Rotation About Arbitrary Point other than the Origin

- Default rotation matrix is about origin
- How to rotate about any arbitrary point $p_f$ (Not origin)?
  - Move fixed point to origin $T(-p_f)$
  - Rotate $R(\theta)$
  - Move fixed point back $T(p_f)$

So, $M = T(p_f) \cdot R(\theta) \cdot T(-p_f)$
Scale about Arbitrary Center

• Similary, default scaling is about origin

• To scale about arbitrary point $P = (Px, Py, Pz)$ by $(Sx, Sy, Sz)$
  1. **Translate** object by $T(-Px, -Py, -Pz)$ so $P$ coincides with origin
  2. **Scale** object by $(Sx, Sy, Sz)$
  3. **Translate** object back: $T(Px, Py, Py)$

• In matrix form: $T(Px,Py,Pz) \ (Sx, Sy, Sz) \ T(-Px,-Py,-Pz) \ * \ P$

\[
\begin{pmatrix}
x' \\
y' \\
z' \\
1
\end{pmatrix} = \begin{pmatrix}
1 & 0 & 0 & Px \\
0 & 1 & 0 & Py \\
0 & 0 & 1 & Pz \\
0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
S_x & 0 & 0 & 0 \\
0 & S_y & 0 & 0 \\
0 & 0 & S_z & 0 \\
0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
1 & 0 & 0 & -Px \\
0 & 1 & 0 & -Py \\
0 & 0 & 1 & -Pz \\
0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]
Example

- Rotation about \( z \) axis by 30 degrees about a fixed point \((1.0, 2.0, 3.0)\)

```cpp
mat 4 m = Identity();
m = Translate(1.0, 2.0, 3.0) * 
    Rotate(30.0, 0.0, 0.0, 1.0) * 
    Translate(-1.0, -2.0, -3.0);
```

- Remember last matrix specified in program (i.e. translate matrix in example) is first applied
Computer Graphics (CS 4731)
Lecture 11: Hierarchical 3D Models

Prof Emmanuel Agu

Computer Science Dept.
Worcester Polytechnic Institute (WPI)
**Instance Transformation**

- Start with unique object (a *symbol*)
- Each appearance of object in model is an *instance*
  - Must scale, orient, position
  - Defines instance transformation

![Diagram showing transformation process from symbol to instance](image)
Symbol-Instance Table

Can store *intances + 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>
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
Structure Program Using Function Calls?

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

- Fails to show relationships between parts
- Look into 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 (position, orientation, etc) depend on each other

A Robot Hammer!
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);
...

![Diagram of transformations with axes and sample translations]
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 descendants by (5,0,0);
Relative Transformation

Step 2: Rotate the lower arm and all its descendants relative to the base’s local y axis by -90 degree
Relative Transformation

- Relative transformation using scene graph

Base
   ↓
Lower arm
   ↓
Upper arm
   ↓
Hammer

Translate (5,0,0)

Rotate (-90) about its local y

Apply all the way down

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
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

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

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

Before PushMatrix

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

After PushMatrix

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

Current top Of CTM stack

Copy of matrix at top of CTM
PushMatrix

- Further Rotate, Scale, Translate affect only top matrix
- E.g. $\text{ctm} = \text{ctm} \times \text{Translate}(3,8,6)$

**After PushMatrix**

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

- Translate(3,8,6) applied only to current top Of CTM stack
- Matrix in second position saved. Unaffected by Translate(3,8,6)
- **PopMatrix( ):** Delete position 1 matrix, position 2 matrix becomes top

**Before PopMatrix**

Current top Of CTM stack

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

Current top Of CTM stack

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

Delete this matrix

**After PopMatrix**
Ref: Computer Graphics Through OpenGL by Guha

- **Note:** Diagram uses old `glTranslate`, `glScale`, etc commands
- We want same behavior though

Apply matrix at top of CTM to vertices of object created

---

![Diagram](image)

**Figure 4.19:** Transitions of the modelview matrix stack.
Humanoid Figure

- Torso
- Upper arm
- Lower arm
- Upper leg
- Lower leg
Building the Model

- Draw each part as a function
  - torso()
  - left_upper_arm(), etc
- Transform Matrices: transform of node wrt its parent
  - $M_{lla}$ positions left lower arm with respect to left upper arm
- Stack based traversal (push, pop)
Draw Humanoid using Stack

```cpp
figure() {
    PushMatrix();  // save present model-view matrix
    torso();      // draw torso
}
```
Draw Humanoid using Stack

\begin{verbatim}
figure() {
    PushMatrix()
    torso();
    Rotate (...);
    head();
}
\end{verbatim}

$M_h$ Transformation of head Relative to torso

draw head
Draw Humanoid using Stack

PushMatrix();
torso();
Rotate (...);
head();
PushMatrix();
PopMatrix();
PushMatrix();
Translate(...);
Rotate(...);
left_upper_arm();
.......;
// rest of code()
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)
- Want scene graph for World Wide Web
- Need links scene parts in distributed data bases
- **Virtual Reality Markup Language**
  - Based on Inventor data base
  - Implemented with OpenGL
VRML World Example
References