CS380: Computer Graphics

Interacting with a 3D World

Sung-Eui Yoon

(윤성의)

Course URL:
http://sglab.kaist.ac.kr/~sungeui/CG/

Announcement

- **Mid-term exam**
  - **4:00pm ~ 5:40pm, Apr-18 (Tue.)**

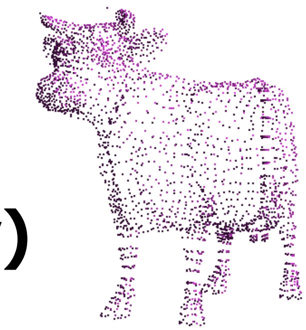
Class Objectives

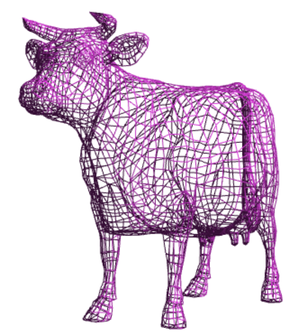
- **Read a mesh representation**
- **Understand a selection method and a virtual-trackball interface**
- **Understand the modeling hierarchy**

- **Related chapter: Chapter 5, Interaction**

Primitive 3D

- **How do we specify 3D objects?**
  - **Simple mathematical functions, z = f(x,y)**
  - **Parametric functions, (x(u,v), y(u,v), z(u,v)**
  - **Implicit functions, f(x,y,z) = 0**





- **Build up from simple primitives**
  - **Point – nothing really to see**
  - **Lines – nearly see through**
  - **Planes – a surface**

Simple Planes

- **Surfaces modeled as connected planar facets**
  - **N (>3) vertices, each with 3 coordinates**
  - **Minimally a triangle**

Specifying a Face

- **Face or facet**

  Face [v0.x, v0.y, v0.z] [v1.x, v1.y, v1.z] … [vN.x, vN.y, vN.z]

- **Sharing vertices via indirection**

  Vertex[0] = [v0.x, v0.y, v0.z]

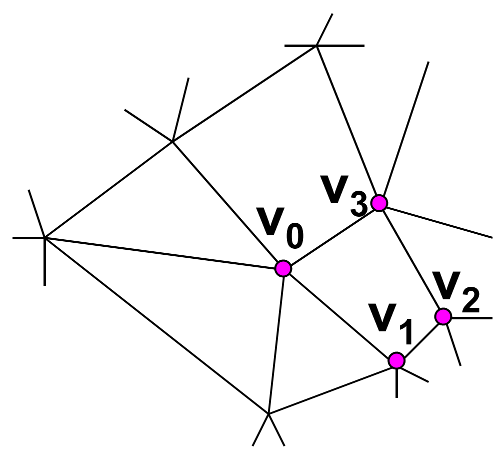
  Vertex[1] = [v1.x, v1.y, v1.z]

  Vertex[2] = [v2.x, v2.y, v2.z]

  :

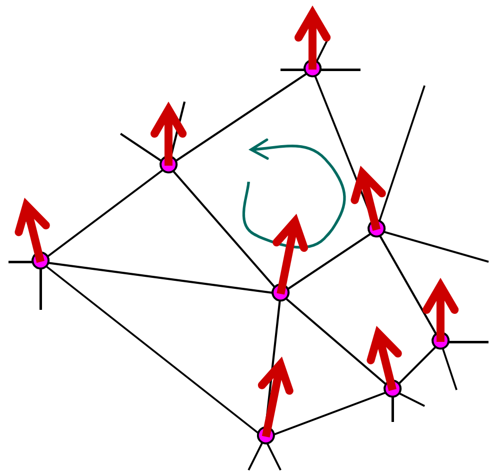
  Vertex[N] = [vN.x, vN.y, vN.z]

  Face v0, v1, v2, … vN



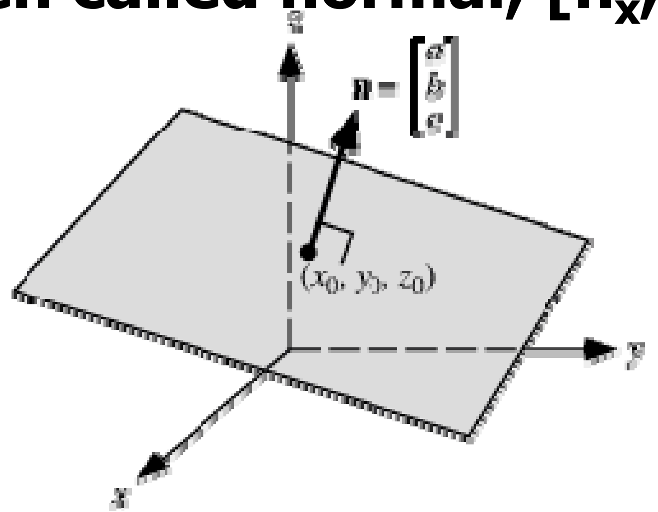
Vertex Specification

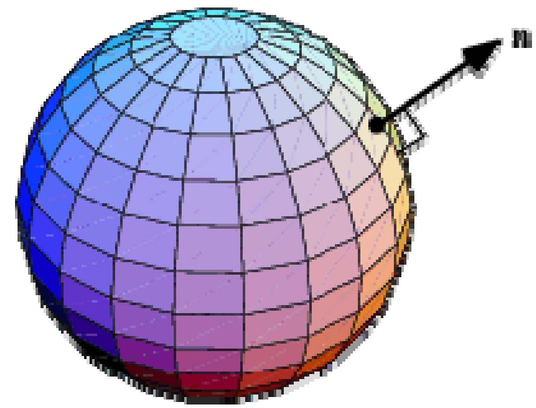
- **Where**
  - **Geometric coordinates [x, y, z]**
- **Attributes**
  - **Color values [r, g, b]**
  - **Texture Coordinates [u, v]**
- **Orientation**
  - **Inside vs. Outside**
  - **Encoded implicitly in ordering**
- **Geometry nearby**
  - **Often we'd like to "fake" a more complex shape than our true faceted (piecewise-planar) model**



Normal Vector

- **Often called normal, [n_x, n_y, n_z]**





- **Normal to a surface is a vector perpendicular to the surface**
  - **Will be used in illumination**
-
- **Normalized:** $\hat{n} = \frac{[n_x, n_y, n_z]}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$

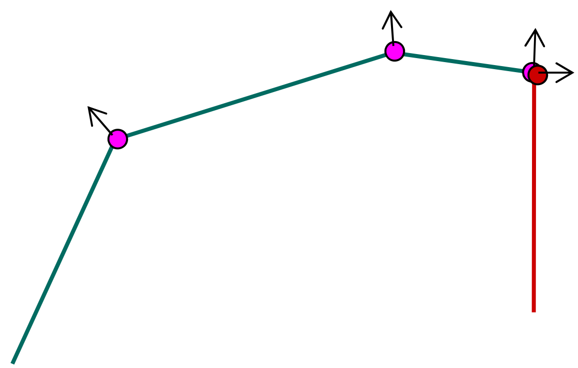
Drawing Faces in OpenGL

```
glBegin(GL_POLYGON);
foreach (Vertex v in Face) {
  glColor4d(v.red, v.green, v.blue, v.alpha);
  glNormal3d(v.norm.x, v.norm.y, v.norm.z);
  glTexCoord2d(v.texture.u, v.texture.v);
  glVertex3d(v.x, v.y, v.z);
}
glEnd();
```

- **Heavy-weight model**
  - Attributes specified for every vertex
- **Redundant**
  - Vertex positions often shared by at least 3 faces
  - Vertex attributes are often face attributes (e.g. face normal)

Decoupling Vertex and Face Attributes via Indirection

- Works for many cases
  - Used with vertex array or vertex buffer objects in OpenGL
- Exceptions:
  - Regions where the surface changes materials
  - Regions of high curvature (a crease)



3D File Formats

- MAX – Studio Max
- DXF – AutoCAD
  - Supports 2-D and 3-D; binary
- 3DS – 3D studio
  - Flexible; binary
- VRML – Virtual reality modeling language
  - ASCII – Human readable (and writeable)
- OBJ – Wavefront OBJ format
  - ASCII
  - Extremely simple
  - Widely supported

OBJ File Tokens

- **File tokens are listed below**

 # some text

Rest of line is a comment

v *float float float*

A single vertex's geometric position in space

vn *float float float*

A normal

vt *float float*

A texture coordinate

OBJ Face Varieties

f *int int int* ... **(vertex only)**

or

f *int/int int/int int/int* . . . **(vertex & texture)**

or

f *int/int/int int/int/int int/int/int* ... **(vertex, texture, & normal)**

- **The arguments are 1-based indices into the arrays**
  - **Vertex positions**
  - **Texture coordinates**
  - **Normals, respectively**

OBJ Example

- **Vertices followed by faces**
  - **Faces reference previous vertices by integer index**
  - **1-based**

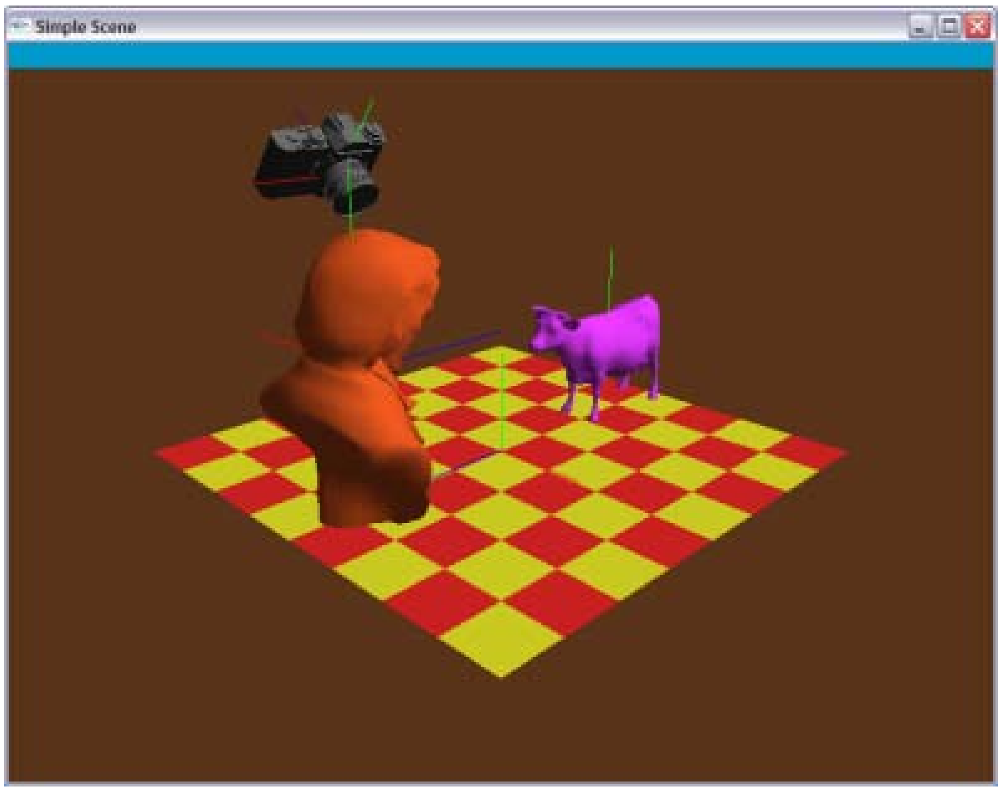
```
# A simple cube
v 1 1 1
v 1 1 -1
v 1 -1 1
v 1 -1 -1
v -1 1 1
v -1 1 -1
v -1 -1 1
v -1 -1 -1
f 1 3 4
f 5 6 8
f 1 2 6
f 3 7 8
f 1 5 7
f 2 4 8
```

OBJ Sources

- **Avalon – Viewpoint (http://avalon.viewpoint.com/) old standards**
- **3D Café – (http://www.3dcafe.com/asp/meshes.asp) Nice thumbnail index**
- **Others**
- **Most modeling programs will export .OBJ files**
- **Most rendering packages will read in .OBJ files**

Picking and Selection

- **Basic idea: Identify objects selected by the user**
  - **Click-selection: select one object at a time**
  - **Sweep-selection: select objects within a bounding rectangle**



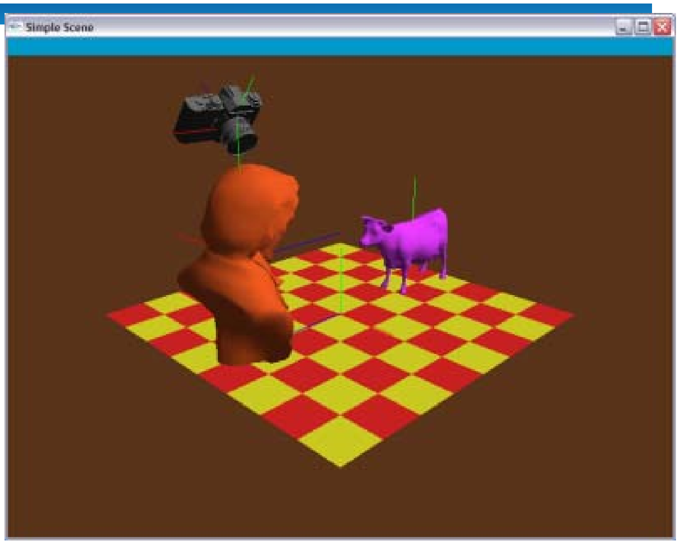
Demo
(click h)

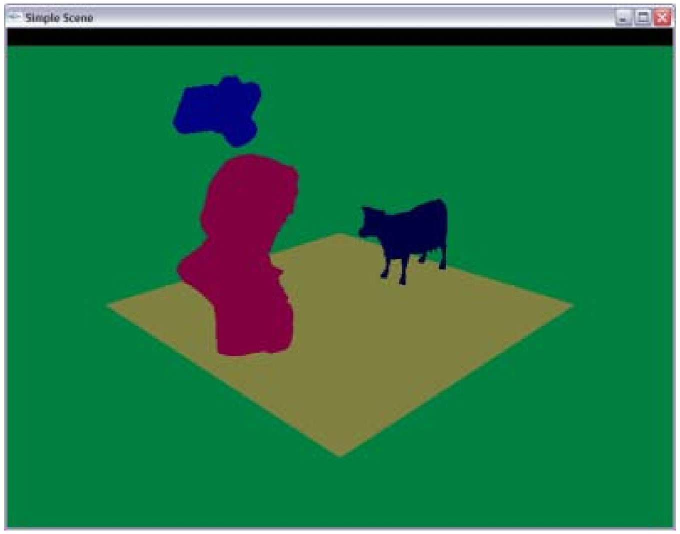
Picking and Selection

- **Several ways to implement selection:**
  - **Find screen space bounding boxes contained in pick region**
  - **Compute a pick ray and ray trace to find intersections**
  - **OpenGL selection buffers**
  - **Render to back buffer using colors that encode object IDs and return ID under pick point**

Selection with the Back Buffer

- Selects only objects that are visible
- Render objects to back buffer with color that encodes ID
- Use glReadPixels() to read the pixel at the pick point
- Back buffer is never seen





KAIST

An Example of Reading the Back Buffer

```
void onMouseButton(int button, int state, int x, int y)
{ ...
if (button == GLUT_LEFT_BUTTON  && state == GLUT_DOWN)
   {
     printf( "Left mouse click at (%d, %d)\n", x, y );
     selectMode = 1;
     display();
     glReadBuffer(GL_BACK);
     unsigned char pixel[3];
     glReadPixels(x, y, 1, 1, GL_RGB, GL_UNSIGNED_BYTE, pixel);
     printf( "pixel = %d\n", unmunge(pixel[0],pixel[1],pixel[2]));
     selectMode = 0;
    }
…
}
```

Buffer Operations in OpenGL

- **Still supported in OpenGL 4.3**

- **glReadBuffer (mode)**
  - **GL_FRONT, GL_BACK, etc.**

- **glReadPixels(x, y, w, h, pixel_format, data_type, * buffers)**
  - **Pixel_format: GL_RGB, GL_RGBA, GL_RED, etc.**
  - **Data_type: GL_UNSIGNED_BYTE, GL_FLOAT, etc.**

- **Other related APIs**
  - **glDrawPixels**

Interaction Paradigms

- **Can move objects or camera**
  - **Object moving is most intuitive if the object “sticks” to the mouse while dragging**

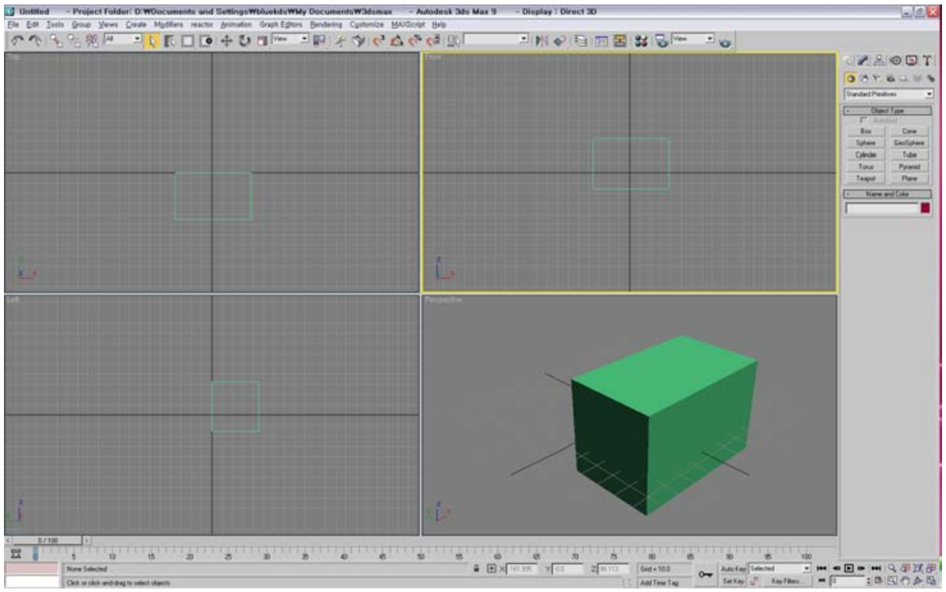
Interaction Paradigms

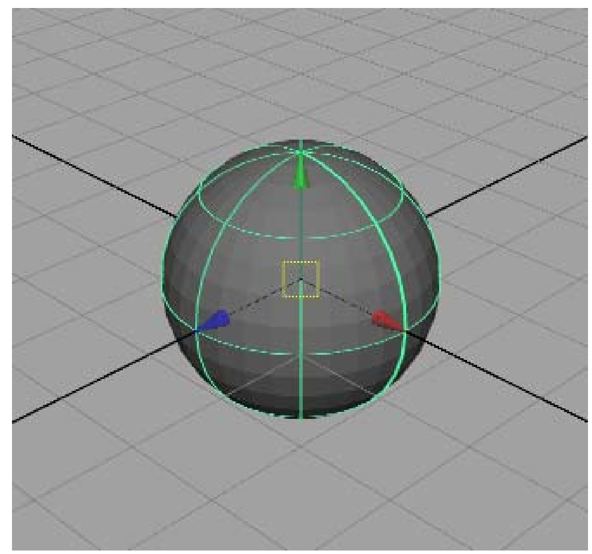
- Move w.r.t. to camera frame
  - Pan – move in plane perpendicular to view direction
  - Dolly – move along the view direction
  - Zoom - looks like dolly: objects get bigger, but position remains fixed
  - Rotate
    - up/down controls elevation angle
    - left/right controls azimuthal angle
  - Roll – spin about the view direction
  - Trackball – can combine rotate and roll

KAIST

Interaction Paradigms

- **Move w.r.t to modeling (or world) frame**



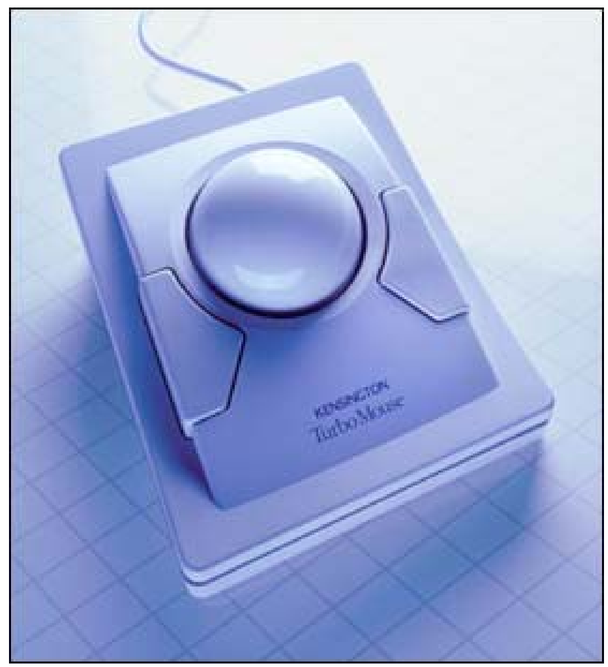


- **Maya combines both**
  - **Presents a frame where you can drag w.r.t the world axes**
  - **Dragging origin moves w.r.t. to camera frame**

Interaction - Trackball

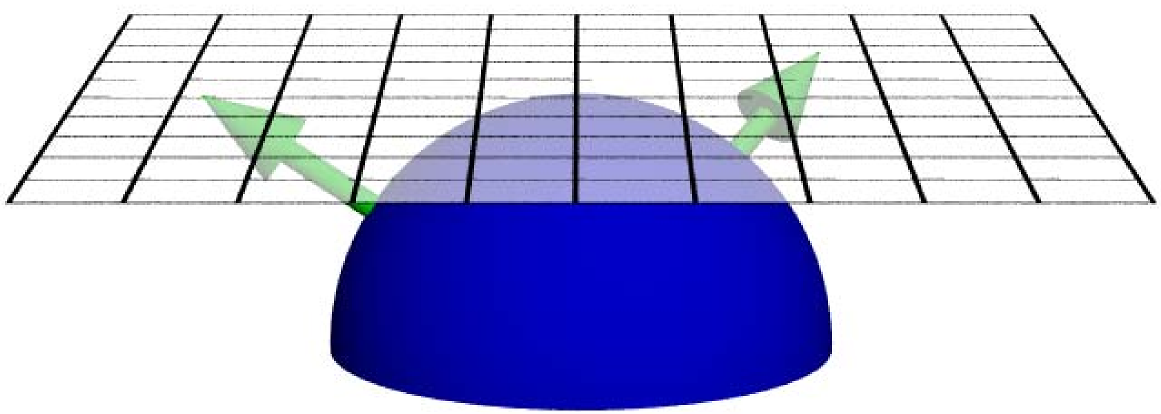
- A common UI for manipulating objects
- 2 degree of freedom device
- Differential behavior provides a intuitive rotation specification

Trackball demo



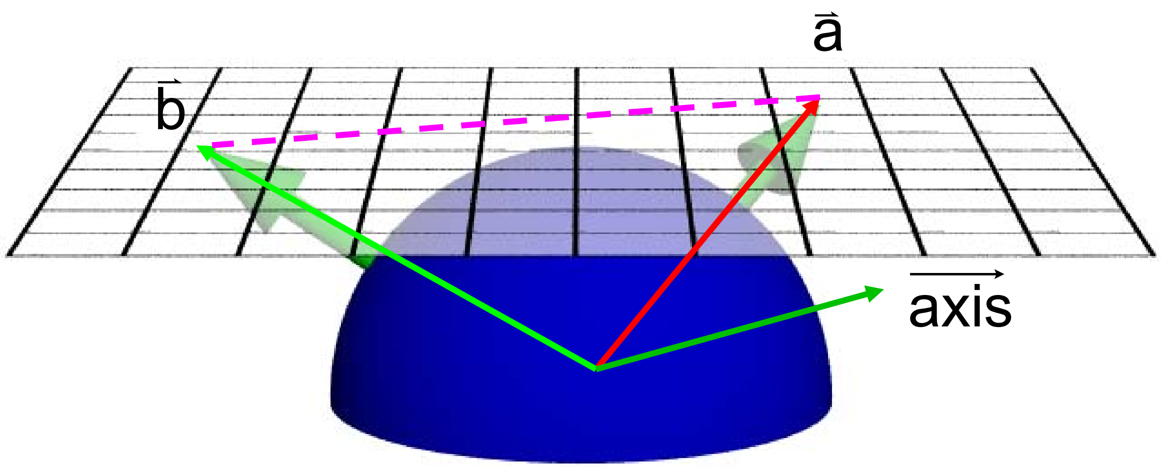
A Virtual Trackball

- Imagine the viewport as floating above, and just touching an actual trackball
- You receive the coordinates in screen space of the MouseDown() and MouseMove() events
- What is the axis of rotation?
- What is the angle of rotation?



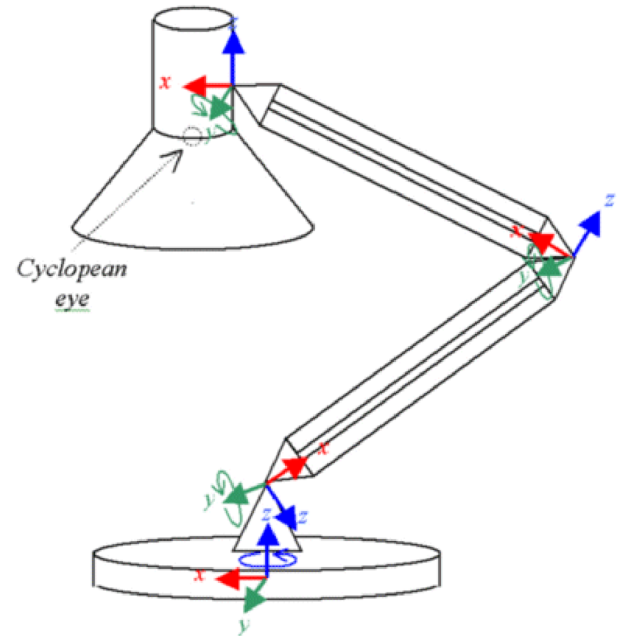
Computing the Rotation

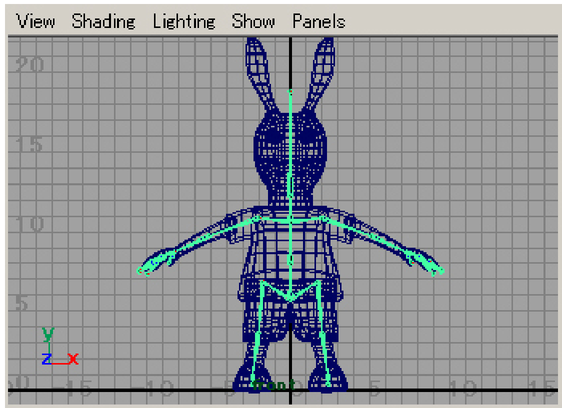
- **Construct a vector $\vec{a}$ from the center of rotation of the virtual trackball to the point of the MouseDown() event**
- **Construct a 2nd vector $\vec{b}$ from the center of rotation for a given MouseMove() event**
- **Normalize $\hat{a} = \frac{\vec{a}}{|\vec{a}|}$, and $\hat{b} = \frac{\vec{b}}{|\vec{b}|}$, and then compute $\overrightarrow{axis} = \hat{a} \times \hat{b}$**
- **Then find the $angle = \cos^{-1}(\hat{a} \cdot \hat{b})$ and construct $\mathbf{R} = Rotate(angle, \frac{\overrightarrow{axis}}{|\overrightarrow{axis}|})$**



Transformation Hierarchies

- Many models are composed of independent moving parts
- Each part defined in its own coordinate system
  - Compose transforms to position and orient the model parts
- A simple "One-chain" example



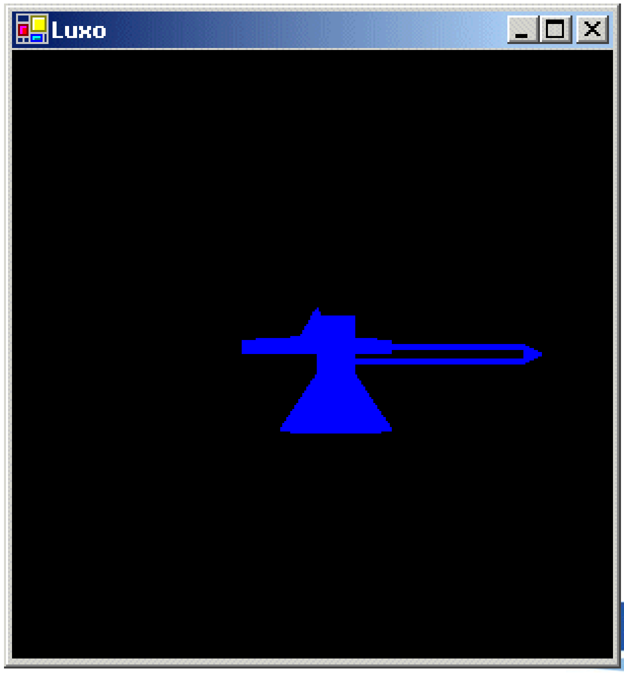


http://www.imanishi.com

Code Example (Take One)

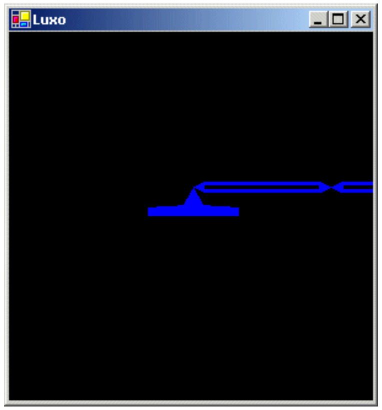
```
public void Draw() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookat(0, 0,-60,  0,0,0,   0,1,0); // world-to-camera transform

       glColor3d(0,0,1);
        glRotated(-90, 1, 0, 0);        // base-to-world transform
        Draw(Lamp.BASE);
        Draw(Lamp.BODY);
        Draw(Lamp.NECK);
        Draw(Lamp.HEAD);
        glFlush();
}
```



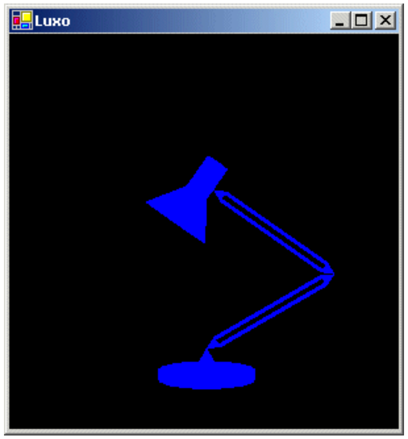
Code Example (Take Two)

```
public void Draw() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslated(0.0, 0.0, -60.0);     // world-to-view transform
    glColor3d(0,0,1);
    glRotated(-90, 1, 0, 0);           // base-to-world transform
      Draw(Lamp.BASE);
    glTranslated(0,0,2.5);             // body-to-base transform
      Draw(Lamp.BODY);
    glTranslated(12,0,0);              // neck-to-body transform
      Draw(Lamp.NECK);
    glTranslated(12,0,0);              // head-to-neck transform
      Draw(Lamp.HEAD);
    glFlush();
}
```



Code Example (Take Three)

```
public void Draw() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslated(0.0, -12.0, -60.0);   // world-to-view transform
    glColor3d(0,0,1);
    glRotated(-90, 1, 0, 0);                    // base-to-world transform
      Draw(Lamp.BASE);
    glTranslated(0,0,2.5);         // body-to-base transform
    glRotated(-30, 0, 1, 0);       // rotate body at base pivot
      Draw(Lamp.BODY);
    glTranslated(12,0,0);  // neck-to-body transform
    glRotated(-115, 0, 1, 0);      // rotate neck at body pivot
      Draw(Lamp.NECK);
    glTranslated(12,0,0);  // head-to-neck transform
    glRotated(180, 1, 0, 0);// rotate head at neck pivot
      Draw(Lamp.HEAD);
    glFlush();
  }
```

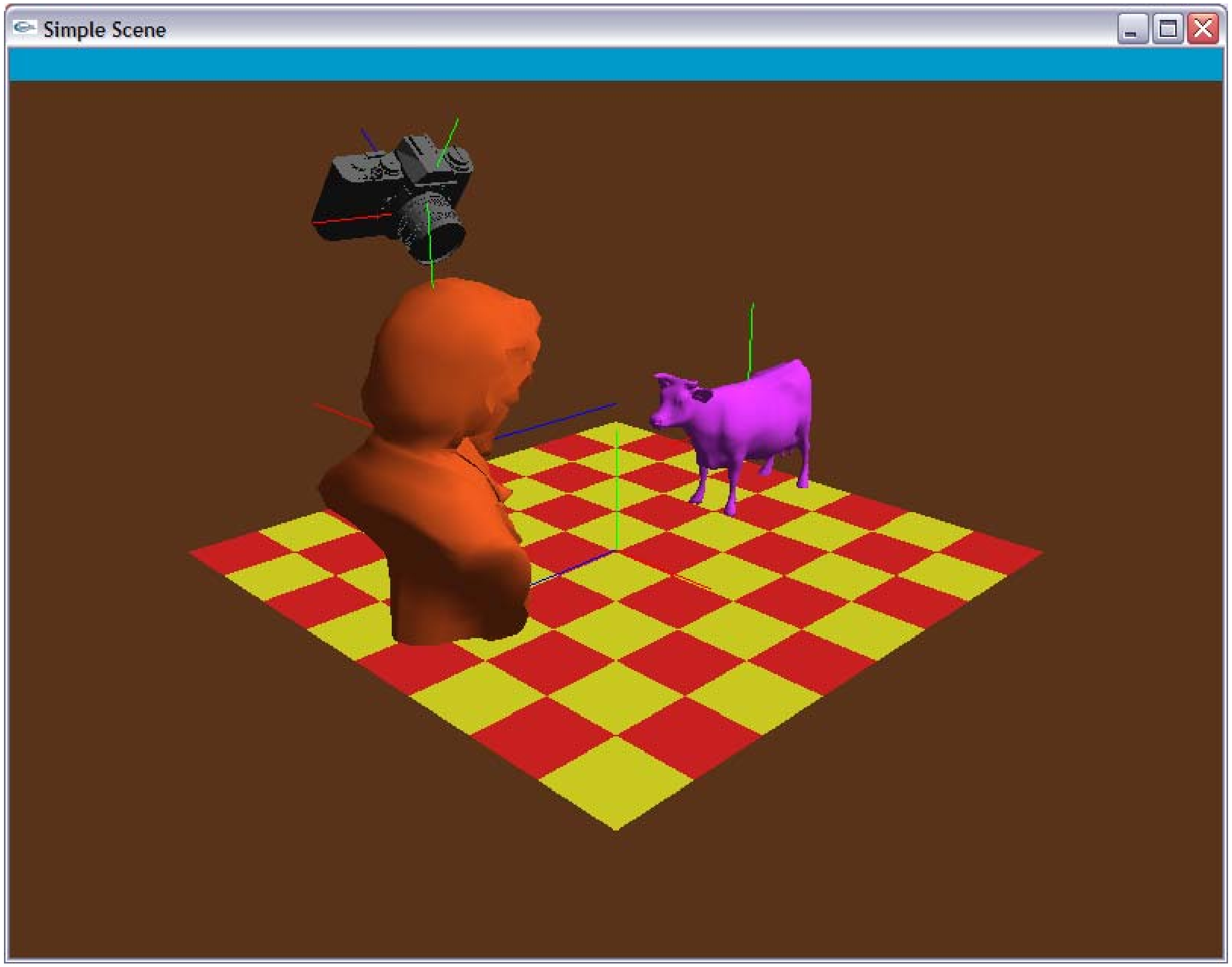


Class Objectives were:

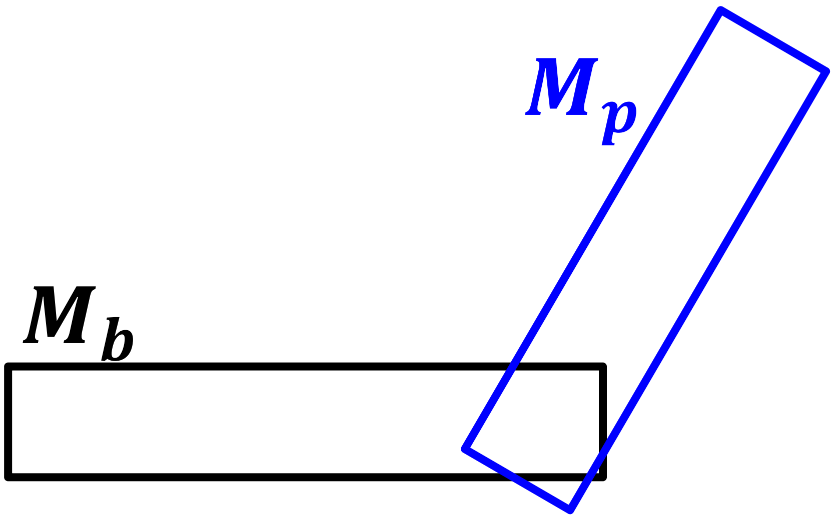
- **Read a mesh representation**
- **Understand a selection method and a virtual-trackball interface**
- **Understand the modeling hierarchy**

Program Assignment 4

- **Use the previous skeleton codes**



Figs



M_b : transform from the base to the world

M_p : transform from the part to the base