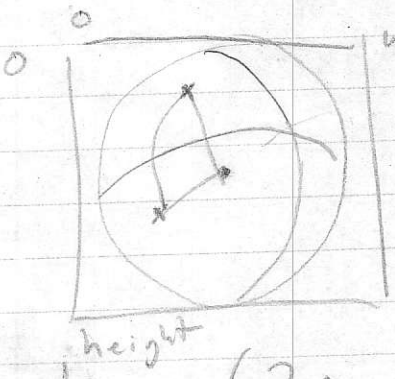
Virtual Trackball

- based on Fisher's code

Remember how it works ...



- Project mouse m_x, m_y onto unit hemisphere

$h_x = (2m_x - width) / width$

↳ take this from range 0 – width to range -1 – 1

$h_y = (2m_y - height) / height$ ✗

$h_y = (height - 2m_y) / height$ ✓

$h_2 = \sqrt{1 - h_x^2 - h_y^2}$ (equation of sphere)

∴ what if outside the circle?

→ don't do $\sqrt{-1}$

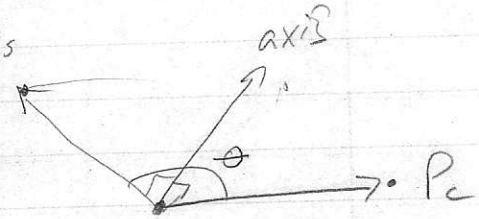
if $(h_x^2 + h_y^2 > 1)$

$h_z = \emptyset$

else

$h_z = \sqrt{1 - h_x^2 - h_y^2}$

So we now have p^s and p^c.
Now we rotate:



$$axis = p_i^s \times p^c$$

$$|\sin\theta| = |axis| \rightarrow \text{cross product property}$$

$$\theta = \arcsin(|axis|)$$

then because math.h uses degrees

$$\theta *= 180.0 / M\_PI$$

& normalize axis length_axis /= length(axis)

glRotatef(θ, axis.x, axis.y, axis.z)

Idea:
- record mouse pos when button pressed (p^s)
- when moved, find p^c, & angle, axis
- store this rotation matrix & replaces p^s with p^c
- when next mouse, find p^c, & angle, axis
- find matrix, multiply by old matrix and then store result.

→ set matrix to identity so we don't have special case first time.

```
float matrix[16] = { Identity };

glPushMatrix(); // so we don't mess current stuff up
glLoadIdentity();
glRotatef(angle, axis.x, axis.y, axis.z); //new rot
glMultMatrix(matrix);                     // old rot
glGetFv(GL_MODEL_VIEW_MATRIX, (GLfloat*) matrix);
glPopMatrix();
```

→ write mouse motion code.

Adding & Picking Points

① Lame way

reshape

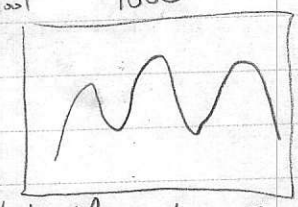
```
glOrtho(0, w, 0, h, -1, 1);
```

mouseButton → x, y

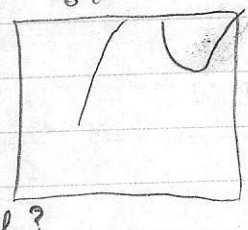
```
points.push_back(Point(x, height - y));
```

Why Lame?

→ School 1600 × 1200



Home 640 × 480



→ what if you've zoomed, shifted, or scaled?

② Better Way

reshape

```
glOrtho(-aspect, aspect, -1, 1, -1, 1);
```

mouseButton → x, y

→ could do manually $wx = \frac{2x - width}{width}$

→ short cut

```
GLint vp[4]
GLdouble proj[16], mv[16]
glGetIntegerv(GL_VIEWPORT, vp);
glGetDoublev(GL_MODELVIEW, mv);
glGetDoublev(GL_PROJECTION, proj);
GLfloat wx, wy, wz
gluUnProject(x, height - y, 0, mv, proj, vp, &wx, &wy, &wz);
```

→ use wx, wy (ignore wz)

Bonus → this works for perspect (glFrustum)

Picking [A] ① project "mouse point" ② find closest control point ③ do stuff

[B] ① project all control points ② find closest ③ if closest < 10 pixels, select & do stuff