Hierarchical Models

Projections and Shadows
Hierarchical Models
Basic Animation

[Angel Ch 5.10, 9.1-9.6]

January 30, 2003
Frank Pfenning
Carnegie Mellon University

http://www.cs.cmu.edu/~fp/courses/graphics/
Roadmap

• Last lecture: Viewing and projection
• Today:
  – Shadows via projections
  – Hierarchical models
  – Basic animation
• Next: lighting and material properties
• Goal: background for Assignment 3 (next week)
Shadow Algorithms

• With visibility tests
  – Accurate yet expensive
  – Example: ray casting or ray tracing
  – 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
  – Examples: flight simulator, Assignment 3
Shadows via Projection

• Assume light source at \([x_l \ y_l \ z_l \ 1]^T\)
• Assume shadow on plane \(y = 0\)
• Viewing ~ shadow projection
  – Center of projection ~ light
  – Viewing plane ~ 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

\[
\frac{x_p}{y_p} = \frac{x}{y} \quad \text{(see picture)}
\]

\[
y_p = -y_l \quad \text{(moved light)}
\]

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

\[
z_p = \frac{z}{y} y_p = \frac{x}{y / y_l}
\]
Light Source at Origin

• After translation, solve

\[
\begin{bmatrix}
x \\ y \\ z \\ 1 
\end{bmatrix}
\begin{bmatrix}
x & -y/y_1 \\ y & -y_1 \\ z & z \\ 1 & y/y_1 
\end{bmatrix}
= w
\]

• \( w \) can be chosen freely
• Use \( w = -y/y_1 \)
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 drawPolygon(); draws object
Saving State

• Assume xl, yl, zl hold light coordinates

```c
glmMatrixMode(GL_MODELVIEW);
drawPolygon();    /* draw normally */

glmPushMatrix();   /* save current matrix */
glTranslatef(xl, yl, zl);   /* translate back */
glMultMatrixf(m);       /* project */
glTranslatef(-xl, -yl, -zl); /* move light to origin */
drawPolygon();   /* draw polygon again for shadow */
glPopMatrix();   /* restore original transformation */
...
```
The Matrix and Attribute Stacks

- Mechanism to save and restore state
  - `glPushMatrix();`
  - `glPopMatrix();`
- Apply to current matrix
- Can also save current attribute values
  - Examples: color, lighting
  - `glPushAttrib(GLbitfield mask);`
  - `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 special properties of scene
• 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
• Basic Animation
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 server
- May contain drawing commands and transfns.
- Initialization:
  
  ```
  GLuint torus = glGenLists(1);
  glNewList(torus, GL_COMPILE);
  Torus(8, 25);
  glEndList();
  ```

- Use: glCallList(torus);
- In animation, can also share at different times
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
• 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

- $h_1 = \text{height of base}, \quad h_2 = \text{length of lower arm}$

```c
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();

... etc...
```

- 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

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();

What is wrong?

   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 */
    drawTorso();
    glPushMatrix();
        glTranslatef(...); /* move head */
        glRotatef(...); /* rotate head */
        drawHead();
    glPopMatrix(); /* restore */
    glPopMatrix(); /* restore */
    drawUpperArm();
    glTranslatef(...);
    drawLowerArm();
}
```
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

treenode torso, head, ...;
/* in init function */
glLoadIdentity();
glRotatef(...);
glGetFloatv(GL_MODELVIEW_MATRIX, torso.m);

• Initializing pointers

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
Outline

• Projections and Shadows
• Hierarchical Models
• Basic Animation
Unified View of Computer Animation

• Models with parameters
  – Polygon positions, control points, joint angles, ...
  – $n$ parameters define $n$-dimensional state space

• Animation defined by path through state space
  – Define initial state, repeat:
    – Render the image
    – Move to next point (following motion curves)

• Animation = specifying state space trajectory
Animation vs Modeling

- Modeling: what are the parameters?
- Animation: how do we vary the parameters?
- Sometimes boundary not clear
- Build models that are easy to control
- Hierarchical models often easy to control
Basic Animation Techniques

• Traditional (frame by frame)
• Keyframing
• Procedural techniques
• Behavioral techniques
• Performance-based (motion capture)
• Physically-based (dynamics)
Traditional Cel Animation

- Film runs at 24 frames per second (fps)
- Video at 30 frames per second
- Production process critical: render farms
- Artistic issues: story and style
Traditional Animation Process

• Story board: sequence of sketches with story
• Key frames
  – Important frames as line drawings
  – Motion-based description
  – Example: beginning of stride, end of stride
• Inbetweens: draw remaining frames
• Painting: redraw onto acetate cels, color them
Layered Motion

- Multiple layers of animation
  - Reuse background
  - Multiple parallel animators
  - Supported by transparent acetate for drawing
- Also used in computer animation
- Example: painters algorithm for hidden surface removal
Storyboard Examples [A Bug’s Life]
Computer Assisted Animations

• Eliminate human labor, bottom to top
• Computerized cel painting
  – Digitize line drawing, color using seed fill
  – Widely used in production (e.g., Lion King)
• Cartoon inbetweening
  – Interpolate between two drawings (morphing)
  – Difficult to make look natural
  – Choice of parameters?
  – Rarely used in production
True Computer Animations

- Generate images by rendering a 3D model
- Vary parameters to produce animation
- Brute force
  - Manually set the parameters for every frame
  - $1440n$ values per minute for $n$ parameters
  - Maintenance problem
- Computer keyframing
  - Lead animators create important frames
  - Computers draw inbetweens from 3D(!)
  - Dominant production method
Example: From Toy Story
Some Research Issues

- **Inverse kinematics**
  - How to plot a path through state space
  - Multiple degrees of freedom
  - Also important in robotics

- **Physical accuracy**
  - Collision detection
  - Computer graphics: only needs to look right
  - Simulation: must follow model correctly
Summary

- Projections and Shadows
- Hierarchical Models
- Basic Animation
Preview

• Tuesday – lighting and shading
• Assignment 2 out today
• Due in one week (Thursday, before lecture)