CSCI 420 Computer Graphics

Lecture 6

Hierarchical Models

Projections and Shadows

Hierarchical Models

[Angel Ch 5.10, 10.1-10.6]

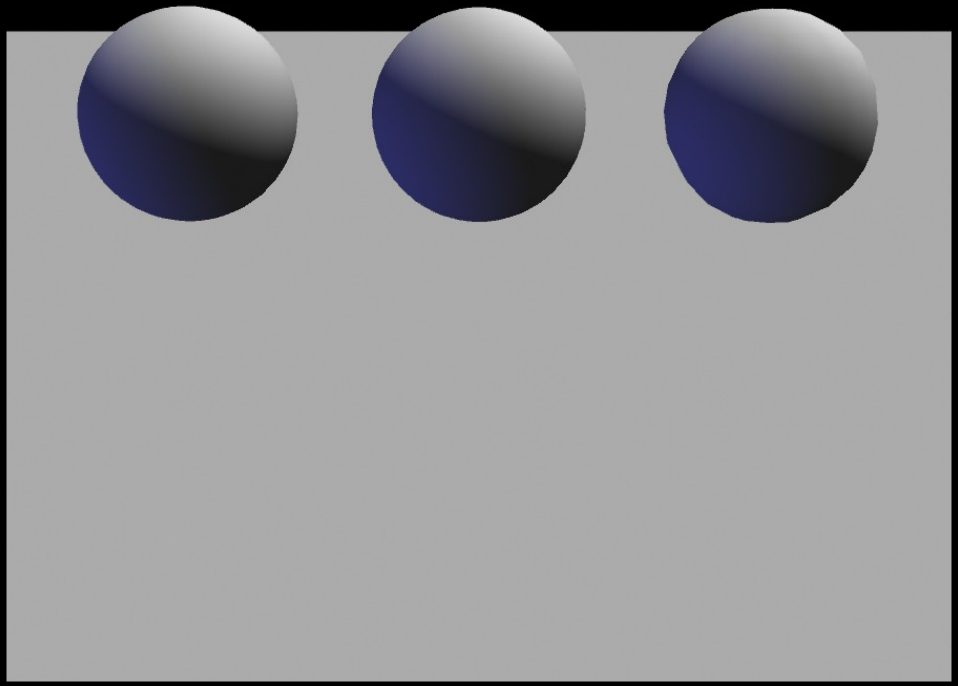
Jernej Barbic

University of Southern California

Roadmap

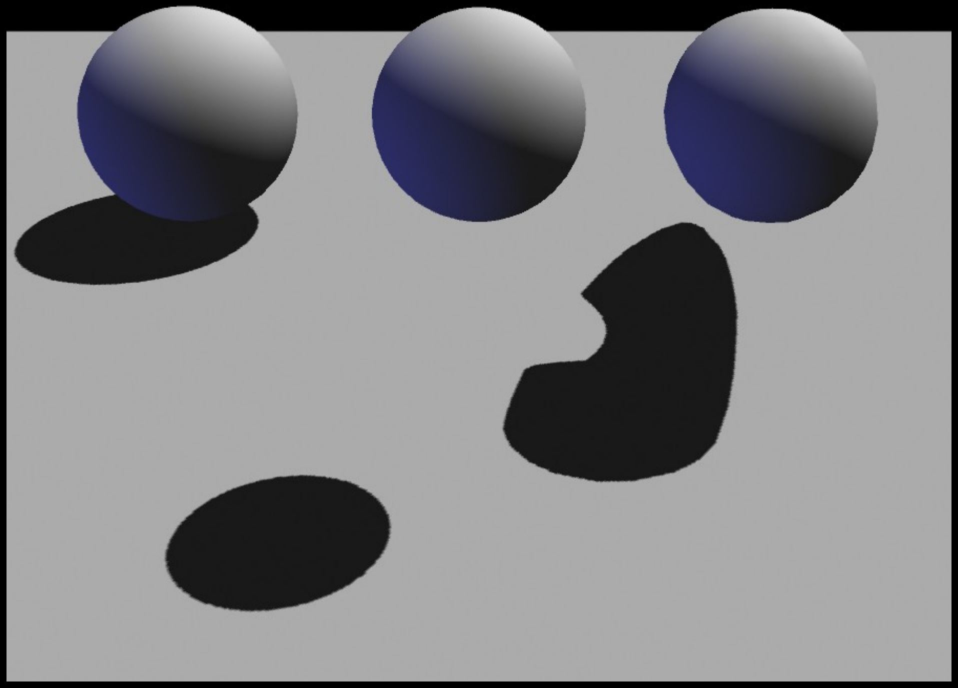
- Last lecture: Viewing and projection
- Today:
  - Shadows via projections
  - Hierarchical models
- Next: Polygonal Meshes, Curves and Surfaces
- Goal: background for Assignment 2 (next week)

Importance of shadows

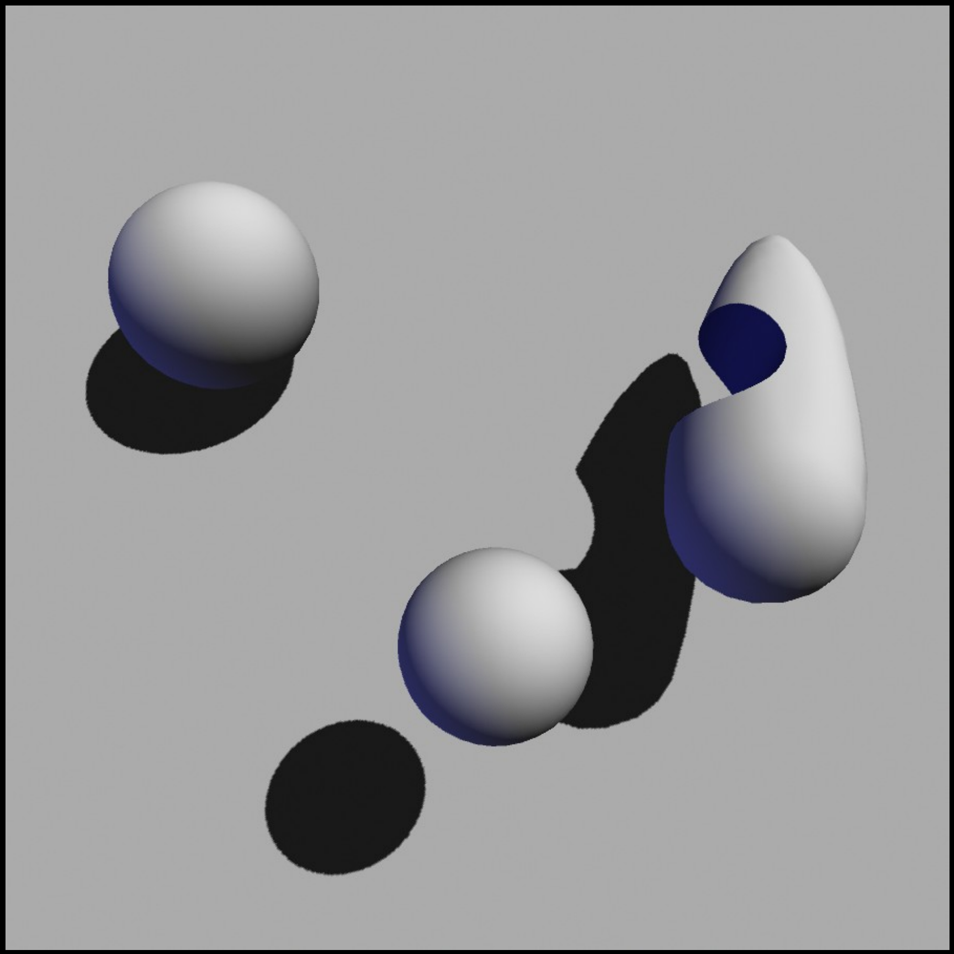


Source: UNC

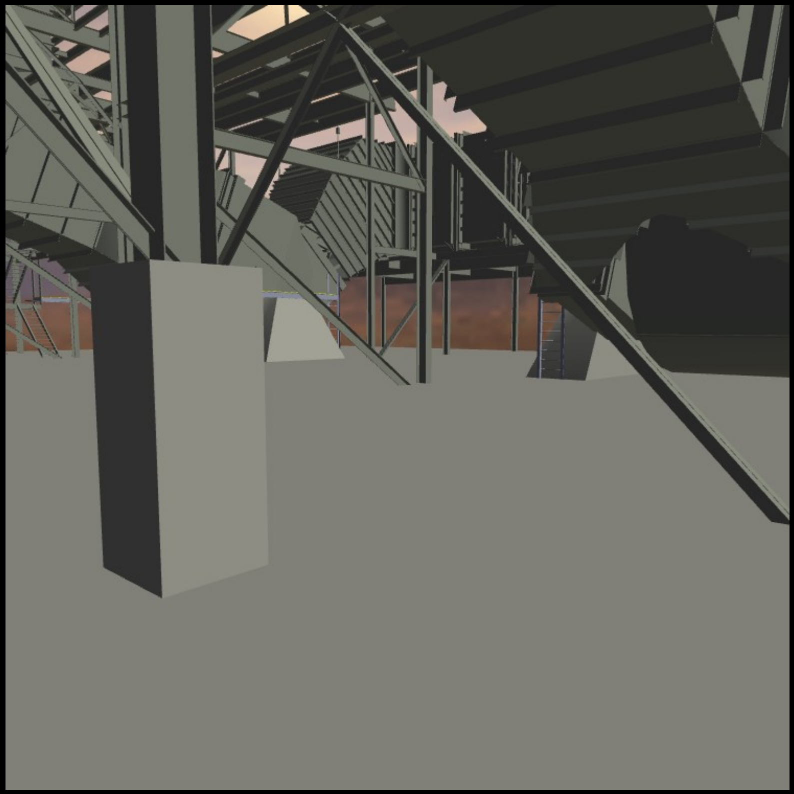
Importance of shadows



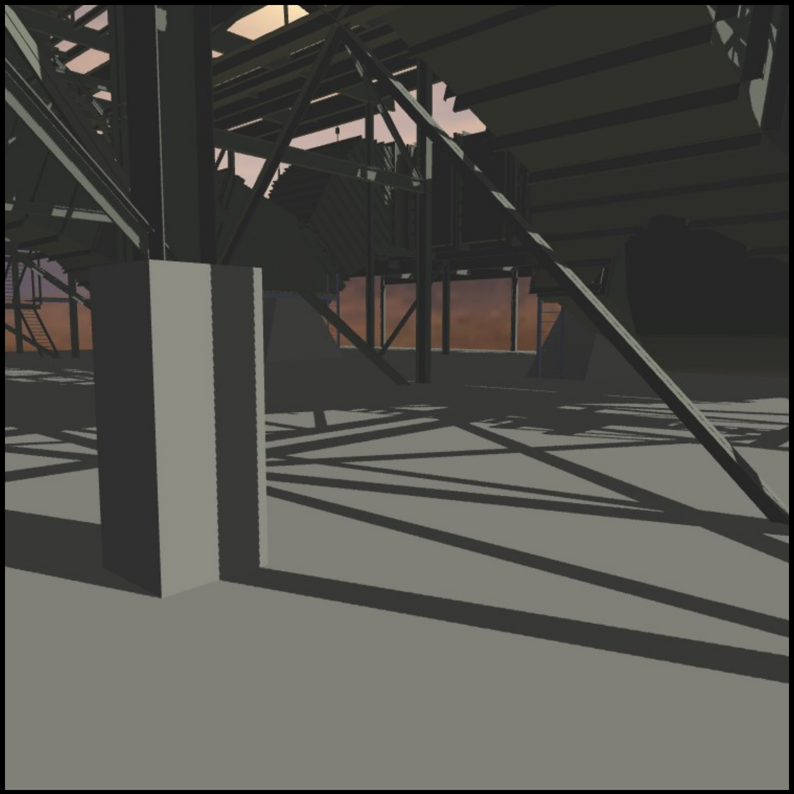
Importance of shadows



Importance of shadows



Without shadows



With shadows

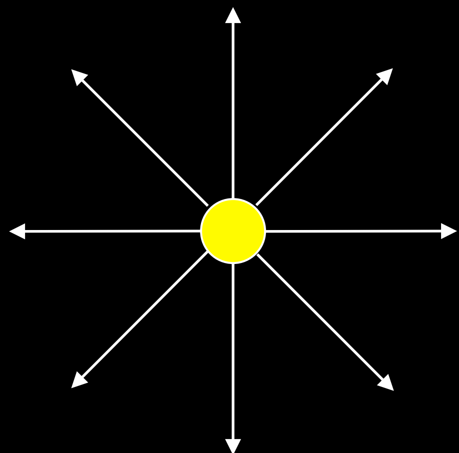
Source: UNC

Doom III

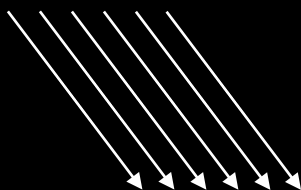
Source: Wikipedia

Reported to spend 50% of time rendering shadows!

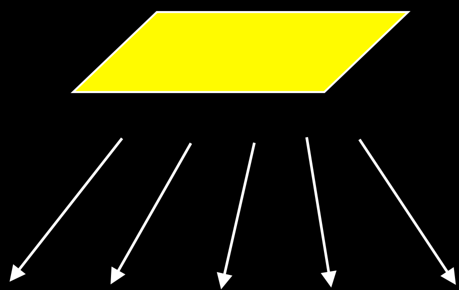
Light sources



point
light source

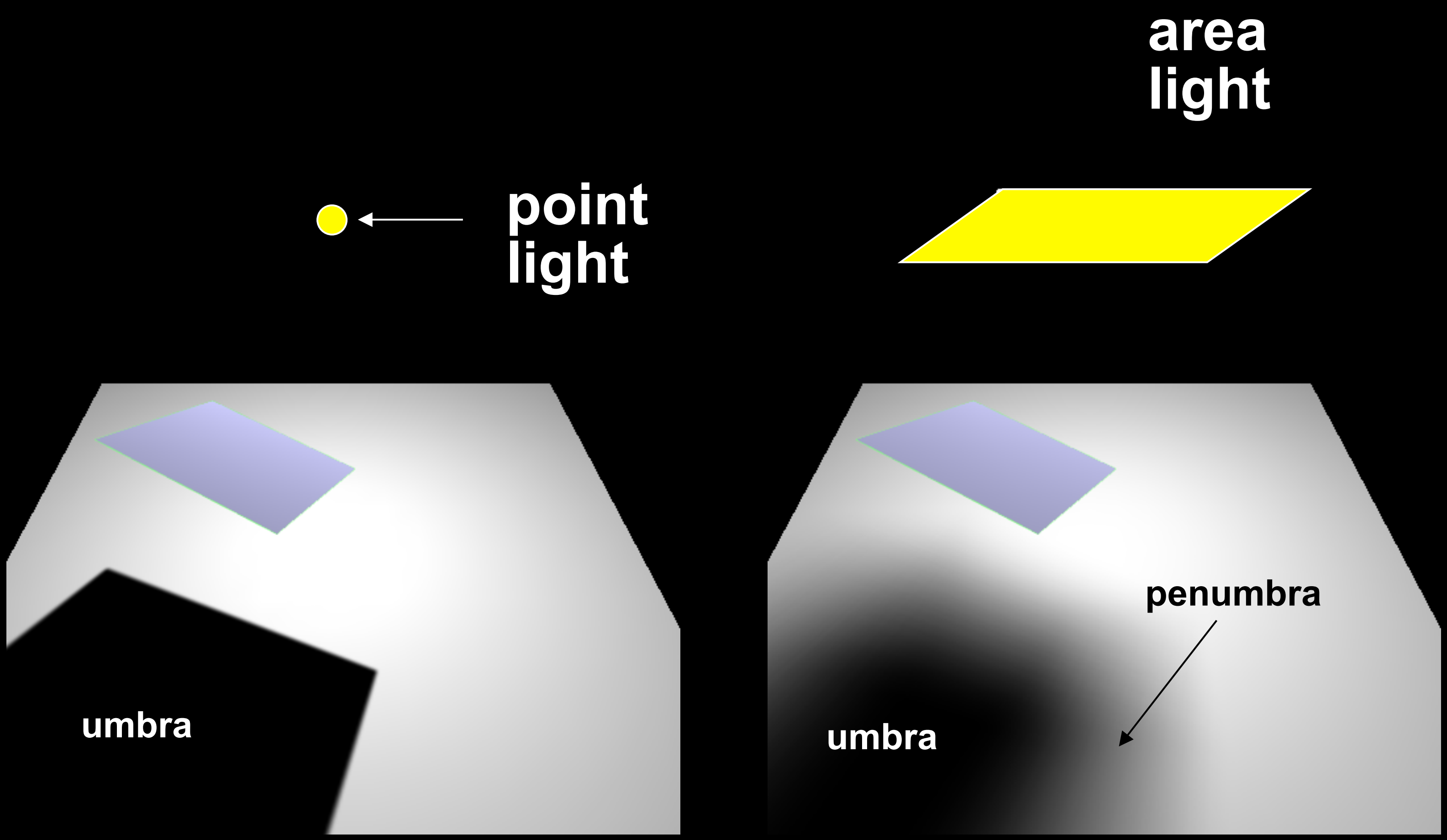


directional
light source



area
light source

Hard and soft shadows



Source: UNC

Hard shadow

Soft shadow

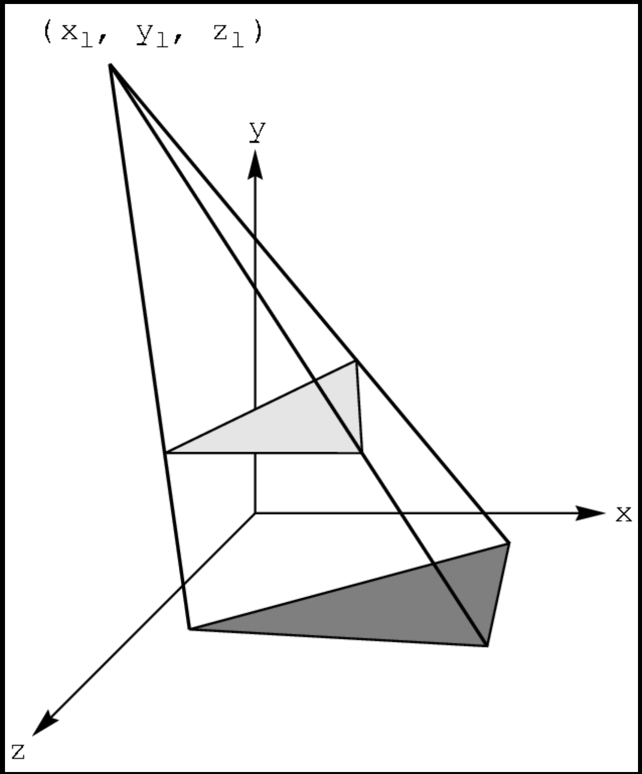
Shadow Algorithms

- 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"

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



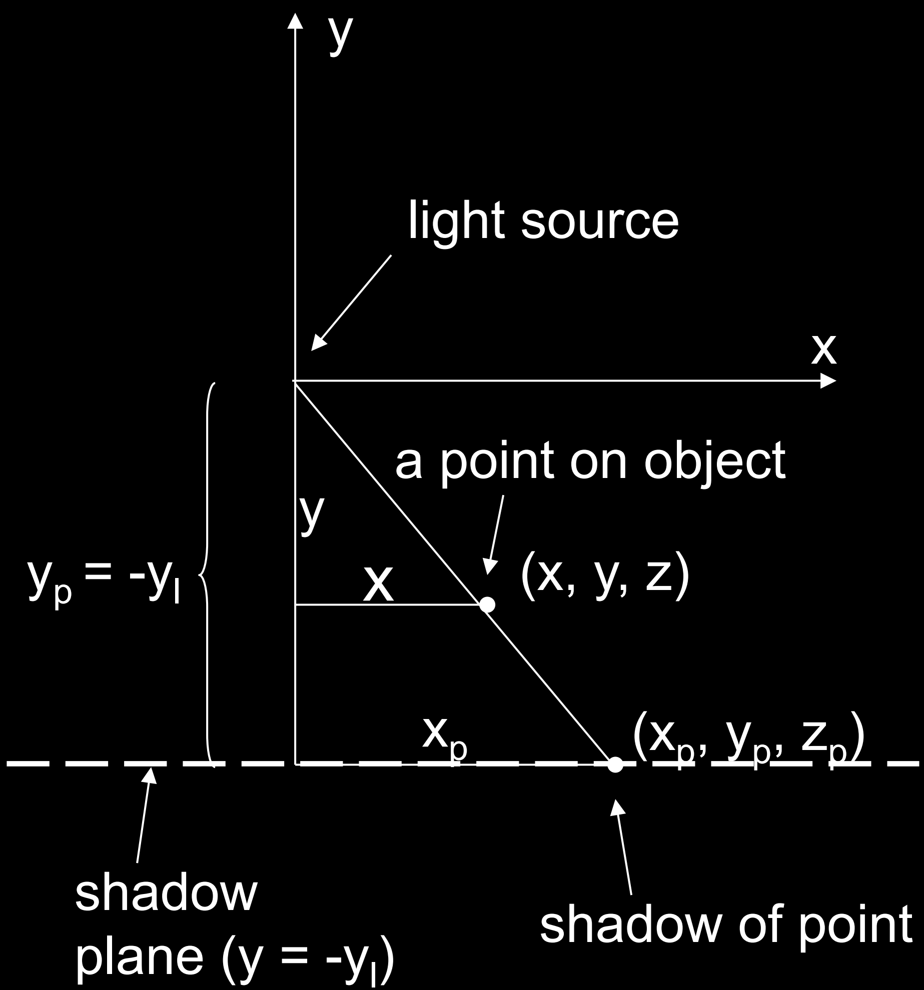
Shadow Projection Strategy

- 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



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

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

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

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

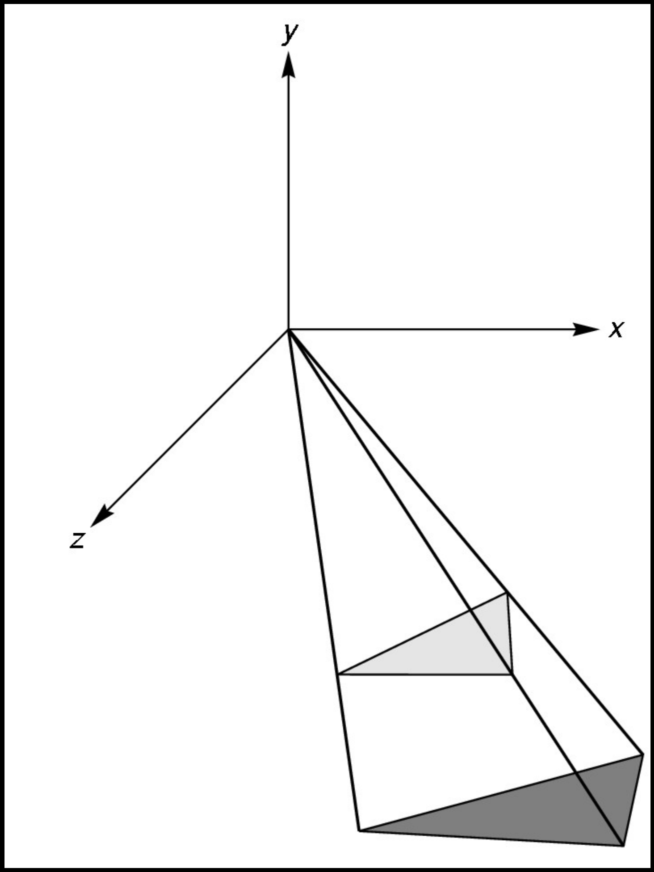
Light Source at Origin

- 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 State

- Assume xl, yl, zl hold light coordinates

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

glPushMatrix();    /* 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 ("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)

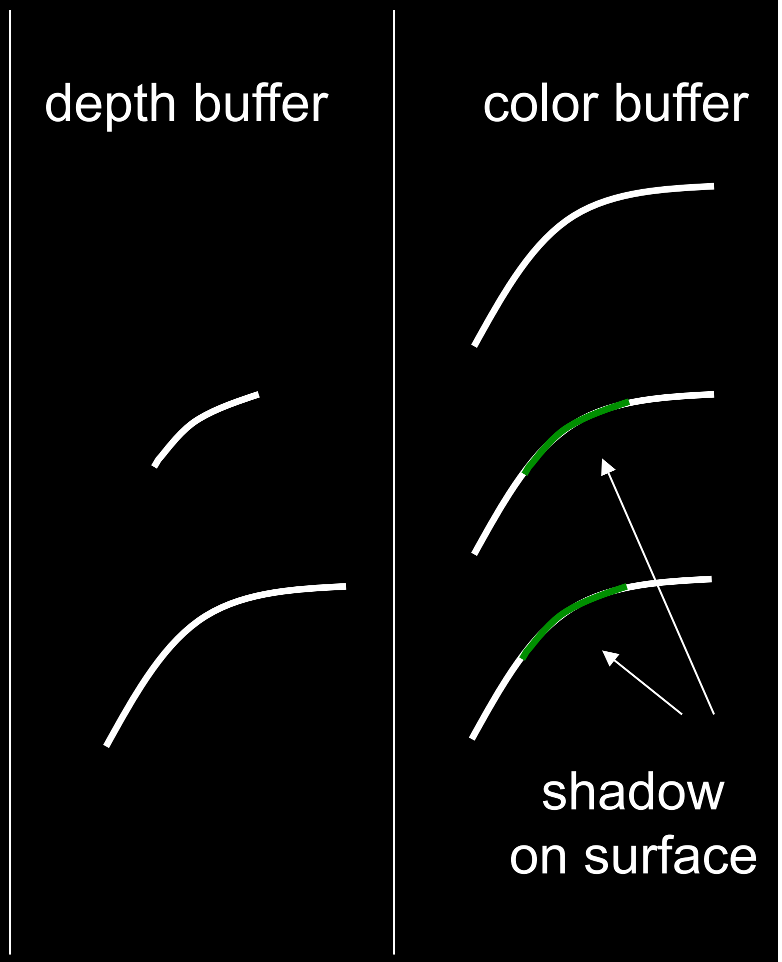
z-buffer fighting

no z-buffer fighting

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

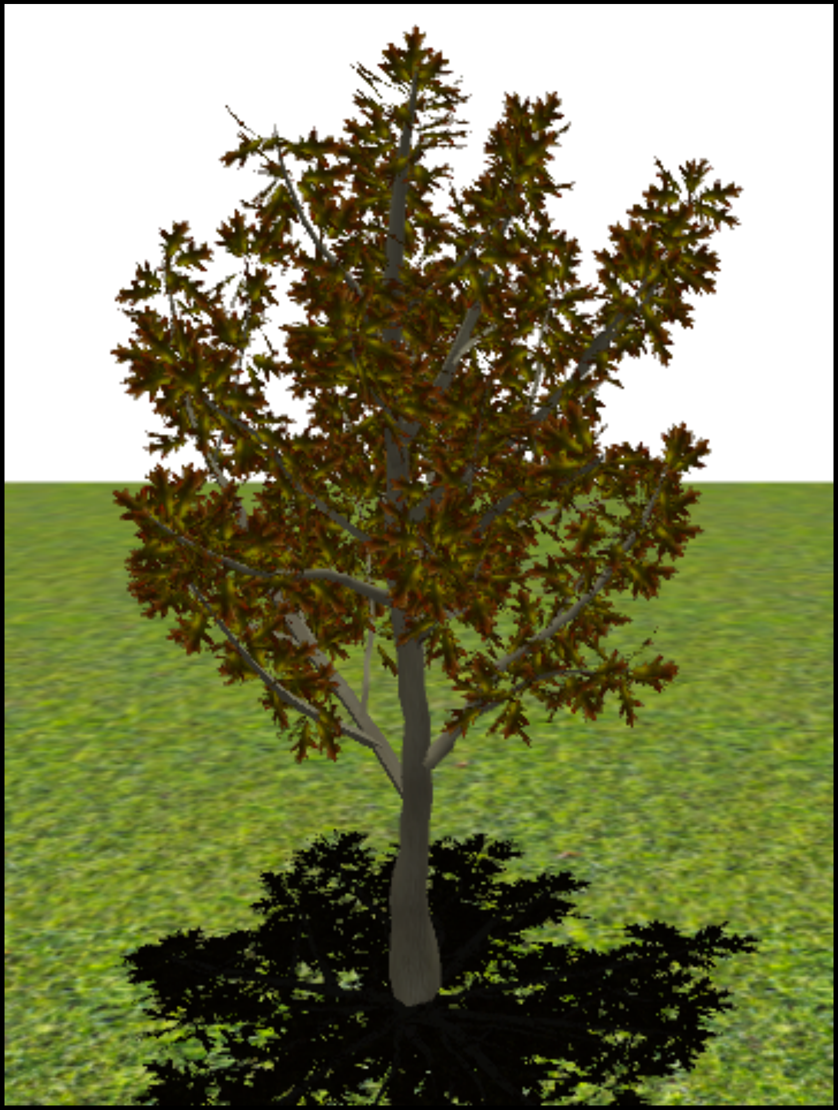


Outline

- Projections and Shadows
- Hierarchical Models

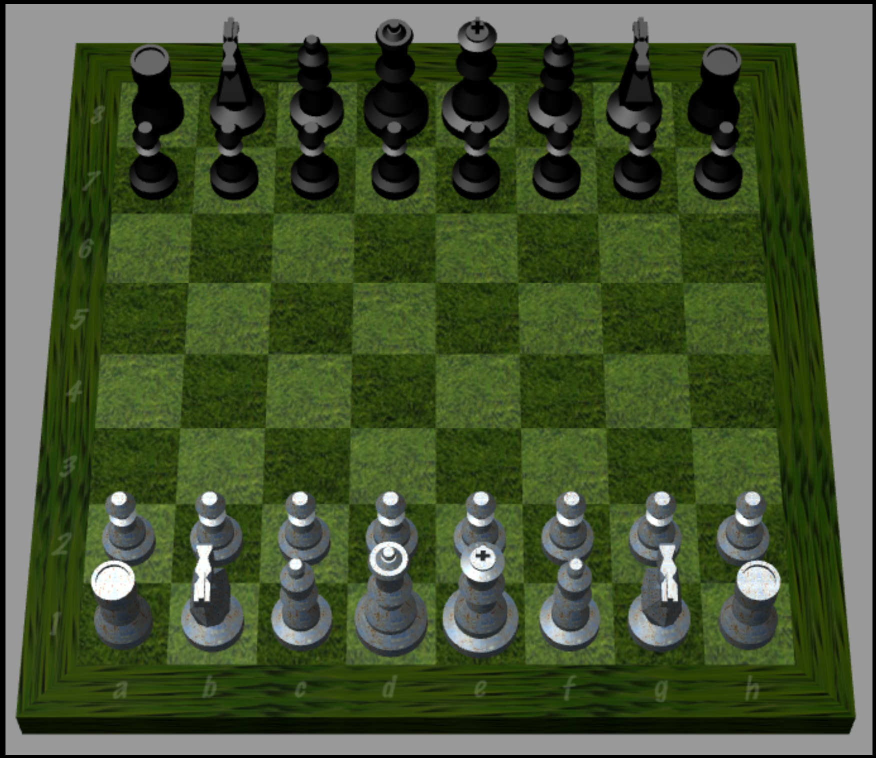
Hierarchical Models

- 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



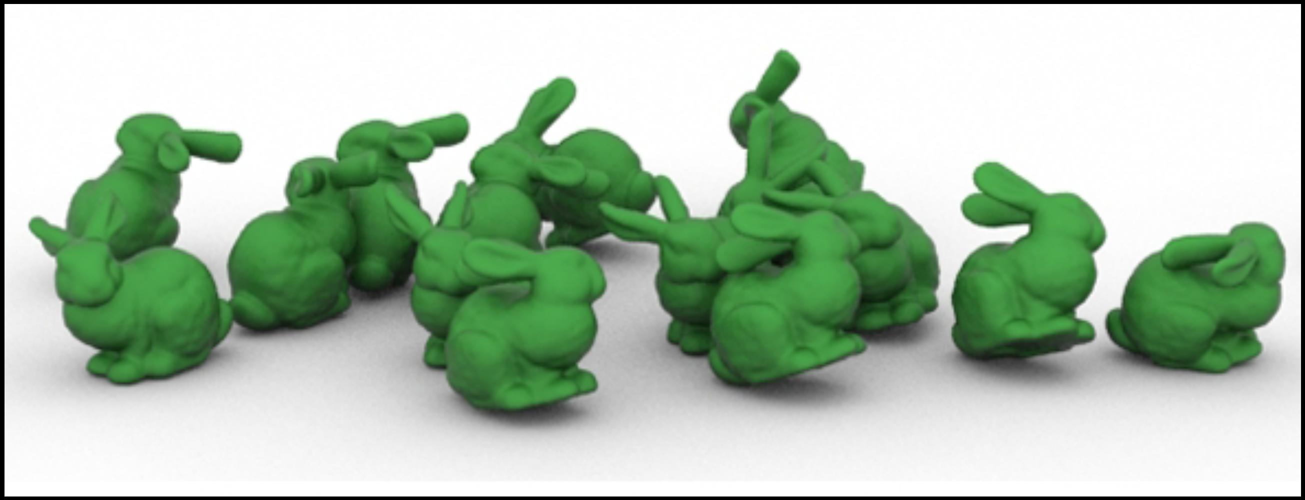
Instance Transformation

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



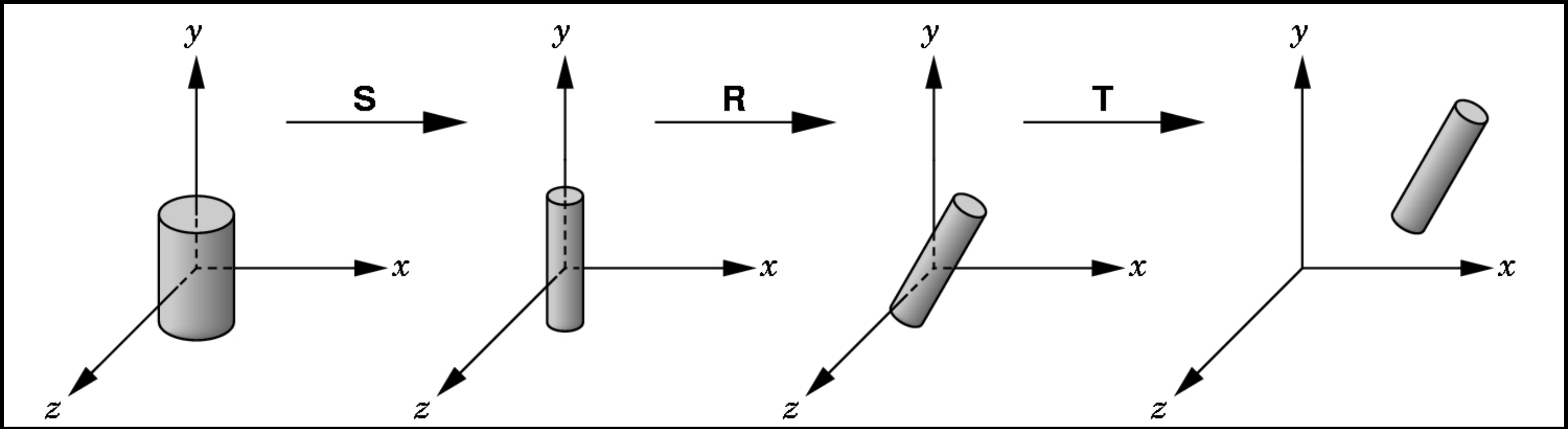
Instance Transformation

- 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

```
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:

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

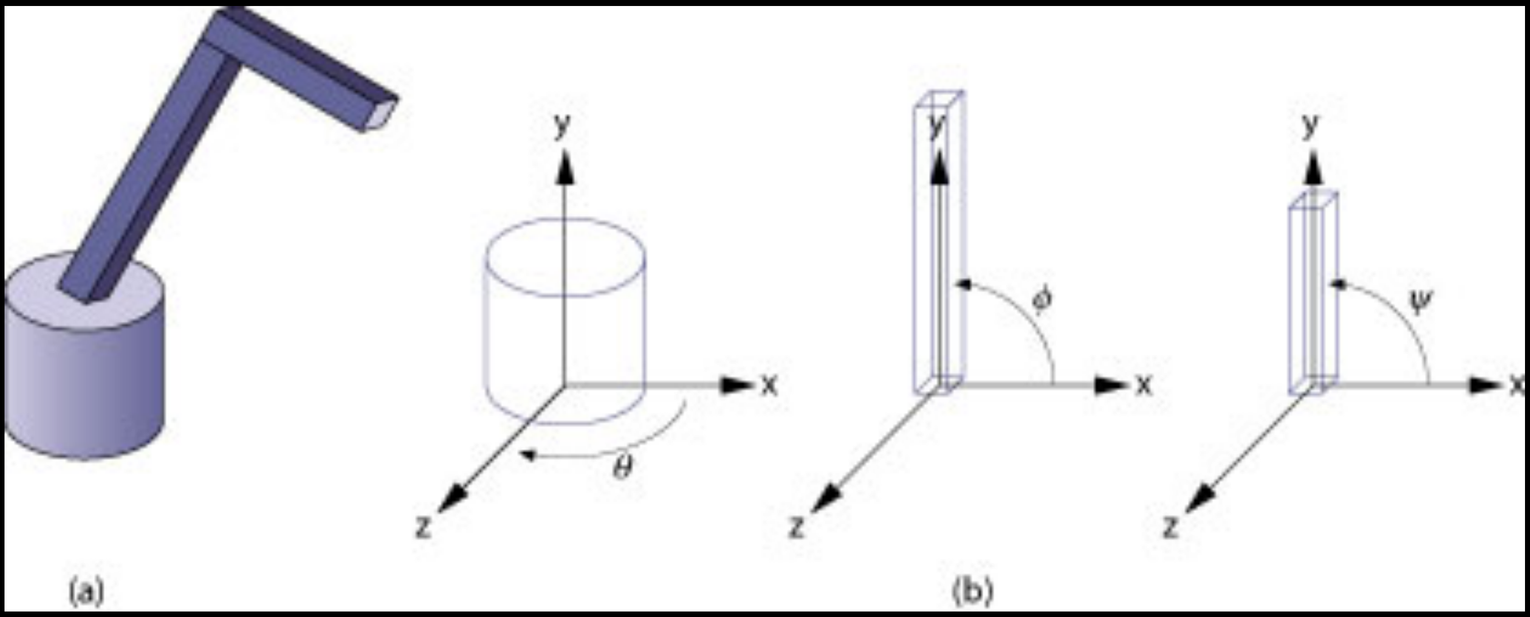
- Use: `glCallList(torus);`
- Can share both within each frame, and across different frames in time
- Can be hierarchical: a display list may call another

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

Drawing a Compound Object

- Example: simple "robot arm"

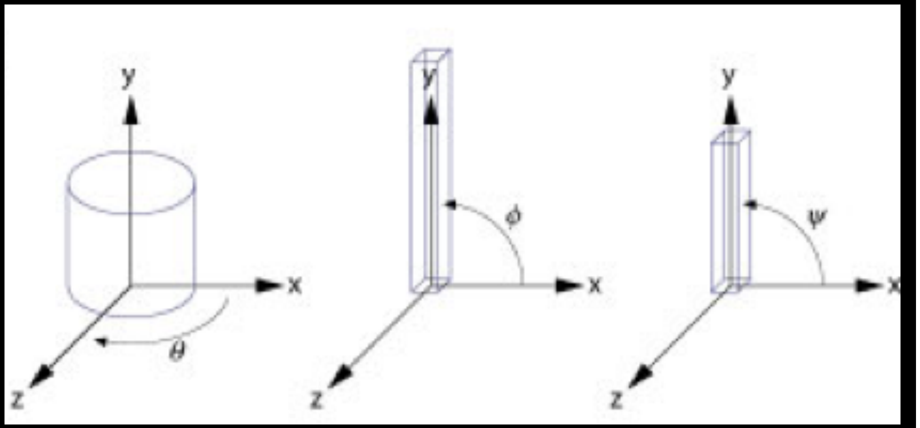


Base rotation θ, arm angle ϕ, joint angle ψ

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();
}
```



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

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

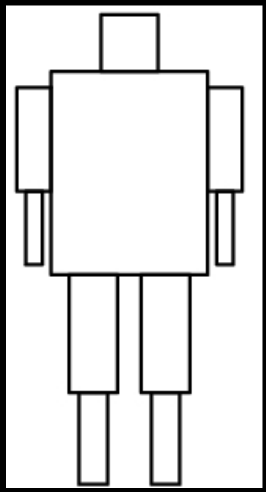
A Bug to Watch

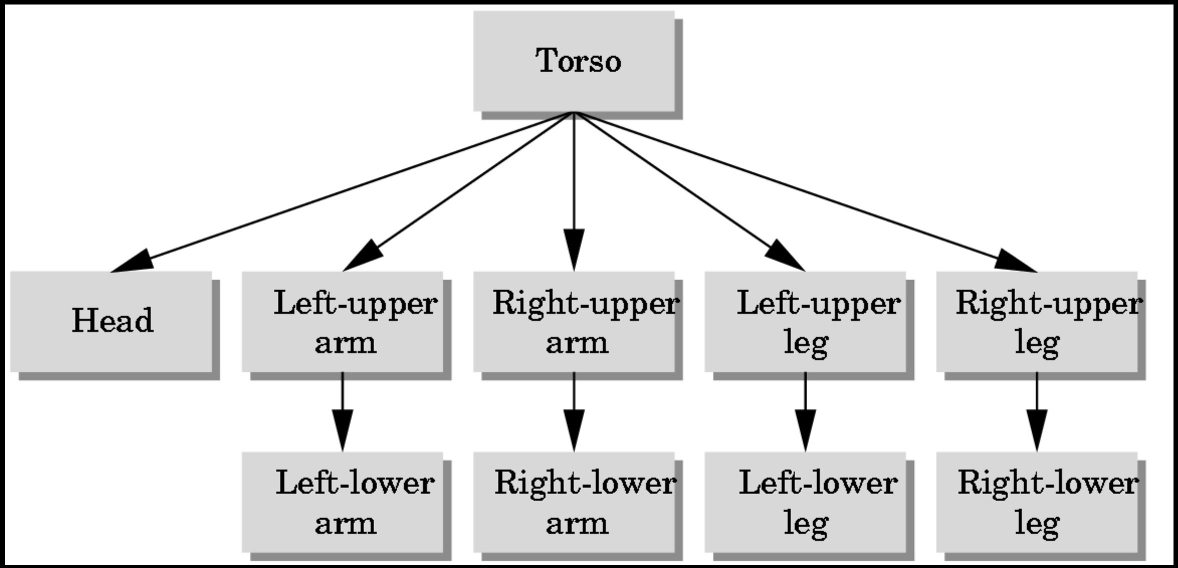
```
GLfloat theta = 0.0; ...;  /* update in idle callback */
GLfloat phi = 0.0; ...;  /* update in idle callback */
GLuint arm = glGenLists(1);
/* in init function */
glNewList(arm, GL_COMPILE);
   glRotatef(theta, 0.0, 1.0, 0.0);
   drawBase();
   ...
   drawUpperArm();
glEndList();
/* in display callback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

What is wrong?

More Complex Objects

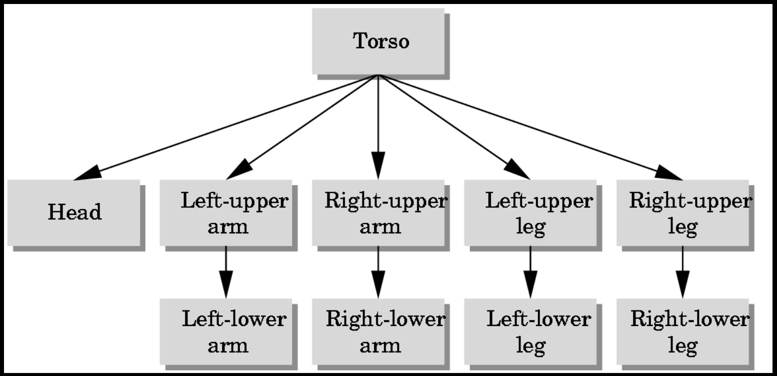
- Tree rather than linear structure
- Interleave along each branch
- Use push and pop to save state





Hierarchical Tree Traversal

- Order not necessarily fixed
- Example:




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

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

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;
```

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

```
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