Hierarchical Modeling

- Instance transformations
- Transformation hierarchies
- Tree data structure

What is a Model?

- A representation of features of an abstract of concrete identity
- Allows people to visualize the entity and understand the behavior of the entity
- A convenient means to experiment and predict
- Different Types of Models:
  - **Geometric:** collections of components with well-defined geometry and often interconnections between components (e.g. architectural structures)
  - **Quantitative:** equations describing some type of system (e.g. mathematical, economic, or chemical)
  - **Organizational:** representations of hierarchies and taxonomies (e.g. org chart, library classification scheme)

Geometric Models

Geometric models are models that represent:

- spatial layout and shape of components geometry and other attributes such as color
- connectivity of components topology
- application specific data and properties associated with components such as physical characteristics or descriptive text

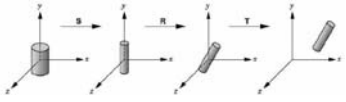
⇒ lend themselves naturally to graphical representation

Instance Transformation

- Often we need several instances of an object
  - Wheels of a car
  - Arms or legs of a figure
  - Chess pieces
- Instances can be shared across space or time
- Encapsulate basic object in a function
- Apply transformations to different instances
- Typical order: *scaling* ⇒ *rotation* ⇒ *translation*

Instance Transformation: Sample

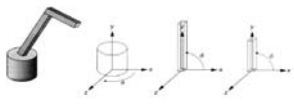
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
glScalef(...);
gluCylinder(...);
```



How to connect these primitives?

Example: A Robot Arm

- Consider this robot arm with 3 degrees of freedom:
  - Base rotates about its vertical axis by θ
  - Lower arm rotates in its *xy*-plane by Φ
  - Upper arm rotates in its *xy*-plane by Ψ



What matrix should we use to transform each part?

Robot Arm Implementation

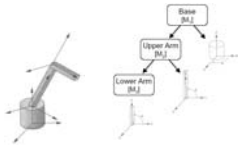
```
main()
{
  ...
  glMatrixMode( GL_MODELVIEW );
  glLoadIdentity();
  robot_arm(a, b, c);
  ...
}

robot_arm(theta, phi, psi)
{
  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();
}
```

Robot Arm: Transformation Hierarchies

Model object as hierarchy of components

- each object node has a local coordinate system
- each component defined relative to parent
- each level stores matrix representing transformation from parent's coordinate system
- each one can move and they move relative to parent Body



Robot Arm: Transformation Hierarchies

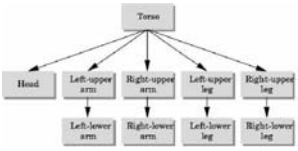
Traverse hierarchy during draw

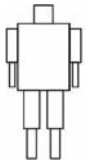
- multiply into current matrix on way down
- draw current node
- traverse current node (recursively)
- remove from current matrix on way up

⇒ Constantly changing local coordinate system

More complex objects

- Tree rather than linear structure
- Interleave along each branch
- Use push and pop to save state



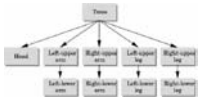


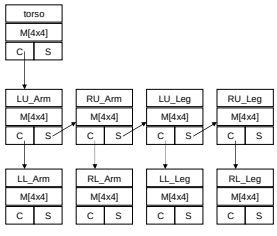
More complex objects (2)

```
human_figure ()
{
  torso();
  glPushMatrix();
    glTranslate(…);
    glRotate(…);
    head();
  glPopMatrix();
  // draw left leg
  glPushMatrix();
    glTranslate(…);
    glRotate(…);
    left_upper_leg();
    glPushMatrix();
      glTranslate(…);
      glRotate(…);
      left_lower_leg();
    glPopMatrix();
  glPopMatrix();
  // right leg, left arm, right arm
  ...
}
```

Tree Data Structures

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





Tree Data Structure: Initializing

- Initializing transformation matrix for node

```
treenode torso, head, ...;
/* in init function */
glLoadIdentity();
glRotatef(...);
glGetFloatv(GL_MODELVIEW_MATRIX, torso.m);
```

- Initializing pointers

```
torso.f = drawTorso;
torso.sibling = NULL;
torso.child = &head;
```


Tree Data Structure: Traversing

```
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);
}
```