Modeling Objects

- A prototype has a default size, position, and orientation
- You need to perform modeling transformations to position it within the scene

```
myCube() - 
Create a unit cube with its origin at (0,0,0)
```

To create a 2 x 0.1 x 2 table top - need to call `glScalef(2, 0.1, 2)`

Instance Transformation

- Start with a prototype object (a symbol)
- Each appearance of the object in the model is an instance
- Must scale, orient, position

Symbol-Instance Table

Can store a model by assigning a number to each symbol and storing the parameters for the instance transformation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Scale</th>
<th>Rotate</th>
<th>Translate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Relationships in Car Model

- Symbol-instance table does not show relationships between parts of model
- Consider model of car
  - Chassis + 4 identical wheels
  - Two symbols

- Rate of forward motion determined by rotational speed of wheels

Structure Through Function

Calls

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

Fails to show relationships well
Look at problem using a graph

Graphs

- Set of nodes and edges (links)
- Edge connects a pair of nodes
  - Directed or undirected
- Cycle: directed path that is a loop

Tree

- Graph in which each node (except the root) has exactly one parent node
  - May have multiple children
- Leaf or terminal node: no children
Tree Model of Car

DAG Model

If we use the fact that all the wheels are identical, we get a directed acyclic graph. Not much different than dealing with a tree.

Modeling with Trees

- Must decide what information to place in nodes and what to put in edges
  - Nodes
    - What to draw
    - Pointers to children
  - Edges
    - May have information on incremental changes to transformation matrices (can also store in nodes)

Robot Arm

Robot arm parts in their own coordinate systems.
Articulated Models

- Robot arm is an example of an articulated model
  - Parts connected at joints
  - Can specify state of model by giving all joint angles

Relationships in Robot Arm

- Base rotates independently
  - Single angle determines position
- Lower arm attached to base
  - Its position depends on rotation of base
  - Must also translate relative to base and rotate about connecting joint

Relationships in Robot Arm

- Upper arm attached to lower arm
  - Its position depends on both base and lower arm
  - Must translate relative to lower arm and rotate about joint connecting to lower arm

Required Matrices

- Rotation of base: $\mathbf{R}_b$
  - Apply $\mathbf{M} = \mathbf{R}_b$ to base
- Translate lower arm relative to base: $\mathbf{T}_{la}$
- Rotate lower arm around joint: $\mathbf{R}_{la}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{la} \mathbf{R}_{la}$ to lower arm
- Translate upper arm relative to upper arm: $\mathbf{T}_{ua}$
- Rotate upper arm around joint: $\mathbf{R}_{ua}$
  - Apply $\mathbf{M} = \mathbf{R}_b \mathbf{T}_{la} \mathbf{R}_{la} \mathbf{T}_{ua} \mathbf{R}_{ua}$ to upper arm
OpenGL Code for Robot

```c
robot_arm()
{
  glRotate(theta, 0.0, 1.0, 0.0);
  base();
  glTranslate(0.0, h1, 0.0);
  glRotate(phi, 0.0, 1.0, 0.0);
  lower_arm();
  glTranslate(0.0, h2, 0.0);
  glRotate(psi, 0.0, 1.0, 0.0);
  upper_arm();
}
```

Tree Model of Robot

- Note code shows relationships between parts of model
  - Can change “look” of parts easily without altering relationships
- Simple example of tree model
- Want a general node structure for nodes

Scene Graphs

- Encoding this information in the code is not very productive. Want it to be flexible, data-driven and extensible.
- Scene-graphs provide this functionality.
  - OpenInventor (http://www.coin3d.org/)
  - Open Scene Graph (http://www.openscenegraph.com/)
  - Many others

Hierarchical Modeling

- Triangles, parametric curves and surfaces are the building blocks from which more complex real-world objects are modeled.
- Hierarchical modeling creates complex real-world objects by combining simple primitive shapes into more complex aggregate objects.
Articulated Models

Multiple Components
- Different Materials

Scene Layout - Worlds
Hierarchical Grouping of Objects

Logical organization of scene

Simple Example with Groups

Group {
    numObjects 3
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        numObjects 2
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
        }
        Group {
            Box { <BOX PARAMS> }
            Sphere { <SPHERE PARAMS> }
            Sphere { <SPHERE PARAMS> }
        }
    }
    Plane { <PLANE PARAMS> }
}

Adding Materials

Group {
    numObjects 3
    Material { <BLUE> }
    Group {
        numObjects 3
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
        Box { <BOX PARAMS> }
    }
    Group {
        numObjects 2
        Material { <BROWN> }
        Group {
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
        }
        Group {
            Material { <GREEN> }
            Box { <BOX PARAMS> }
            Material { <RED> }
            Sphere { <SPHERE PARAMS> }
            Material { <ORANGE> }
            Sphere { <SPHERE PARAMS> }
            Material { <BLACK> }
            Plane { <PLANE PARAMS> }
        }
    }
}

Adding Transformations
Hierarchical Transformation of Objects

- Transforms position logical groupings of objects within the scene

Simple Example with Transforms

```plaintext
Group {
    numObjects 3
    Transform {
        ZRotate { 45 }
        Group {
            numObjects 3
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> }
            Box { <BOX PARAMS> } } }
    Transform {
        Translate { -2 0 0 }
        Group {
            numObjects 2
            Group {
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> }
                Box { <BOX PARAMS> } }
            Group {
                Box { <BOX PARAMS> }
                Sphere { <SPHERE PARAMS> }
                Sphere { <SPHERE PARAMS> } } } }
} Plane { <PLANE PARAMS> }
```

Separating types of transformation

- Note that we have treated translations, rotations, etc. as separate
- But they are all represented by 4x4 matrices and there is no technical reason not to combine them into the resulting matrix
- It’s just simpler for the human programmer, and corresponds to the handle of 3D modeling/animation packages

Hierarchical modeling in OpenGL

- Commands to change current transformation
  - glTranslate, glScale, etc.
- Affects the `state`, i.e. all following commands will undergo this transformation
- Utilities to maintain a matrix stack (to revert to previous state)
- Difference between model and view matrix
Model vs. Projection matrix

- It is almost the same to rotate the camera or the objects
- Main difference:
  - Lighting

  This is why OpenGL has two transforms: model and projection

  ```glMatrixMode(GL_MODELVIEW);```
  - Tells OpenGL that next matrix commands deal with the objects.
  - Typically used for modeling & animation

  ```glMatrixMode(GL_PROJECTION);```
  - Tells OpenGL we deal with the camera space (typically used to change viewpoint & focal length

Managing the state

- To reset everything:
  ```glLoadIdentity();```

- OpenGL stores a stack of matrices
  - You don't need to remember, OpenGL remembers
  - ```glPushMatrix()```
  - ```glPopMatrix()```

Managing the state

- Push matrix when you start rendering a group
- Pop once you are done

Scene Graph

- Convenient Data structure for scene representation
  - Transformations
  - Materials, color
  - Multiple instances
  - Basic idea: Hierarchical Tree
  - Useful for manipulation/animation
    - Especially for articulated figures
  - Useful for rendering too
    - Multi-pass rendering, occlusion culling
Scene Graphs

- Basic idea: Tree
- Comprised of several node types:
  - Shape: 3D geometric objects
  - Transform: Affect current transformation
  - Property: Appearance, texture, etc.
  - Group: Collection of subgraphs

Traversal

- Depth first
  - Top to bottom, left to right

Traversal State

- The State is updated during traversal
  - Transformations, properties
  - Influence of nodes can be complex
  - E.g. bottom to top

Other Scene Nodes

- Switch or Selector Nodes
  - Level of detail
  - Different rendering styles
  - Damaged v. undamaged states
- Sequence Nodes
  - Animated sequence
    - Objects
    - Textures
    - Transformations
Object Following or Tethering

- Can attach an object to the node of another object (or group them in a parent node).
- Provide an offset to have the other object tag along with the object under control.

Camera Tethering

- Many times, we want the camera to follow an entity.
- Need to get the coordinate frame for the entity.
  - Traverse the scene graph to the entity.
- Or, need to get the current camera system
  - Attach camera and determine transformations to the root of the scene graph.