CSCI 480 Computer Graphics
Lecture 6

Hierarchical Models

Projections and Shadows
Hierarchical Models
[Angel Ch 5.10, 10.1 - 10.6]

January 31, 2011
Jernej Barbic
University of Southern California
http://www-bcf.usc.edu/~jbarbic/cs480-s11/

Roadmap

• Last lecture: Viewing and projection
• Today:
  – Shadows via projections
  – Hierarchical models
• Next: Polygonal Meshes, Curves and Surfaces
  • Goal: background for Assignment 2 (next week)

Importance of shadows

Without shadows

With shadows

Source: UNC
**Doom III**

Reported to spend 50% of time rendering shadows!

**Light sources**

- Point light source
- Directional light source
- Area light source

**Hard and soft shadows**

- Hard shadow
- Soft shadow

**Shadow Algorithms**

- With visibility tests
  - Accurate yet expensive
  - Example: ray casting or ray tracing
  - Example: 2-pass z-buffer
  - Example: 2-pass z-buffer
    [Foley, Ch. 16.4.4] [RTR 6.12]

- Without visibility tests (“fake” shadows)
  - Approximate and inexpensive
  - Using projection in model-view matrix

**Shadows via Projection**

- Assume light source at \([x_l, y_l, z_l]^T\)
- Assume shadow on plane \(y = 0\)
- Viewing \(\sim\) shadow projection
  - Center of projection \(\sim\) light
  - Viewing plane \(\sim\) shadow plane
- View plane in front of object
- Shadow plane behind object

**Shadow Projection Strategy**

- Move light source to origin
- Apply appropriate projection matrix
- Move light source back
- Instance of general strategy: compose complex transformation from simpler ones!

\[
T = \begin{bmatrix}
1 & 0 & 0 & -x_l \\
0 & 1 & 0 & -y_l \\
0 & 0 & 1 & -z_l \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Derive Equation

- Now, light source at origin

\[ \frac{x_p}{y_p} = \frac{x}{y} \]  
\[ y_p = -y_l \]  

(see picture)

\[ x_p = \frac{x}{y}, y_p = -\frac{x}{y/y_l}, z_p = \frac{z}{y/y_l} \]

Light Source at Origin

- After translation, solve

\[ M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = W \begin{bmatrix} -\frac{x}{y/y_l} \\ -\frac{y}{y_l} \\ \frac{z}{y/y_l} \\ 1 \end{bmatrix} \]

- \( w \) can be chosen freely
- Use \( w = -y/y_l \)

Shadow Projection Matrix

- Solution of previous equation

\[ M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -\frac{1}{y_l} & 0 & 0 \end{bmatrix} \]

- Total shadow projection matrix

\[ S = T^{-1}MT = \ldots \]

Implementation

- Recall column-major form
  
  \[ \text{GLfloat} m[16] = \{ 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, -1.0/y_l, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0 \}; \]

- Assume \( \text{drawPolygon}(); \) draws object

Saving State

- Assume \( x_l, y_l, z_l \) hold light coordinates
  
  \begin{verbatim}
  glMatrixMode(GL_MODELVIEW);
  drawPolygon(); /* draw normally */
  glPushMatrix(); /* save current matrix */
  glTranslate(x_l, y_l, z_l); /* translate back */
  glMultMatrixf(m); /* project */
  glTranslate(-x_l, -y_l, -z_l); /* move light to origin */
  drawPolygon(); /* draw polygon again for shadow */
  glPopMatrix(); /* restore original transformation */
  ...
  \end{verbatim}

The Matrix and Attribute Stacks

- Mechanism to save and restore state
  - \( \text{glPushMatrix}(); \)
  - \( \text{glPopMatrix}(); \)

- Apply to current matrix

- Can also save current attribute values
  - Examples: color, lighting
    - \( \text{glPushAttrib(GLbitfield mask);} \)
    - \( \text{glPopAttrib();} \)
    - Mask determines which attributes are saved
Drawing on a Surface

• Shimmering when drawing shadow on surface
• Due to limited precision depth buffer
• Either displace surface or shadow slightly (glPolygonOffset in OpenGL)

Or use general technique
1. Set depth buffer to read-only, draw surface
2. Set depth buffer to read-write, draw shadow
3. Set color buffer to read-only, draw surface again
4. Set color buffer to read-write

Outline

• Projections and Shadows
• Hierarchical Models

Hierarchical Models

• Many graphical objects are structured
• Exploit structure for
  – Efficient rendering
  – Example: bounding boxes (later in course)
  – Concise specification of model parameters
  – Example: joint angles
  – Physical realism
• Structure often naturally hierarchical

Instance Transformation

• Often we need several instances of an object
  – Wheels of a car
  – Arms or legs of a figure
  – Chess pieces

• Instances can be shared across space or time
• Encapsulate basic object in a function
• Object instances are created in “standard” form
• Apply transformations to different instances
• Typical order: scaling, rotation, translation
Sample Instance Transformation

```c
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(...);
glRotatef(...);
glScalef(...);
gluCylinder(...);
```

Display Lists

- Sharing display commands
- Display lists are stored on the GPU
- May contain drawing commands and transfns.
- Initialization:
  ```c
  GLuint torus = glGenLists(1);
gNewList(torus, GL_COMPILE);
  Torus(8, 25);
  glEndList();
  ```
- Use: glCallList(torus);
- Can share both within each frame, and across different frames in time

Display Lists Caveats

- Store only values of expressions
- Display lists cannot be changed or updated
- Only store commands that change server state
- Effect of executing display list depends on current transformations and attributes
- They are deprecated:
  - for complex usage, use Vertex Buffer Object OpenGL extension instead
- Display lists may be hierarchical
  - One list may call another
  - Can be useful for hierarchical objects
  - Some implementation-dependent nesting limit

Drawing a Compound Object

- Example: simple "robot arm"
- Base rotation $\theta$, arm angle $\phi$, joint angle $\psi$

Interleave Drawing & Transformation

- $h_1 = \text{height of base}, h_2 = \text{length of lower arm}$
- void drawRobot(GLfloat theta, GLfloat phi, GLfloat psi) {
  glRotatef(theta, 0.0, 1.0, 0.0);
  drawBase();
  glTranslatef(0.0, h1, 0.0);
  glRotatef(phi, 0.0, 0.0, 1.0);
  drawLowerArm();
  glTranslatef(0.0, h2, 0.0);
  glRotatef(psi, 0.0, 0.0, 1.0);
  drawUpperArm();
}

Assessment of Interleaving

- Compact
- Correct "by construction"
- Efficient
- Inefficient alternative:
  ```c
  glPushMatrix();
  glRotatef(theta, ...);
  drawBase();
  glPopMatrix();
  ```
- Count number of transformations
Hierarchical Objects and Animation

- Drawing functions are time-invariant
  - `drawBase();` `drawLowerArm();` `drawUpperArm();`
- Can be easily stored in display list
- Change parameters of model with time
- Redraw when idle callback is invoked

A Bug to Watch

```c
GLfloat theta = 0.0; /* update in idle callback */
GLfloat phi = 0.0; /* update in idle callback */
GLuint arm = glGenLists(1); /* in init function */
glNewList(arm, GL_COMPILE);
  glRotatef(theta, 0.0, 1.0, 0.0);
  drawBase();
  drawUpperArm();
  glEndList(); /* in display callback */
glCallList(arm);
```

More Complex Objects

- Tree rather than linear structure
- Interleave along each branch
- Use push and pop to save state

Hierarchical Tree Traversal

- Order not necessarily fixed
- Example:

```c
void drawFigure()
{
  glPushMatrix(); /* save */
  glPushMatrix();
  glPushMatrix(); /* move head */
  glTranslatef(...);
  glRotatef(...);
  drawHead();
  glPopMatrix(); /* restore */
  drawLeftUpperArm();
  glPushMatrix(); /* rotate head */
  glTranslatef(...);
  glRotatef(...);
  drawLeftLowerArm();
  glPopMatrix();
  ...
}
```

Using Tree Data Structures

- Can make tree form explicit in data structure

```c
typedef struct treenode
{
  GLfloat m[16];
  void (*f)( );
  struct treenode *sibling;
  struct treenode *child;
} treenode;
```

Initializing Tree Data Structure

- Initializing transformation matrix for node
- Initializing pointers

```c
treenode torso, head, ...;
/* in init function */
  glLoadIdentity();
  glGetFloatv(GL_MODELVIEW_MATRIX, torso.m);
  torso.f = drawTorso;
  torso.sibling = NULL;
  torso.child = &head;
```
Generic Traversal

- Recursive definition
  
  ```c
  void traverse (treenode *root)
  {
    if (root == NULL) return;
    glPushMatrix();
    glMultMatrixf(root->m);
    root->f();
    if (root->child != NULL) traverse(root->child);
    glPopMatrix();
    if (root->sibling != NULL) traverse(root->sibling);
  }
  ```

- C is really not the right language for this

Summary

- Projections and Shadows
- Hierarchical Models

Notes

- Wednesday – polygonal meshes, curves and surfaces
- Assignment 1 is due in one week