CS380: Computer Graphics
Interacting with a 3D World

Sung-Eui Yoon
(윤성의)

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

KAIST

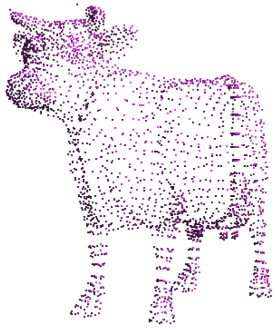
Class Objectives

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

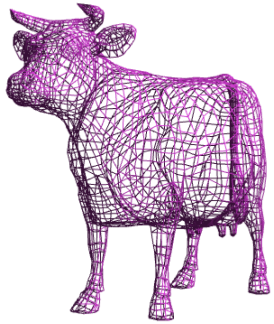
- **Related chapter: Chapter 5, Interaction**

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, $f(x,y,z) = 0$

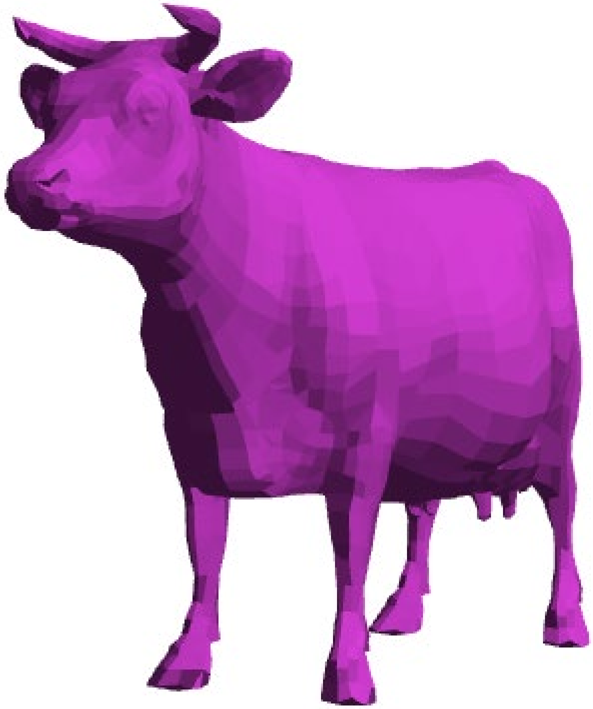


- Build up from simple primitives
  - Point – nothing really to see
  - Lines – nearly see through
  - Planes – a surface



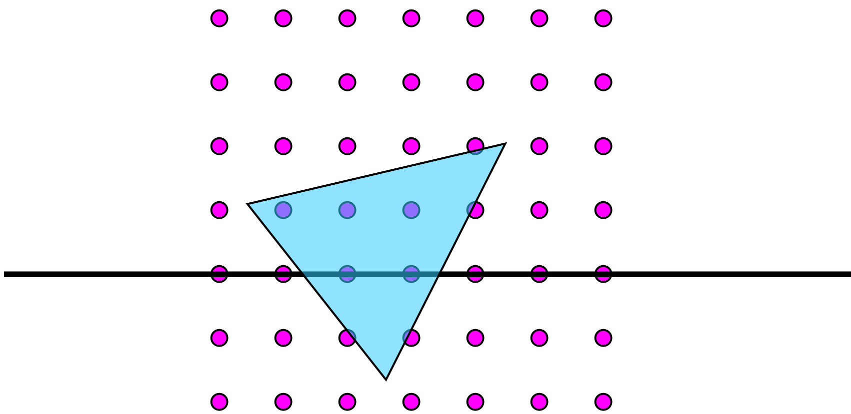
Simple Planes or Facets

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



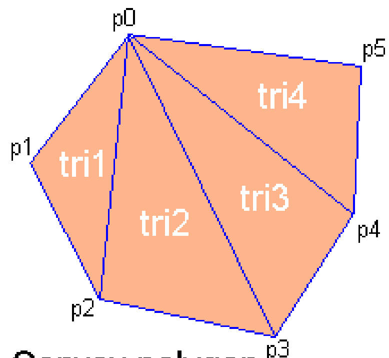
Why Triangles?

- Triangles are commonly used
  - Triangles are simple and convex
- Why is convexity important?
  - Simplify rasterization processes, which will be discussed later

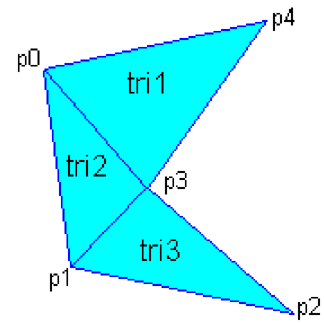


Why Triangles?

- **Arbitrary polygons can be decomposed into triangles**



Convex polygon

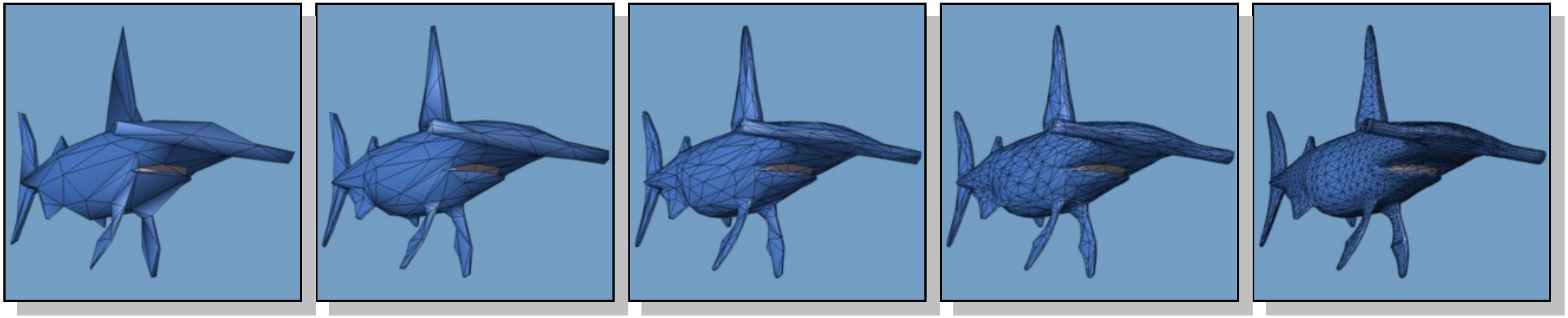


Non-convex

- **Decomposing a convex n-sided polygon is trivial**
  - **Suppose the polygon has ordered vertices $\{v_0, v_1, \dots v_n\}$**
  - **It can be decomposed into triangles $\{(v_0,v_1,v_2), \{v_0,v_2,v_3), (v_0,v_i,v_{i+1}), \dots (v_0,v_{n-1},v_n)\}$**
- **Decomposing a non-convex polygon is non-trivial**
  - **Sometimes have to introduce new vertices**

Why Triangles?

- **Triangles can approximate any 2-dimensional shape (or 3D surface)**
  - **Polygons are a locally linear (planar) approximation**
- **Improve the quality of fit by increasing the number edges or faces**



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]

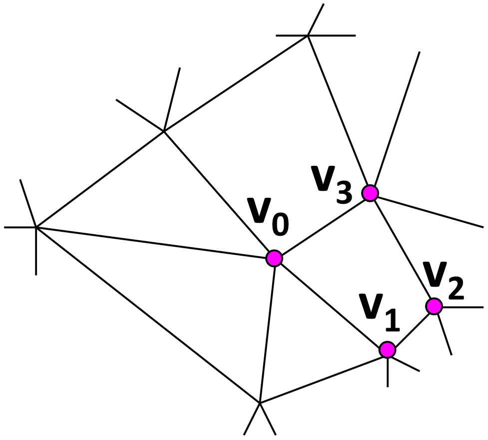
  Vertex[1] = [v1.x, v1.y, v1.z]

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

  :

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

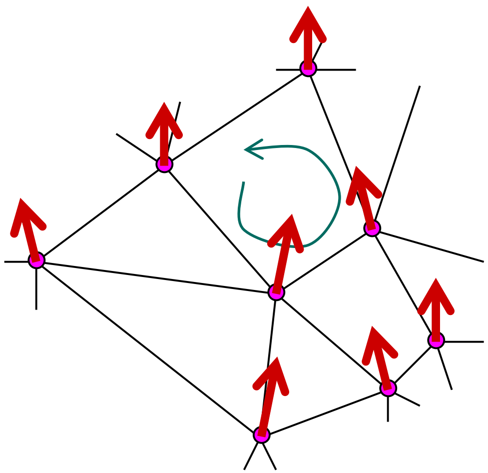
  Face v0, v1, v2, … vN



KAIST

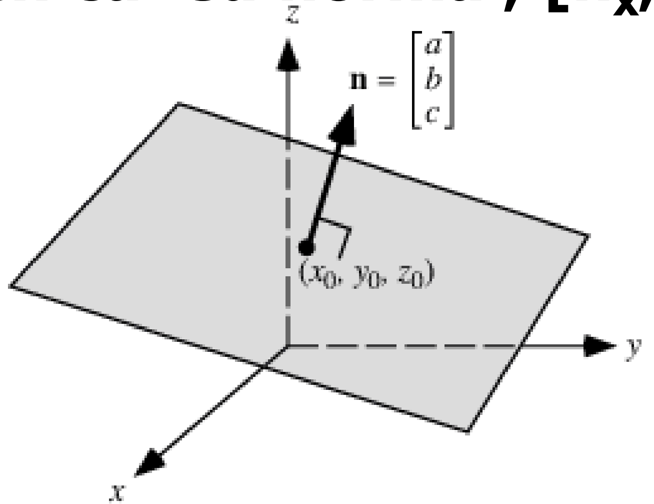
Vertex Specification

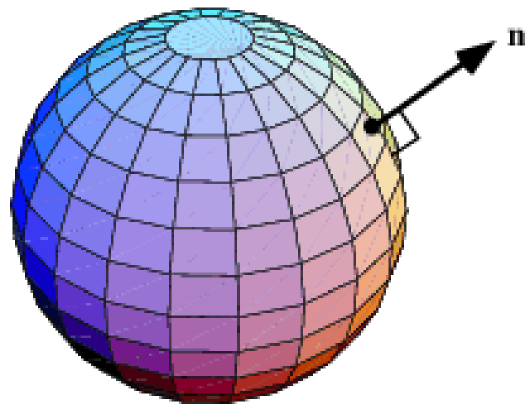
- **Where**
  - **Geometric coordinates [x, y, z]**
- **Attributes**
  - **Color values [r, g, b]**
  - **Texture Coordinates [u, v]**
- **Orientation**
  - **Inside vs. Outside**
  - **Encoded implicitly in ordering**



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

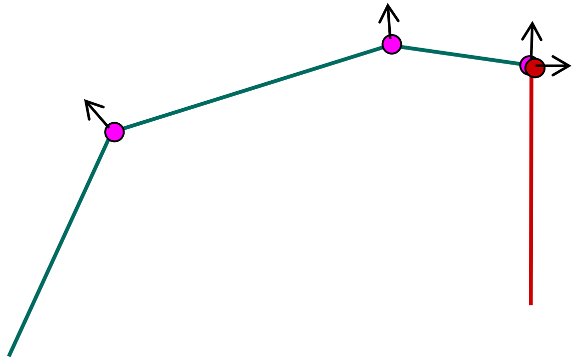
```
glBegin(GL_POLYGON);
foreach (Vertex v in Face) {
    glColor4d(v.red, v.green, v.blue, v.alpha);
    glNormal3d(v.norm.x, v.norm.y, v.norm.z);
    glTexCoord2d(v.texture.u, v.texture.v);
    glVertex3d(v.x, v.y, v.z);
}
glEnd();
```

- **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)**

KAIST

Decoupling Vertex and Face Attributes via Indirection

- **Use vertex index for defining faces**
- **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**

KAIST

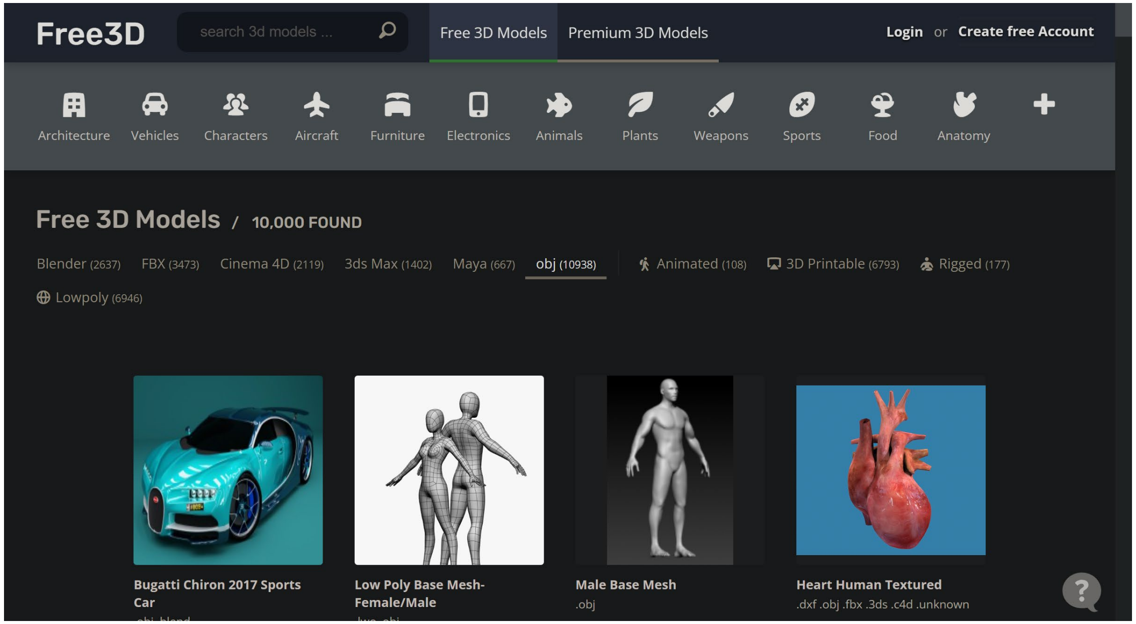
OBJ Example

- **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

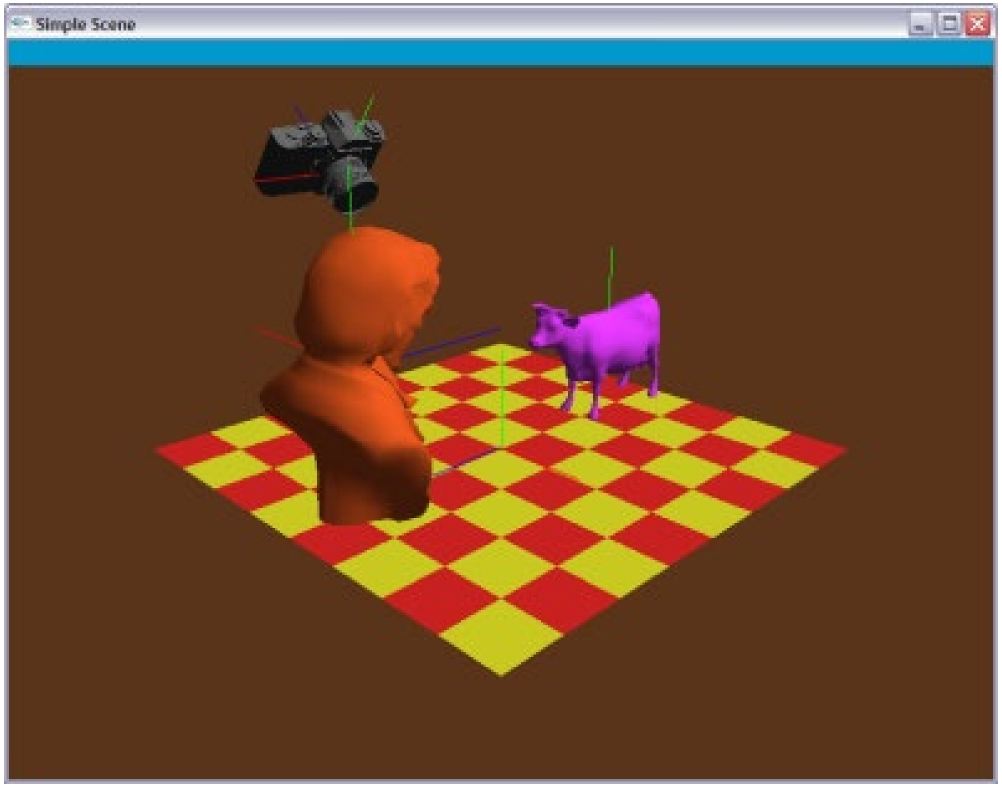
- Google "3d mesh obj"



- Most modeling programs export .OBJ files
- Most rendering packages 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)

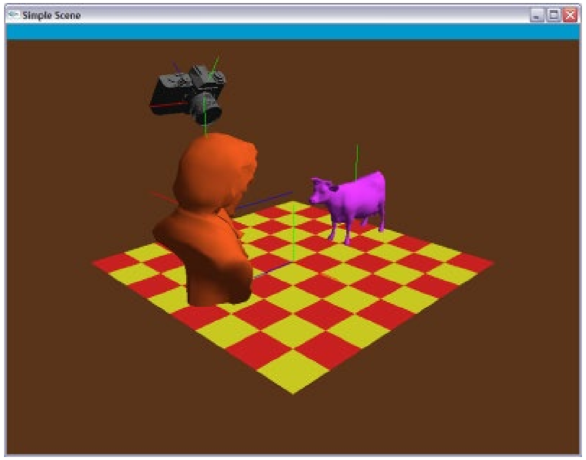
Picking and Selection

- **Several ways to implement selection:**
- **Object-based approaches**
  - **Find screen space bounding boxes contained in pick region**
  - **Compute a pick ray and ray trace to find intersections**
  - **Related to collision detection and ray tracing**
- **Image-space approaches**
  - **Render to back buffer using colors that encode object IDs and return ID under pick point**

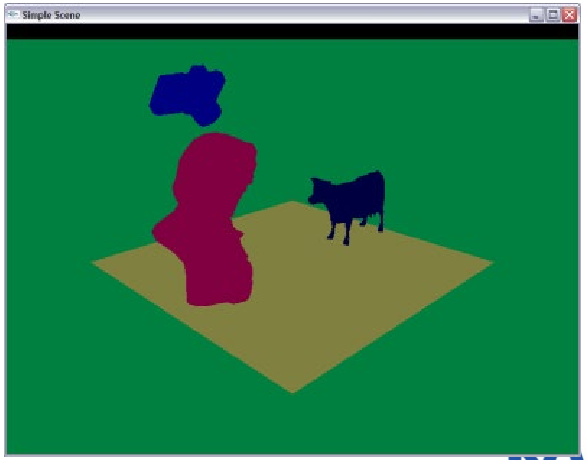
Selection with the Back Buffer

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

Front buffer



Back buffer



Interaction Paradigms

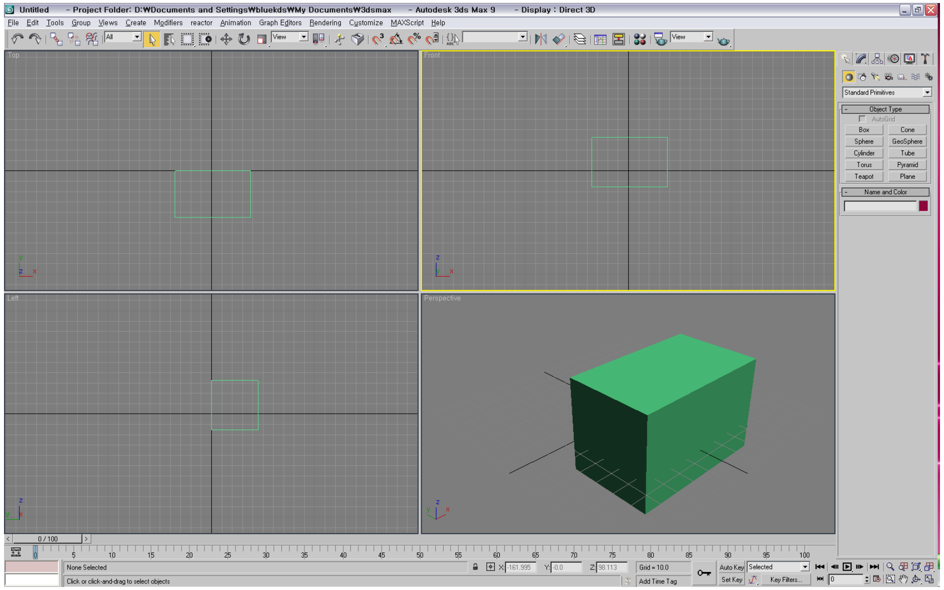
- **Can move objects or camera**
  - **Object moving is most intuitive if the object "sticks" to the mouse while dragging**

Interaction Paradigms

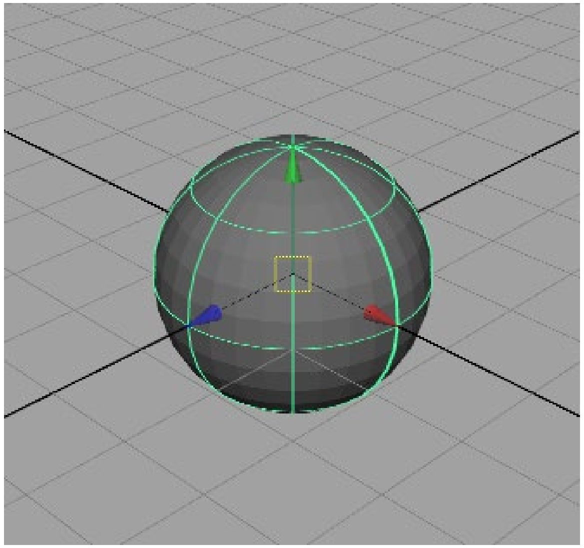
- **Move w.r.t. to camera frame**
  - **Pan – move in plane perpendicular to view direction**
  - **Dolly – move along the view direction**
  - **Zoom - looks like dolly: objects get bigger, but position remains fixed**
  - **Rotate**
    - **up/down controls elevation angle**
    - **left/right controls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

Interaction Paradigms

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

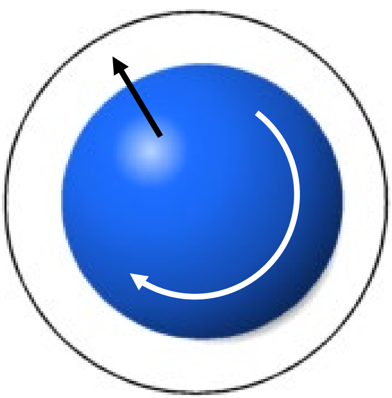


- **Combines both**
  - **Presents a frame where you can drag w.r.t the world axes**
  - **Dragging origin moves w.r.t. to camera frame**



Interaction - Trackball

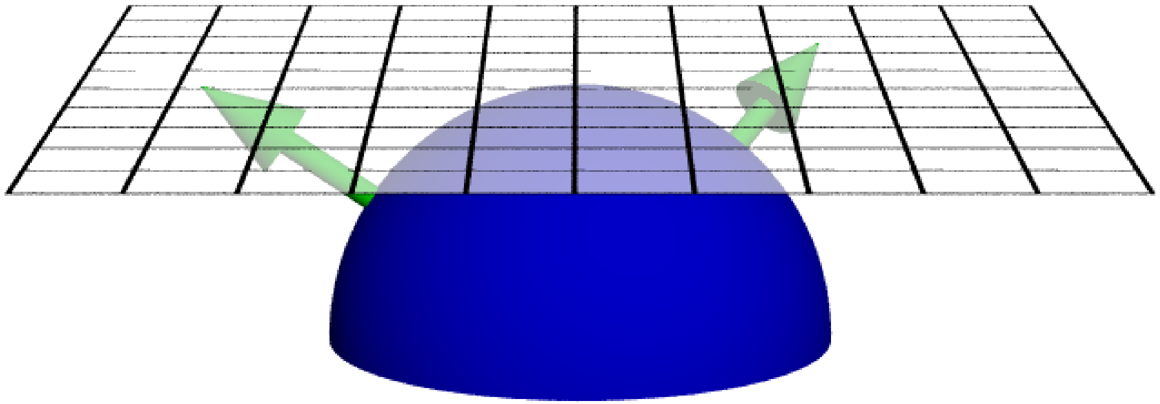
- A common UI for manipulating objects
- 2 degree of freedom device
- Differential behavior provides a intuitive rotation specification



Trackball demo

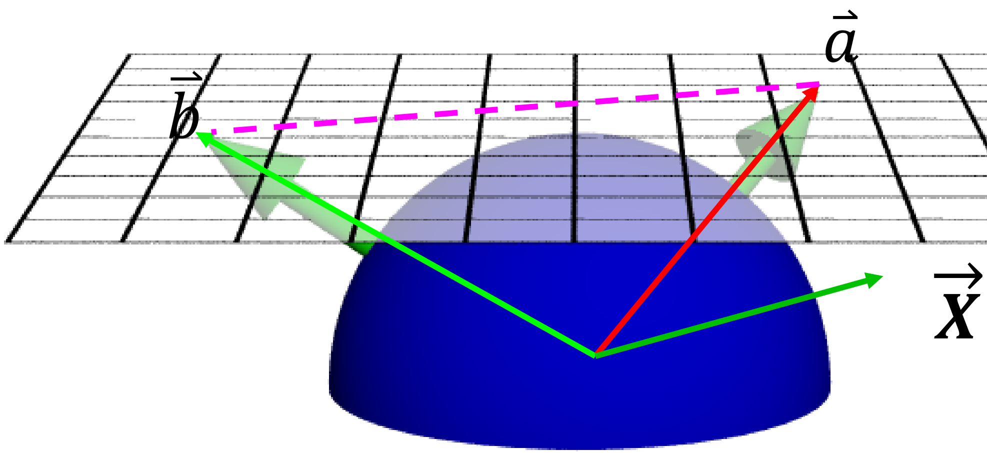
A Virtual Trackball

- **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** $\vec{X} = \hat{a} \times \hat{b}$
- **Then find $\theta = cos^{-1}(\hat{a} \cdot \hat{b})$ and construct** $\mathbf{R} = Rotate(\theta, \frac{\vec{X}}{|\vec{X}|})$



Class Objectives were:

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

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