CSCI 420 Computer Graphics
Lecture 8

Hierarchical Models

Projections and Shadows
Hierarchical Models
[Angel Ch. 8]

Jernej Barbic
University of Southern California

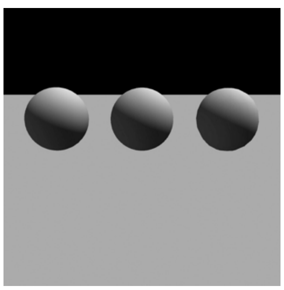
1

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)

2

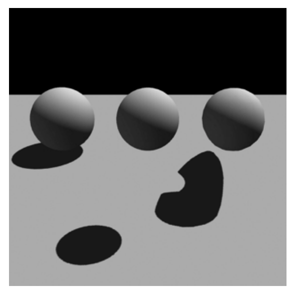
Importance of shadows



Source: UNC

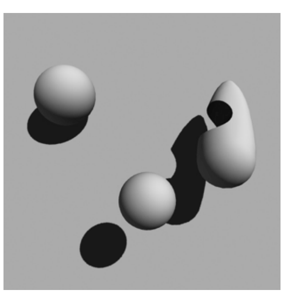
3

Importance of shadows



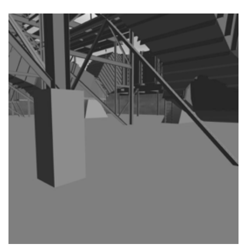
4

Importance of shadows

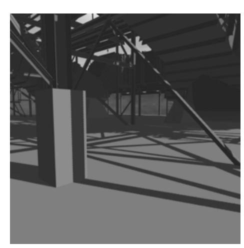


5

Importance of shadows



Without shadows



With shadows

Source: UNC

6

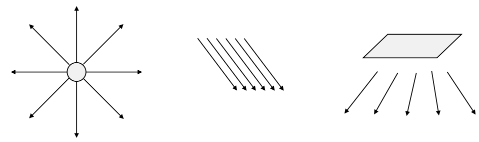
Doom III

Reported to spend 50% of time rendering shadows!

Source: Wikipedia

7

Light sources

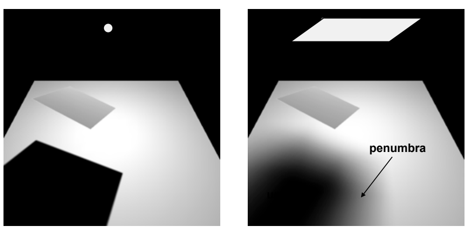


point light source | directional light source | area light source

8

Hard and soft shadows

area light



Source: UNC

Hard shadow | Soft shadow

9

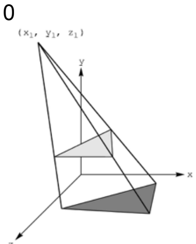
Shadow Algorithms

- 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 a model-view matrix "trick"

10

Shadows via Projection

- Assume light source at $[x_l \;\; y_l \;\; z_l]^T$
- 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



11

Shadow Projection Strategy

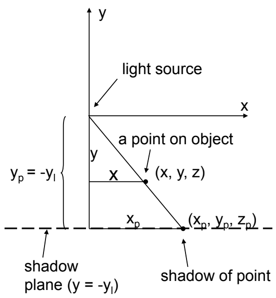
- Move light source to origin
- Apply appropriate projection matrix
- Move light source back
- Instance of general strategy: compose complex transformation from simpler ones!

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_l \\ 0 & 1 & 0 & -y_l \\ 0 & 0 & 1 & -z_l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

12

Derive Equation

- Now, light source at origin



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

$$y_p = -y_l \quad \text{(move light)}$$

$$x_p = \frac{x}{y} y_p = -\frac{x}{y} y_l$$

$$z_p = \frac{z}{y} y_p = -\frac{z}{y} y_l$$

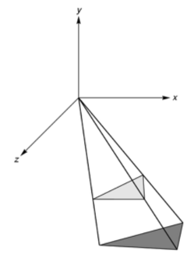
Light Source at Origin

- After translation, solve

$$M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = w \begin{bmatrix} -\frac{x y_l}{y} \\ -y_l \\ -\frac{z y_l}{y} \\ 1 \end{bmatrix}$$

- w can be chosen freely
- Use $w = -y / y_l$

$$M \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -\frac{y}{y_l} \end{bmatrix}$$



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

$$S = T^{-1} M T = \ldots$$

Implementation

- Recall column-major form

```
GLfloat m[16] =
{1.0, 0.0, 0.0, 0.0,
  0.0, 1.0, 0.0, -1.0 / yl,
  0.0, 0.0, 1.0, 0.0,
  0.0, 0.0, 0.0, 0.0};
```

- yl is light source height
- Assume drawPolygon(); draws object

Saving the ModelView Matrix State

- Assume xl, yl, zl hold light coordinates
- Core OpenGL code (compatibility code is similar)

```
openGLMatrix->MatrixMode(OpenGLMatrix::ModelView);
// here, set the model view matrix, in the usual way
// …

drawPolygon();    // draw normally
openGLMatrix->PushMatrix();    // save current matrix
openGLMatrix->Translate(xl, yl, zl);      // translate back
openGLMatrix->MultMatrix(m);        // project
openGLMatrix->Translate(-xl, -yl, -zl);  // move light to origin

float ms[16];
openGLMatrix->GetMatrix(ms); // read the shadow matrix
```


Saving the ModelView Matrix State (cont.)

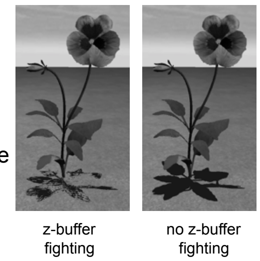
```
// upload the shadow matrix to the GPU
glUniformMatrix4fv(h_modelViewMatrix, 1, GL_FALSE, ms);

drawPolygon();    // draw polygon again for shadow

// restore original modelview matrix
openGLMatrix->PopMatrix();
openGLMatrix->GetMatrix(ms);
glUniformMatrix4fv(h_modelViewMatrix, 1, GL_FALSE, ms);

// continue rendering more objects, as usual …
```

The Matrix and Attribute Stacks

- Mechanism to save and restore state
  - {OpenGLMatrix::, gl}PushMatrix();
  - {OpenGLMatrix::, gl}PopMatrix();
- Apply to current matrix
- In compatibility profile, can also save current attribute values
  - Examples: color, lighting
  - glPushAttrib(GLbitfield mask);
  - glPopAttrib();
  - Mask determines which attributes are saved
  - This feature has been removed in the core profile

Drawing on a Surface

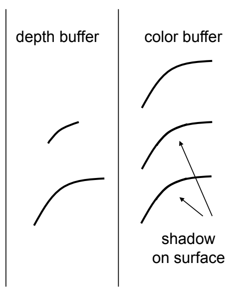
- Shimmering ("z-buffer fighting") when drawing shadow on surface
- Due to limited precision of depth buffer
- Solution: slightly displace either the surface or the shadow
  (glPolygonOffset in OpenGL)



Drawing on a Surface

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

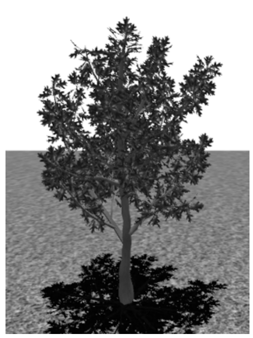


Outline

- Projections and Shadows
- Hierarchical Models

Hierarchical Models

- Many graphical objects are structured
- Exploit structure for
  - Efficient rendering
  - Example: tree leaves
  - Concise specification of model parameters
  - Example: joint angles
  - Physical realism
- Structure often naturally hierarchical



Instance Transformation

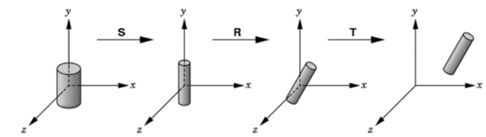
- Often we need several instances of an object
  - Wheels of a car
  - Arms or legs of a figure
  - Chess pieces

Instance Transformation

- Instances can be shared across space or time
- Write a function that renders the object in "standard" configuration
- Apply transformations to different instances
- Typical order: scaling, rotation, translation

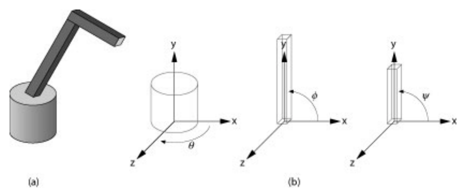
Sample Instance Transformation

```
openGLMatrix->MatrixMode(OpenGLMatrix::ModelView);
openGLMatrix->LoadIdentity();
openGLMatrix->Translate(...);
openGLMatrix->Rotate(...);
openGLMatrix->Scale(...);
// … upload modelview matrix to GPU, as usual …
renderCylinder(...);
```



Drawing a Compound Object

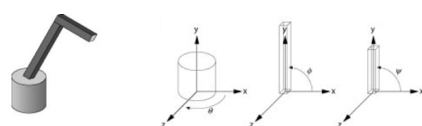
- Example: simple "robot arm"



Base rotation θ, arm angle ϕ, joint angle ψ

Hierarchical Objects and Animation

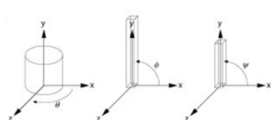
- Drawing functions are time-invariant and draw the object in a canonical position:
  drawBase(); drawLowerArm(); drawUpperArm();
- Can be easily stored in a VBO
- Change parameters of model with time



Interleave Drawing & Transformation

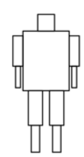
- h1 = height of base, h2 = length of lower arm
- This is pseudocode (must upload matrix to GPU)

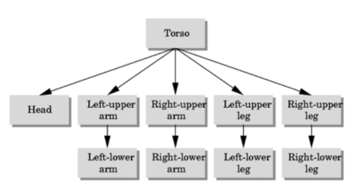
```
void drawRobot(GLfloat theta,
   GLfloat phi, GLfloat psi)
{
   Rotate(theta, 0.0, 1.0, 0.0);
   drawBase();
   Translate(0.0, h1, 0.0);
   Rotate(phi, 0.0, 0.0, 1.0);
   drawLowerArm();
   Translate(0.0, h2, 0.0);
   Rotate(psi, 0.0, 0.0, 1.0);
   drawUpperArm();
}
```



More Complex Objects

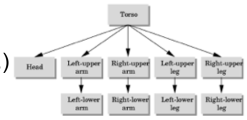
- Tree rather than linear structure
- Interleave along each branch
- Use push and pop to save state





Hierarchical Tree Traversal

- Order not necessarily fixed (breadth-first, depth-first, etc.)
- Example:




```
void drawFigure()
{
  PushMatrix();  // save
  drawTorso();
  Translate(...); // move head
  Rotate(...);     // rotate head
  drawHead();
  PopMatrix();   // restore
  PushMatrix();
  Translate(...);
  Rotate(...);
  drawLeftUpperArm();
  Translate(...)
  Rotate(...)
  drawLeftLowerArm();
  PopMatrix();
  ... }
```


Using Tree Data Structures

- Can make tree form explicit in data structure

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


Initializing Tree Data Structure

- Initializing transformation matrix for node

```
treenode torso, head, ...;
// in init function
LoadIdentity();
Rotate(...);
GetMatrix(torso.m);
```

- Initializing pointers

```
torso.render = drawTorso;
torso.sibling = NULL;
torso.child = &head;
```


Generic Traversal: Recursion

```
void traverse (treenode *root)
{
  if (root == NULL)
    return;
  PushMatrix();
  MultMatrix(root->m);
  root->render();
  if (root->child != NULL)
    traverse(root->child);
  PopMatrix();
  if (root->sibling != NULL)
    traverse(root->sibling);
}
```


Summary

- Projections and Shadows
- Hierarchical Models

Notes

- Next lecture: polygonal meshes, curves and surfaces
- Assignment 1 is due in one week