Instance Transformation

- Start with unique object (a symbol)
- Each appearance of object in model is an instance
  - Then scale, orient, position (instance transformation)
Symbol-Instance Table

Approach 1: store instances + instance transformations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Scale</th>
<th>Rotate</th>
<th>Translate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$s_x, s_y, s_z$</td>
<td>$\theta_x, \theta_y, \theta_z$</td>
<td>$d_x, d_y, d_z$</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>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram: 3D transformations of a cylinder.
Problems with Symbol-Instance Table

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

- Relationships:
  - Wheels connected to chassis
  - Chassis motion determined by rotational speed of wheels
car(speed)
{
    chassis()
    wheel(right_front);
    wheel(left_front);
    wheel(right_rear);
    wheel(left_rear);
}

- Fails to show relationships between parts
- Explore graph representation
Graphs

- **Set of** *nodes* + *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 root) has exactly one parent node
  - A parent may have multiple children
  - Leaf node: no children
Tree Model of Car
Hierarchical Transforms

- **Robot arm**: Many small **connected** parts
- Attributes of parts (position, orientation, etc) depend on each other

![A Robot Hammer!](image)
Hierarchical Transforms

- Object dependency description using tree structure

Object position and orientation can be affected by its parent, grand-parent, grand-grand-parent ... nodes

Hierarchical representation is known as a **Scene Graph**
Transformations

- Two ways to specify transformations:
  - (1) **Absolute transformation**: each part transformed independently (relative to origin)

  ```
  Translate the base by (5,0,0);
  Translate the lower arm by (5,0,0);
  Translate the upper arm by (5,0,0);
  ...
  ```
Relative Transformation

A better (and easier) way:

(2) **Relative transformation**: Specify transformation for each object relative to its parent

Step 1: Translate base and its child nodes by \((5,0,0)\);
Step 2: Rotate the lower arm and all its descendants by -90 degrees, relative to the base’s local y axis.
Relative Transformation

- Relative transformation using scene graph

- Base
  - Translate (5,0,0)
  - Rotating (-90) about y axis
  - Apply all the way down

- Lower arm
  - Apply all the way down

- Upper arm

- Hammer
  - Apply all the way down
Hierarchical Transforms Using OpenGL

- Translate base and all its descendants by (5,0,0)
- Rotate lower arm and its descendants by -90 degree about local y

```c
ctm = LoadIdentity();
... // setup your camera
ctm = ctm * Translatef(5,0,0);
Draw_base();
ctm = ctm * Rotatef(-90, 0, 1, 0);
Draw_lower_arm();
Draw_upper_arm();
Draw_hammer();
```
Hierarchical Modeling

● For large objects with many parts, need to transform groups of objects
● Need better tools
● Need matrix stack
Hierarchical Modeling

- Previous CTM had 1 level
- **Hierarchical modeling:** extend CTM to stack with multiple levels using linked list
- Manipulate stack levels using 2 operations
  - pushMatrix
  - popMatrix
PushMatrix

- **PushMatrix()**: Save current modelview matrix (CTM) in stack
- Positions 1 & 2 in linked list are same after PushMatrix

**Before PushMatrix**

Current top of CTM stack

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

**After PushMatrix**

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Saved copy of matrix at CTM top

Current top of CTM stack

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
PushMatrix

- Subsequent Rotate, Scale, Translate change only top matrix
- \[ \text{E.g. } \text{ctm} = \text{ctm} \times \text{Translate } (3,8,6) \]

After PushMatrix

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Translate(3,8,6) applied only to current top of CTM stack

Matrix in second position saved.
Unchanged by Translate(3,8,6)
PopMatrix

- **PopMatrix( ):** Delete position 1 matrix, position 2 matrix becomes top

```
Before PopMatrix

Current top Of CTM stack

\[
\begin{pmatrix}
1 & 5 & 4 & 0 \\
0 & 2 & 2 & 0 \\
0 & 6 & 3 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]

After PopMatrix

Current top Of CTM stack

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1 \\
\end{pmatrix}
\]
```
PopMatrix and PushMatrix Illustration

- **Note:** Diagram uses old `glTranslate`, `glScale`, etc commands. **Deprecated!!**

- We want same behavior though

Apply matrix at top of CTM to vertices of object created

Figure 4.19: Transitions of the modelview matrix stack.

**Ref:** Computer Graphics Through OpenGL by Guha
Humanoid Figure

- Upper arm
- Lower arm
- Upper leg
- Lower leg

Diagram with labeled parts and a hierarchical structure:
- Torso
  - Head
  - Left-upper arm
  - Right-upper arm
  - Left-upper leg
  - Right-upper leg
  - Left-lower arm
  - Right-lower arm
  - Left-lower leg
  - Right-lower leg
Building the Model

- Draw each part as a function
  - torso()
  - left_upper_arm(), etc

- **Transform Matrices**: transform of node wrt its parent
  - E.g. $M_{lla}$ positions left lower arm with respect to left upper arm

- Stack based traversal (push, pop)
figure() {
    PushMatrix();
    torso();
}

- save present model-view matrix
- draw torso
Draw Humanoid using Stack

```cpp
figure() {
    PushMatrix();
    torso();
    Rotate (...);
    head();
}
```

$(M_h)$ Transformation of head Relative to torso
draw head
Draw Humanoid using Stack

```
figure() {
    PushMatrix()
    torso();
    Rotate (...);
    head();
    PopMatrix();
    PushMatrix();
    Translate (...);
    Rotate (...);
    left_upper_arm();
    .......
    // rest of code()
```

- **PushMatrix()**
- **torso();**
- **Rotate (...);**
- **head();**
- **PopMatrix();**
- **PushMatrix();**
- **Translate (...);**
- **Rotate (...);**
- **left_upper_arm();**

Go back to torso matrix, and save it again

\((M_{\text{lua}})\) Transformation(s) of left upper arm relative to torso

Draw Humanoid

- Head
- Left-upper arm
- Torso

\(M_h\)
Complete Humanoid Tree with Matrices

Scene graph of Humanoid Robot
VRML

- Scene graph introduced by SGI Open Inventor
- Used in many graphics applications (Maya, etc)
- **Virtual Reality Markup Language**
  - Scene graph representation of virtual worlds on Web
  - Scene parts can be distributed across multiple web servers
  - Implemented using OpenGL
References