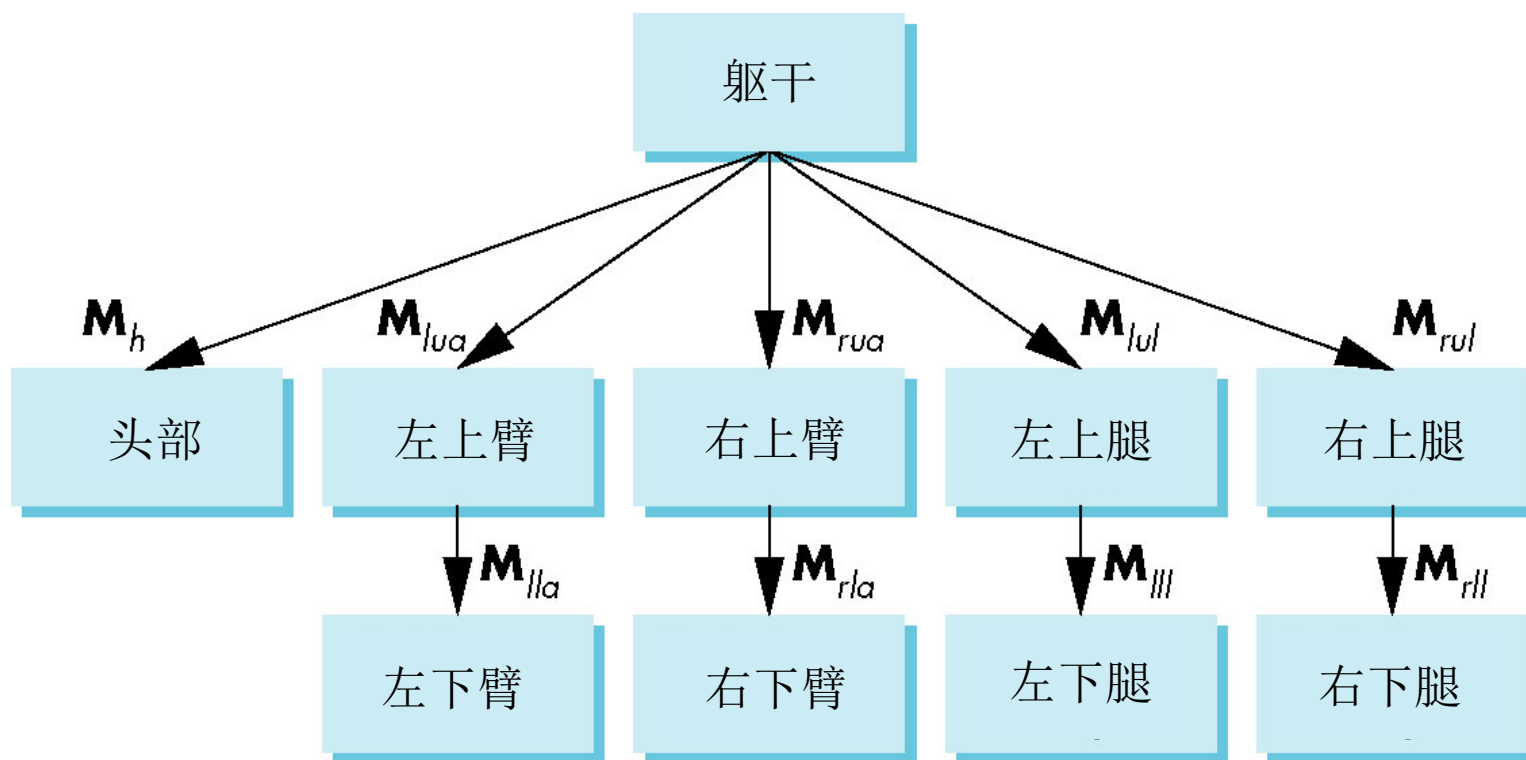


模型的建立



- 可以用二次曲面简单实现前述示意图
 - 椭球面与圆柱面
- 通过函数调用创建每一部分
 - torso();
 - left_upper_arm();
- 矩阵描述节点相对于其父节点的位置
 - M_{III} 相对于左上腿定位左下腿

带有矩阵的树结构



- 示意图的位置是由11个关节角（头部有两个，其它每部分一个）确定的
- 树结构的显示需要对图进行遍历
 - 访问每个节点一次
 - 在每个节点处的绘制函数描述与节点相联系的部分，并且应用恰当的变换矩阵进行定位和定向

变换矩阵










■ 有10个相关的矩阵

- M 定位和定向整个示意图，它作用在根节点躯干上
- M_h 相对于躯干定位头部
- $M_{lua}, M_{rua}, M_{lul}, M_{rul}$ 相对于躯干定位手臂与腿
- $M_{lla}, M_{rla}, M_{lll}, M_{rll}$ 相对于四肢的上半部分定位各自的下半部分

基于堆栈的遍历



- 把模型视图矩阵设为 M ，并绘制躯干
- 设置模型视图矩阵为 MM_h ，并绘制头部
- 对于左上臂，应用矩阵 MM_{lua} ，其它部分类似
- 在实际应用时，没有必要自己重新计算 MM_{lua} 或者应用逆矩阵，可以应用矩阵堆栈存储矩阵 M 以及其它在遍历树结构时遇到的矩阵

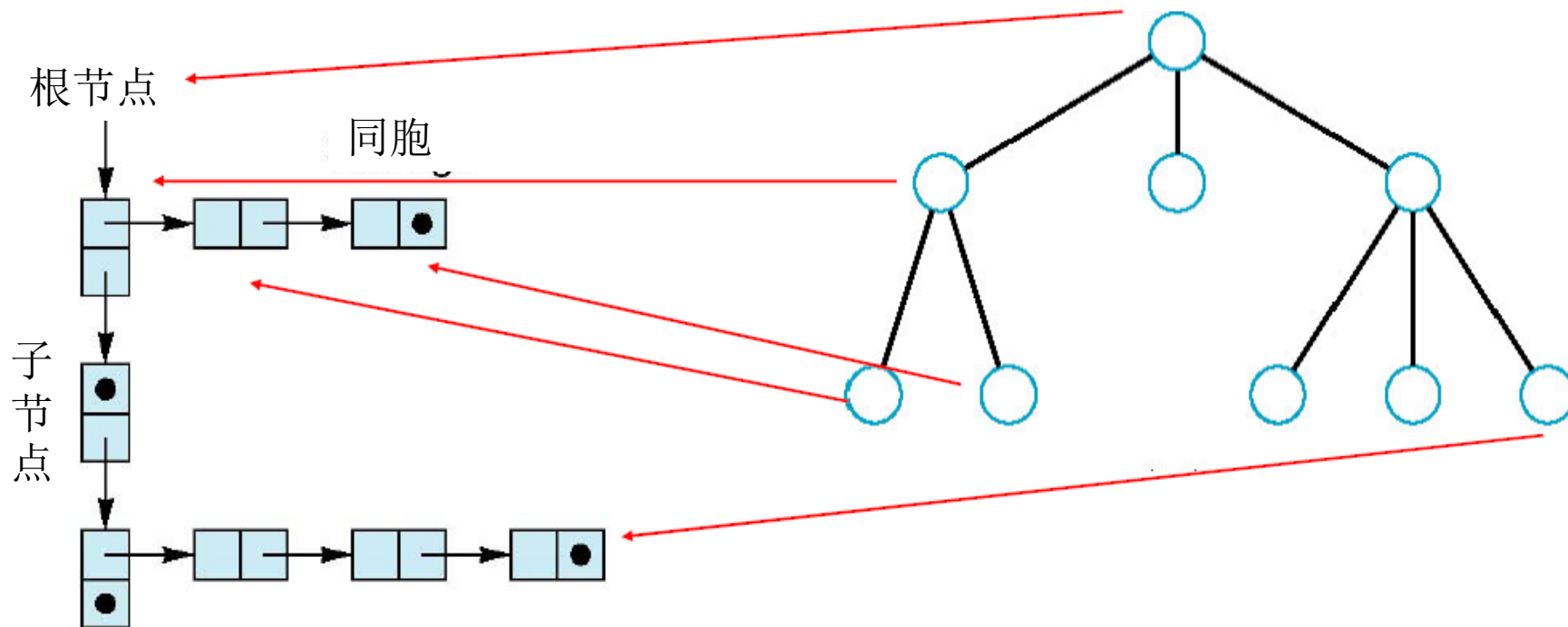
```
figure() {  
    glPushMatrix()  保存当前的模型视图矩阵  
    torso();  
    glRotate3f(...)  为头部绘制更新模型视图矩阵  
    head();  
    glPopMatrix()  恢复原来的模型视图矩阵  
    glPushMatrix();  再次保存模型视图矩阵  
    glTranslate3f(...);  
    glRotate3f(...)  对左上臂更新模型视图矩阵  
    left_upper_arm();  
    glPopMatrix();  再次恢复并保存原来的模型视图矩阵  
    glPushMatrix();  其它代码类似  
    .....  
}
```

- 上述代码描述了特定的树结构，并采用了特定的遍历策略
 - 能否设计出更一般的方法呢？
- 注意在示例代码中没有对状态进行修改，例如没有改变颜色
 - 也可以采用`glPushAttrib`和`glPopAttrib`以避免意料之外的状态改变对后面的代码有影响

一般的树数据结构

- 需要一个数据结构表示树和一个算法遍历树
- 采用左子右亲 (left-child right-sibling) 结构：
 - 应用链表
 - 在数据结构中每个节点有两个指针
 - 左：下一个节点
 - 右：子节点链表

左子右亲树结构



该图正确吗？为什么？

树节点的结构



■ 在每个节点需要存储下列信息

- 指向同胞的指针
- 指向子节点的指针
- 一个函数指针，指向绘制节点表示对象的函数
- 乘在当前模型视图矩阵右边的齐次坐标矩阵
 - 表示从父节点到当前节点的改变
 - 在OpenGL中这个矩阵存储为一维的形式，矩阵按列展开

树节点的C定义



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

定义躯干节点



```
treenode torso_node, head_node, lua_node, ...;
/*应用OpenGL函数形成矩阵 */
glLoadIdentity();
glRotatef(theta[0], 0.0, 1.0, 0.0);
/*把模型视图矩阵移到m中*/
glGetFloatfv(GL_MODELVIEW_MATRIX, torso_node.m);
torso_node.f = torso; //torso()绘制躯干
torso_node.sibling = NULL;
torso_node.child = &head_node;
```

- 示意图的位置是由存储在`theta[11]`中的11个关节角确定的
- 可以通过改变这些角度并且重新显示实现动画效果
- 利用`glRotate`和`glTranslate`形成所需要的矩阵
 - 由于矩阵是相对于模型视图矩阵构成的，因此需要首先把原来的模型视图矩阵送到矩阵堆栈中

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

- 在模型视图矩阵上乘以节点矩阵之前，必须保存模型视图矩阵
 - 对子节点进行的矩阵修改并不应用到同胞节点上，因为它们有自己的矩阵
- 遍历程序适用于任意的左子右亲树结构
 - 特定的树可以通过对树节点进行特别地处理实现
- 遍历的顺序是需要考虑的，因为函数有可能对状态进行了修改

动态树结构



- 如果应用指针，那么树结构可以是动态的

```
typedef treeNode *tree_ptr;  
tree_ptr torso_ptr;  
torso_ptr = malloc(sizeof(treeNode));
```

- 节点的定义以及节点的遍历策略本质上是一样的，但在执行过程中需要增加或删除节点

Thanks for your attention!

