

Hierarchical Models

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

January 30, 2012
Jernej Barbic
University of Southern California
<http://www.bcf.usc.edu/~jbarbic/cs480-s12/>

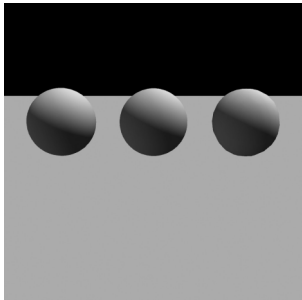
1

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

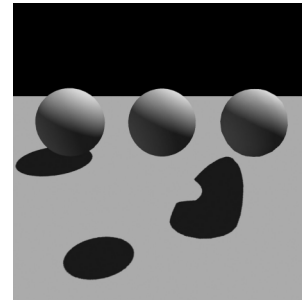
Importance of shadows



Source: UNC

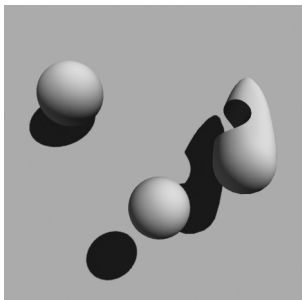
3

Importance of shadows



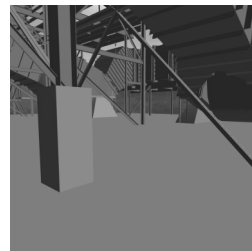
4

Importance of shadows



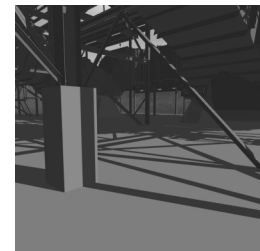
5

Importance of shadows



Without shadows

Source: UNC



With shadows

6

Doom III

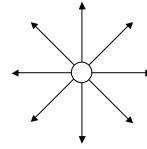


Source: Wikipedia

Reported to spend 50% of time rendering shadows!

7

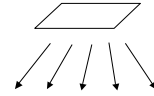
Light sources



point light source



directional light source



area light source

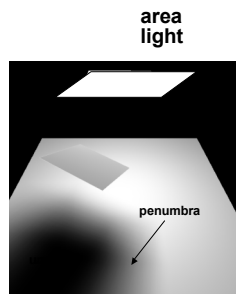
8

Hard and soft shadows



Source: UNC

Hard shadow



Soft shadow

9

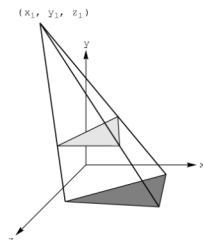
Shadow Algorithms

- 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

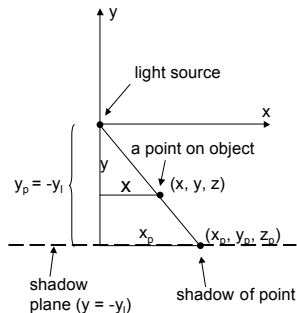
- Move light source to origin
- Apply appropriate projection matrix
- Move light source back
- Instance of general strategy: compose complex transformation from simpler ones!

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

12

Derive Equation

- Now, light source at origin



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

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

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

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

13

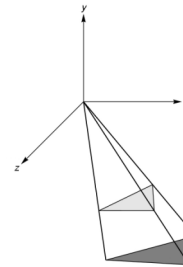
Light Source at Origin

- 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 $w = -y/y_l$

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



14

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

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

15

Implementation

- Recall column-major form

```
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};
```

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

16

Saving State

- Assume x_l, y_l, z_l hold light coordinates

```
glMatrixMode(GL_MODELVIEW);
drawPolygon(); /* draw normally */

glPushMatrix(); /* save current matrix */
glTranslatef(x_l, y_l, z_l); /* translate back */
glMultMatrixf(m); /* project */
glTranslatef(-x_l, -y_l, -z_l); /* move light to origin */
drawPolygon(); /* draw polygon again for shadow */
glPopMatrix(); /* restore original transformation */
...
```

17

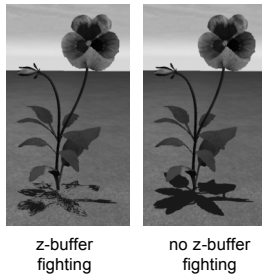
The Matrix and Attribute Stacks

- 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

18

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)

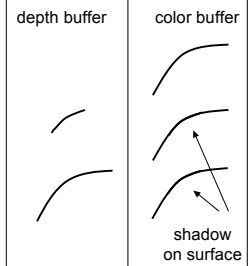


19

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



20

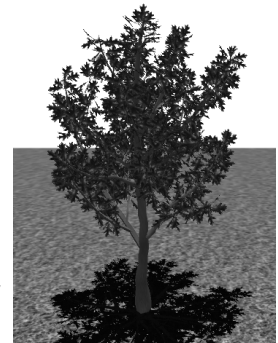
Outline

- Projections and Shadows
- Hierarchical Models

21

Hierarchical Models

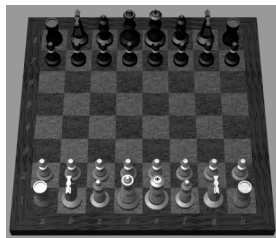
- Many graphical objects are structured
- Exploit structure for
 - Efficient rendering
 - Example: tree leaves
 - Concise specification of model parameters
 - Example: joint angles
 - Physical realism
- Structure often naturally hierarchical



22

Instance Transformation

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



23

Instance Transformation

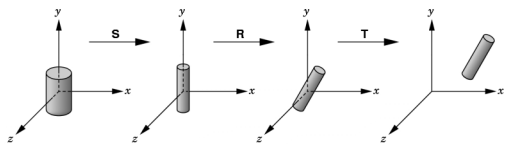
- Instances can be shared across space or time
- Write a function that renders the object in “standard” configuration
- Apply transformations to different instances
- Typical order: scaling, rotation, translation



24

Sample Instance Transformation

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



25

Display Lists

- Sharing display commands
- Display lists are stored on the GPU
- May contain drawing commands and transfn's.
- Initialization:

```
GLuint torus = glGenLists(1);
glNewList(torus, GL_COMPILE);
    Torus(8, 25);
glEndList();
```

- Use: `glCallList(torus);`
- Can share both within each frame, and across different frames in time
- Can be hierarchical: a display list may call another

26

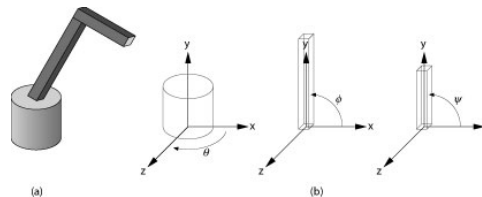
Display Lists Caveats

- Store only results of expressions, not the expressions themselves
- Display lists cannot be changed or updated
- Effect of executing display list depends on current transformations and attributes
- Some implementation-dependent nesting limit
- They are deprecated:
 - for complex usage, use Vertex Buffer Object OpenGL extension instead

27

Drawing a Compound Object

- Example: simple “robot arm”



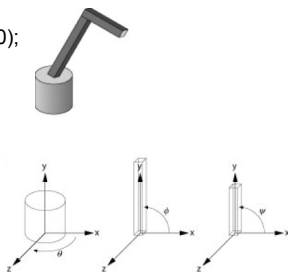
Base rotation θ , arm angle ϕ , joint angle ψ

28

Interleave Drawing & Transformation

- $h1$ = height of base, $h2$ = 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();
}
```



29

## Assessment of Interleaving

- Compact
- Correct “by construction”
- Efficient
- Inefficient alternative:
 

```
glPushMatrix(); glPushMatrix(); ...etc...
glRotatef(theta, ...); glRotatef(theta, ...);
drawBase(); glTranslatef(...);
glPopMatrix(); glRotatef(phi, ...);
 drawLowerArm();
 glPopMatrix();
```
- Count number of transformations

30

## 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

31

## A Bug to Watch

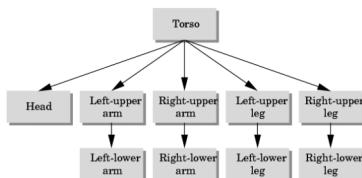
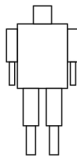
```
GLfloat theta = 0.0; ...; /* update in idle callback */
GLfloat phi = 0.0; ...; /* update in idle callback */
GLuint arm = glGenLists(1);
/* in init function */
glNewList(arm, GL_COMPILE);
 glRotatef(theta, 0.0, 1.0, 0.0);
 drawBase();
 ...
 drawUpperArm();
glEndList();
/* in display callback */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

What is wrong?

32

## More Complex Objects

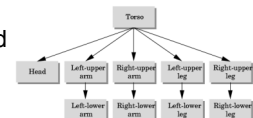
- Tree rather than linear structure
- Interleave along each branch
- Use push and pop to save state



33

## Hierarchical Tree Traversal

- Order not necessarily fixed
- Example:



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

 glPushMatrix();
 glTranslatef(...);
 glRotatef(...);
 drawLeftUpperArm();
 glTranslatef(...);
 glRotatef(...);
 drawLeftLowerArm();
 glPopMatrix();
 ...
}
```

34

## Using Tree Data Structures

- Can make tree form explicit in data structure
- ```
typedef struct treenode
{
    GLfloat m[16];
    void (*f) ();
    struct treenode *sibling;
    struct treenode *child;
} treenode;
```

35

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;
```

36

Generic Traversal

- Recursive definition

```
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

37

Summary

- Projections and Shadows
- Hierarchical Models

38

Notes

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

39