Topics

Hierarchical modeling
  • Examine the limitations of linear modeling
    – Symbols and instances
So far, we have discussed modeling simple geometrical objects.

How can we generate a complicated object, e.g., a robot, which is made up from several parts?

Construct complex objects from a collection of basic objects.
**Instance Transformation**

Start with a prototype object (a *symbol*), e.g.,
- Geometric objects
- Fonts

Each appearance of the object in the model is an *instance*
- A instance transformation from model frame to world frame by scaling, rotation, and translation

\[ M = TRS \]

Model frame  

World frame
Instance Transformation

A model view matrix consists of

- instance transformation and
- a transformation from the world frame to the eye frame

```c
mat4 instance; mat4 model_view;

instance = Translate(dx, dy, dz)*RotateZ(rz)*RotateY(ry)*RotateX(rx)*Scale(sx, sy, sz);
model_view = model_view*instance;
```
Symbol-Instance Table

Can store a model by assigning a number to each symbol and storing the parameters for the instance transformation.

What’s the problem with the table?

A flat structure - each symbol is processed independently.

<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

Represent the relationships among different parts 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
Graphs: Tree

Tree is a directed acyclic graph (DAG)

A directed graph in which each node (except the root) has exactly one parent node

- No loops
- 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
DAG Model

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)
A robot arm consists of two parallelepipeds and a cylinder.
Articulated Models

Robot arm is an example of an *articulated model*

- Parts connected at joints
- Three degrees of freedom described by
  - joint angles measured in its local frame
  - Angle between the base and the ground
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

Base:
- Rotation of base: $R_b$
  - Apply $M = R_b$ to base

Lower arm:
- 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

Upper 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
OpenGL Code for Robot

mat4 ctm;
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 Base

```c
void base()
{
    mat4 instance = ( Translate( 0.0, 0.5 * BASE_HEIGHT, 0.0 ) * Scale( BASE_WIDTH, BASE_HEIGHT, BASE_WIDTH ) );

    glUniformMatrix4fv( ModelView, 1, GL_TRUE, model_view * instance );

    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
}
```
OpenGL Code for Robot

The lower arm and the upper arm are modeled similar to the base.

All the three parts are modeled based on cubes – the same symbol.

Only one set of vertices are needed to send to the buffer!
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

-- storing all information in nodes

Possible Node Structure

- Code for drawing part or pointer to drawing function
- A matrix relating node to parent
- Linked list of pointers to children
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?
Humanoid Figure
Building the Model

Can build a simple implementation using quadrics:

• ellipsoids and cylinders

Access parts through functions drawing individual parts in their own frames

• torso()
• left_upper_arm()

Matrices describe position of node with respect to its parent

• $M_{lla}$ positions left lower arm with respect to left upper arm
Tree with Matrices

```
           Torso
          /       \
   M_h       M_lua     M_rua
   /       \        \       \
Head       Left-upper arm Right-upper arm Left-upper leg Right-upper leg

   /       \\       \\
M llia    M rli     M ll     M rll
Left-lower arm Right-lower arm Left-lower leg Right-lower leg
```