Hierarchical Models

Week 9, Lecture 19

David Breen
Department of Computer Science
Drexel University
Objectives

• Examine the limitations of linear modeling
  - Symbols and instances
• Introduce hierarchical models
  - Articulated models
  - Robots
• Introduce Tree and DAG models
Instance Transformation

• Start with a prototype object (a symbol)
• Each appearance of the object in the model is an instance
  - Must scale, orient, position
  - Defines instance transformation
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>$s_x$, $s_y$, $s_z$</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 + 4 identical wheels
  - Two symbols

• Rate of forward motion determined by rotational speed of wheels
Structure Through 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

![Graph Diagram]
Tree

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

```
root node
```

```
leaf node
```
Tree Model of Car

- Chassis
  - Right-front wheel
  - Left-front wheel
  - Right-rear wheel
  - Left-rear wheel
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

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

- **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: $R_b$
  - Apply $M = R_b$ to base
• Translate lower arm relative to base: $T_{lu}$
• Rotate lower arm around joint: $R_{lu}$
  - Apply $M = R_b T_{lu} R_{lu}$ to lower arm
• Translate upper arm relative to lower arm: $T_{uu}$
• Rotate upper arm around joint: $R_{uu}$
  - Apply $M = R_b T_{lu} R_{lu} T_{uu} R_{uu}$ to upper arm
mat4 ctm;  // current transformation matrix
robot_arm()
{
    ctm = RotateY(theta);
    base();
    ctm *= Translate(0.0, h1, 0.0);
    ctm *= RotateZ(phi);
    lower_arm();
    ctm *= Translate(0.0, h2, 0.0);
    ctm *= RotateZ(psi);
    upper_arm();
}
OpenGL Code for Robot

- At each level of hierarchy, calculate \( \text{ctm} \) matrix in application.
- Send matrix to shaders
- Apply \( \text{ctm} \) matrix in shader
- Draw geometry for one level of hierarchy
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
Possible Node Structure

- Code for drawing part or pointer to drawing function
- Linked list of pointers to children
- Matrix relating node to parent
Generalizations

• Need to deal with multiple children
  - How do we represent a more general tree?
  - How do we traverse such a data structure?

• Animation
  - How to use dynamically?
  - Can we create and delete nodes during execution?
Objectives

• Build a tree-structured model of a humanoid figure
• Examine various traversal strategies
• Build a generalized tree-model structure that is independent of the particular model
Humanoid Figure
Building the Model

• Can build a simple implementation using quadrics: ellipsoids and cylinders

• Access parts through functions
  – torso()
  – left_upper_arm()

• Matrices describe position of node with respect to its parent
  – \( \mathbf{M}_{lla} \) positions left lower arm with respect to left upper arm
Tree with Matrices

- Torso
  - Head
    - Left-lower arm
  - Left-upper arm
  - Right-upper arm
    - Left-lower arm
  - Right-lower arm
  - Left-upper leg
  - Right-upper leg
    - Left-lower leg
  - Right-lower leg
Display and Traversal

• The position of the figure is determined by 11 joint angles (two for the head and one for each other part)

• Display of the tree requires a graph traversal
  - Visit each node once
  - Display function at each node that describes the part associated with the node, applying the correct transformation matrix for position and orientation
There are 10 relevant matrices

- \( M \) positions and orients entire figure through the torso which is the root node
- \( M_h \) positions head with respect to torso
- \( M_{lua}, M_{rua}, M_{lul}, M_{rul} \) position arms and legs with respect to torso
- \( M_{lla}, M_{rla}, M_{lll}, M_{rll} \) position lower parts of limbs with respect to corresponding upper limbs
Stack-based Traversal

• Set model-view matrix to $\mathbf{M}$ and draw torso
• Set model-view matrix to $\mathbf{MM}_h$ and draw head
• For left-upper arm need $\mathbf{MM}_{lua}$ and so on
• Rather than recomputing $\mathbf{MM}_{lua}$ from scratch or using an inverse matrix, we can use the matrix stack to store $\mathbf{M}$ and other matrices as we traverse the tree
Traversal Code

```c
figure() {
    PushMatrix();
    torso();
    Rotate (...);
    head();
    PopMatrix();
    PushMatrix();
    Translate(...);
    Rotate(...);
    left_upper_arm();
    PopMatrix();
    PushMatrix();
}
```

- save present currents xform matrix
- update ctm for head
- recover original ctm
- save it again
- update ctm for left upper arm
- recover and save original ctm again
- rest of code
Analysis

• The code describes a particular tree and a particular traversal strategy
  - Can we develop a more general approach?

• Note that the sample code does not include state changes, such as changes to colors
  - May also want to use a *PushAttrib* and *PopAttrib* to protect against unexpected state changes affecting later parts of the code
General Tree Data Structure

• Need a data structure to represent tree and an algorithm to traverse the tree
• We will use a left-child right sibling structure
  - Uses linked lists
  - Each node in data structure has two pointers
  - Left: linked list of children
  - Right: next node (i.e. siblings)
Left-Child Right-Sibling Tree
Tree node Structure

• At each node we need to store
  - Pointer to sibling
  - Pointer to child
  - Pointer to a function that draws the object represented by the node
  - Homogeneous coordinate matrix to multiply on the right of the current model-view matrix
    • Represents changes going from parent to node
    • In OpenGL this matrix is a 1D array storing matrix by columns
C Definition of treenode

typedef struct treenode
{
    mat4 m;
    void (*f)();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
torso and head nodes

treenode torso_node, head_node, lua_node, ... ;

torso_node.m = RotateY(theta[0]);
torso_node.f = torso;
torso_node.sibling = NULL;
torso_node.child = &head_node;

head_node.m = translate(0.0, TORSO_HEIGHT+0.5*HEAD_HEIGHT, 0.0)*RotateX(theta[1])*RotateY(theta[2]);
head_node.f = head;
head_node.sibling = &lua_node;
head_node.child = NULL;
• The position of figure is determined by 11 joint angles stored in \texttt{theta[11]}
• Animate by changing the angles and redisplaying
• We form the required matrices using \texttt{Rotate} and \texttt{Translate}
  - Because the matrix is formed using the model-view matrix, we may want to first push original model-view matrix on matrix stack
void traverse(treenode* root)
{
    if(root==NULL) return;
    mvstack.push(ctm);
    ctm = ctm*root->m;
    root->f();
    if(root->child!=NULL) traverse(root->child);
    ctm = mvstack.pop();
    if(root->sibling!=NULL)
        traverse(root->sibling);
}
Traversal Code & Matrices

- `figure()` called with CTM set
- $M_{fig}$ defines figure’s place in world

```
figure() {
    PushMatrix();
torso();
    Rotate (...);
    head();
    PopMatrix();
    PushMatrix();
    Translate(...);
    Rotate(...);
    left_upper_arm();
}
```

<table>
<thead>
<tr>
<th>Stack</th>
<th>CTM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M_{fig}$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}M_h$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}$</td>
</tr>
<tr>
<td></td>
<td>$M_{fig}M_{lua}$</td>
</tr>
</tbody>
</table>
Traversing Code & Matrices

```plaintext
PushMatrix();
Translate(...);
Rotate(...);
left_lower_arm();
PopMatrix();
PopMatrix();
PushMatrix();
Translate(...);
Rotate(...);
right_upper_arm();
...  
...  
Stack
M_{fig}M_{lua}
M_{fig}
CTM
M_{fig}M_{lua}
Stack
M_{fig}M_{lua}
M_{fig}
CTM
M_{fig}M_{lua}M_{lla}
Stack
M_{fig}
CTM
M_{fig}M_{lua}
Stack
M_{fig}
CTM
M_{fig}
Stack
M_{fig}
CTM
M_{fig}M_{rua}
```
Notes

• We must save current transformation matrix before multiplying it by node matrix
  - Updated matrix applies to children of node but not to siblings which contain their own matrices
• The traversal program applies to any left-child right-sibling tree
  - The particular tree is encoded in the definition of the individual nodes
• The order of traversal matters because of possible state changes in the functions
Homework 7

• Create models for links (P0, P1, P2 & P3)
• Draw base model (P0)
• \( M = T_z(L0) \cdot R_z(\theta_1) \) \( \equiv \) matrix multiply
• Apply transformation matrix M to first link model (P1): \( P1' = M \cdot P1 \)
• Draw P1'
• \( M = M \cdot (T_z(L1) \cdot R_y(\theta_2)) \)
• \( P2' = M \cdot P2 \) // Apply matrix to second link
• Draw P2'
• $M = M \cdot (T_z(L2) \cdot R_y(\theta 3))$
• $P3' = M \cdot P3$  // Apply matrix to third link
• Draw $P3'$
• $M = M \cdot T_z(L3)$
• Extract translation vector from $M$ as the position for drawing sphere at end of arm