Computer Graphics (CS 543)
Lecture 7 (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 a model by assigning number to each symbol and storing parameters for 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>$s_x$, $s_y$, $s_z$</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>
Relationships in Car Model

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

- Relationship: Rate of forward motion determined by rotational speed of wheels
Structure using Function Calls

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

Chassis

- Right-front wheel
- Left-front wheel
- Right-rear wheel
- Left-rear wheel
Hierarchical Transforms

- Robot arm: Many small 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 of the object is transformed independently relative to the 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 the 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

- Represent relative transformation using scene graph
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

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
- Further Rotate, Scale, Translate affect only top matrix

**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**

Current top of CTM stack

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

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 3 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
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
- We want same behavior though

![Diagram](image)

Figure 4.19: Transitions of the modelview matrix stack.

**Ref:** Computer Graphics Through OpenGL by Guha
Humanoid Figure
Building the Model

- Can build model using simple shapes
- Access parts through functions
  - torso()
  - left_upper_arm()
- Matrices describe position of node with respect to its parent
  - $\mathbf{M}_{\text{lla}}$ positions left lower leg with respect to left upper arm
Tree with Matrices
Set model-view matrix to $M$ and draw torso
Tree with Matrices

Set model-view matrix to $\mathbf{MM}_h$ and draw head

- Torso
  - Head
    - Left-upper arm
    - Left-lower arm
  - Left-upper arm
  - Right-upper arm
    - Right-lower arm
  - Left-upper leg
    - Left-lower leg
  - Right-upper leg
    - Right-lower leg
Tree with Matrices

Set model-view matrix to $\text{MM}_{\text{lua}}$ and draw left-upper arm.
Stack-based Traversal

- We can use stack, Push, Pop for this
- Rather than recomputing $MM_{lua}$ from scratch or using an inverse matrix, we can use the matrix stack to store $M$ and other matrices as we traverse the tree
Traversals Code

```cpp
figure() {
  save present model-view matrix
  PushMatrix();
  torso();
  Rotate (...);
  head();
  PopMatrix();
  PushMatrix();
  Translate(...);
  Rotate(...);
  left_upper_arm();
  PopMatrix();
  PushMatrix();
  update model-view matrix for head
  Rotate (...);
  head();
  PopMatrix();
  Recover original model-view matrix
  save it again
  update model-view matrix for left upper arm
  PopMatrix();
  Translate(...);
  Rotate(...);
  assert(...);
  left_upper_arm();
  PopMatrix();
  PushMatrix();
  recover and save original
  model-view matrix again
  rest of code
```

VRML

- Scene graph introduced by SGI Open Inventor
- Want to have a scene graph that can be used over the World Wide Web
- Need links to other sites to support distributed data bases
- Virtual Reality Markup Language
  - Based on Inventor data base
  - Implemented with OpenGL
VRML World Example
References