Computer Graphics (CS 543)
Lecture 5 (Part 3): 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
Symbol-Instance Table

Can 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>
</tbody>
</table>

![Diagram showing symbol transformations](image)
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?

```c
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

```
  ▼
 ▼  ▼
 ▼  ▼
```

root node

```
  ▼
 ▼
  ▼
```

leaf node
Tree Model of Car
Hierarchical Transforms

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

\[
\begin{align*}
&\text{X} \\
&\text{Y} \\
&\text{Z}
\end{align*}
\]
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

---

**Diagram:**

- **Base**
  - **Translate (5,0,0)**
  - **Rotate (-90) about its local y**

- **Lower arm**
  - **Apply all the way down**

- **Upper arm**
  - **Apply all the way down**

- **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

- Previous CTM had 1 level
- **Hierarchical modeling**: extend CTM to stack with multiple levels using linked list

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 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
**PushMatrix**

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

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.

Unaffected by Translate(3,8,6)
PopMatrix

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

**Before PopMatrix**

Current top Of CTM stack

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

Delete this matrix

**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
- We want same behavior though

```
Code                  | Modelview Matrix Stack
----------------------|------------------------
`glLoadIdentity();`   | \[ \begin{bmatrix} 1 \\ \end{bmatrix} \]
`glTranslatef(0.0, 0.0, -15.0);` | \[ \begin{bmatrix} I * M_1 = M_1 \\ \end{bmatrix} \]
`glPushMatrix();`     | \[ \begin{bmatrix} M_1 \\ M_1 \end{bmatrix} \]
//Copy of \( M_1 \) placed on top.
`glScalef(1.0, 2.0, 1.0);` | \[ \begin{bmatrix} M_1 * M_2 \\ M_1 \end{bmatrix} \]
`glutWireCube(5.0);`  | \[ \begin{bmatrix} M_1 * M_2 \\ M_1 \end{bmatrix} \]
//No change.
`glPushMatrix();`     | \[ \begin{bmatrix} M_1 \\ \end{bmatrix} \]
//Back to before the push statement!
`glTranslatef(0.0, 7.0, 0.0);` | \[ \begin{bmatrix} M_1 * M_3 \\ \end{bmatrix} \]
`glutWireSphere(2.0, 10, 8);` | \[ \begin{bmatrix} M_1 * M_3 \\ \end{bmatrix} \]
//No change.
```

Processing in code order

Apply matrix at top of CTM to vertices of object created

Ref: Computer Graphics Through OpenGL by Guha

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

Torso

Upper arm  Lower arm

Upper leg  Lower leg

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
  - $M_{lla}$ positions left lower arm with respect to left upper arm
- Stack based traversal (push, pop)
Draw Humanoid using Stack

```cpp
figure() {
    PushMatrix();
    torso();
}
```

- `figure()` saves the present model-view matrix.
- `PushMatrix()` pushes the current model-view matrix onto the stack.
- `torso()` draws the torso.

Diagram:
- Torso block with label "Torso"
Draw Humanoid using Stack

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()
}
```

(Mlua) Transformation(s) of left upper arm relative to torso

Go back to torso matrix, and save it again

(M_h) Torso

(M_lua) Head

(M_lua) Left-upper arm
Complete Humanoid Tree with Matrices

```
Complete Humanoid Tree with Matrices

Torso

M_h

M_lla

M_rll

M_lulu

M_rul

M_rua

M_lla

M_lla

M_rll

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
```
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