Introduction to Computer Graphics with WebGL

Ed Angel

Professor Emeritus of Computer Science

Founding Director, Arts, Research, Technology and Science Laboratory

University of New Mexico

The Virtual Trackball

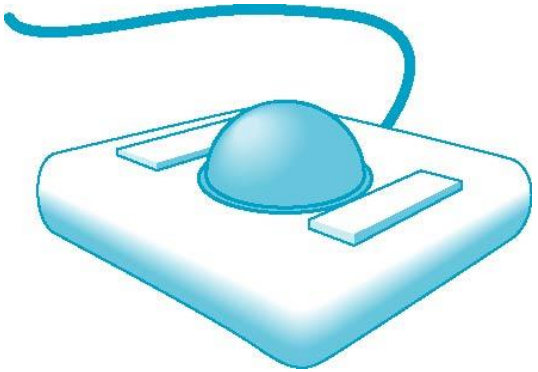
Ed Angel
Professor Emeritus of Computer Science
University of New Mexico

Objectives

- This is an optional lecture that
  - Introduces the use of graphical (virtual) devices that can be created using WebGL
  - Reinforce the benefit of not using direction angles and Euler angles
  - Makes use of transformations
  - Leads to reusable code that will be helpful later

Physical Trackball

- The trackball is an “upside down” mouse



- If there is little friction between the ball and the rollers, we can give the ball a push and it will keep rolling yielding continuous changes
- Two possible modes of operation
  - Continuous pushing or tracking hand motion
  - Spinning

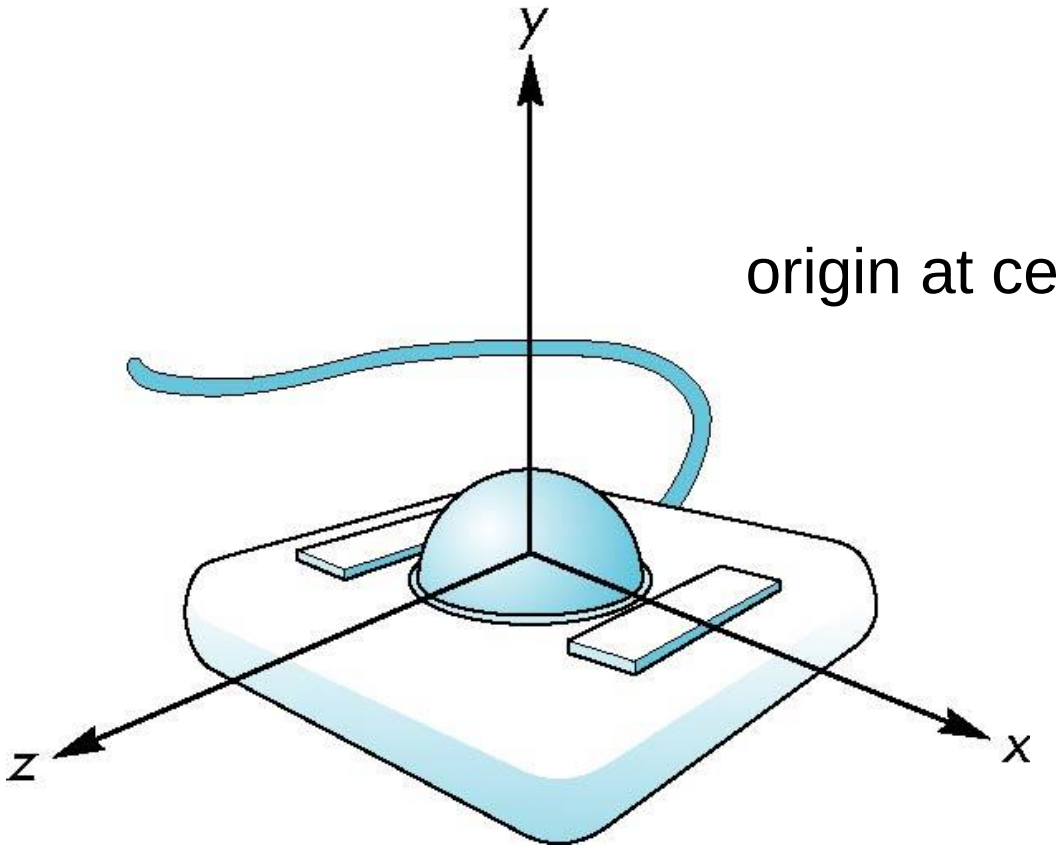
A Trackball from a Mouse

- Problem: we want to get the two behavior modes from a mouse
- We would also like the mouse to emulate a frictionless (ideal) trackball
- Solve in two steps
  - Map trackball position to mouse position
  - Use event listeners to handle the proper modes

Using Quaternions

- Quaternion arithmetic works well for representing rotations around the origin
- Can use directly avoiding rotation matrices in the virtual trackball
- Code was made available long ago (pre shader) by SGI
- Quaternion shaders are simple

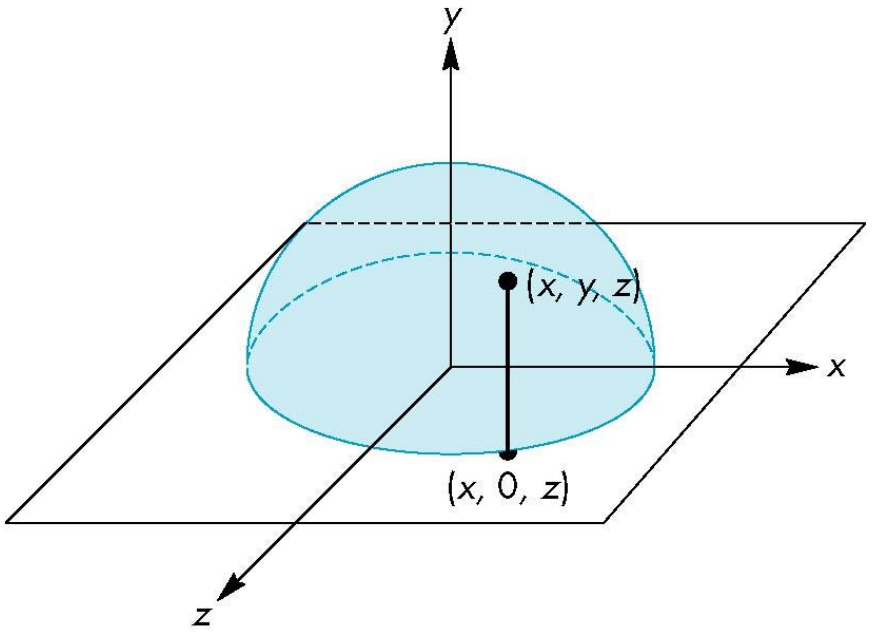
Trackball Frame



origin at center of ball

Projection of Trackball Position

- We can relate position on trackball to position on a normalized mouse pad by projecting orthogonally onto pad



Reversing Projection

- Because both the pad and the upper hemisphere of the ball are two-dimensional surfaces, we can reverse the projection
- A point (x,z) on the mouse pad corresponds to the point (x,y,z) on the upper hemisphere where

$$y = \sqrt{r^2 - x^2 - z^2} \qquad \text{if } r \geq |x| \geq 0,\ r \geq |z| \geq 0$$

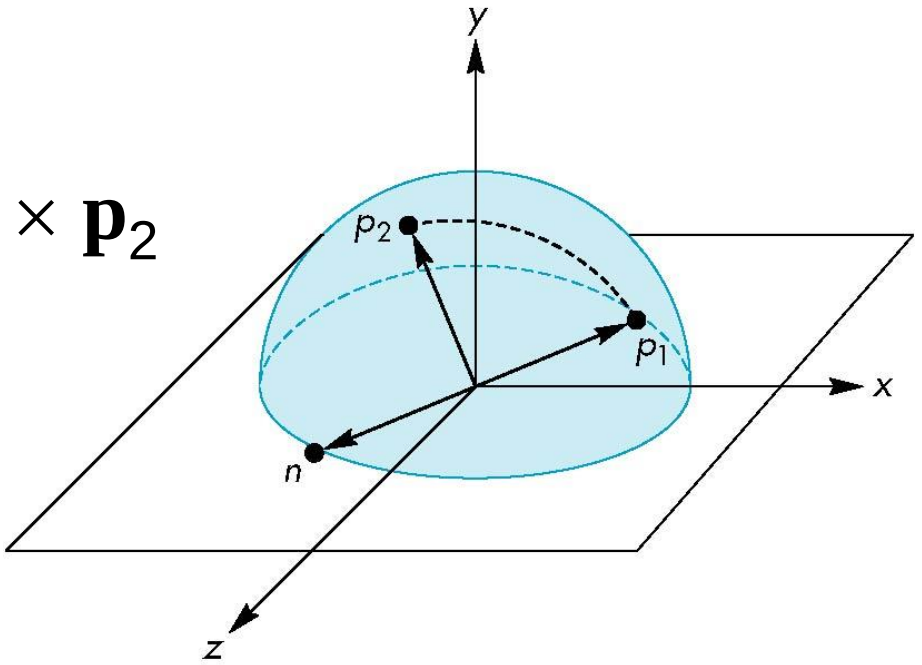
Computing Rotations

- Suppose that we have two points that were obtained from the mouse.
- We can project them up to the hemisphere to points $\mathbf{p}_1$ and $\mathbf{p}_2$
- These points determine a great circle on the sphere
- We can rotate from $\mathbf{p}_1$ to $\mathbf{p}_2$ by finding the proper axis of rotation and the angle between the points

Using the cross product

- The axis of rotation is given by the normal to the plane determined by the origin, $\mathbf{p}_1$, and $\mathbf{p}_2$

$$\mathbf{n} = \mathbf{p}_1 \times \mathbf{p}_2$$



Obtaining the angle

- The angle between $\mathbf{p}_1$ and $\mathbf{p}_2$ is given by

$$|\sin\theta| = \frac{|\mathbf{n}|}{|\mathbf{p}_1||\mathbf{p}_2|}$$

- If we move the mouse slowly or sample its position frequently, then θ will be small and we can use the approximation

$$\sin\theta \approx \theta$$

Implementing with WebGL

- Define actions in terms of three booleans
- **`trackingMouse`**: if true update trackball position
- **`redrawContinue`**: if true, idle function posts a redisplay
- **`trackballMove`**: if true, update rotation matrix

Vertex Shader I

```
in vec4 vPosition;
in vec4 vColor;
out vec4 color;
uniform vec4 rquat; // rotation quaternion

// quaternion multiplier

vec4 multq(vec4 a, vec4 b)
{
   return(vec4(a.x*b.x - dot(a.yzw, b.yzw),
       a.x*b.yzw+b.x*a.yzw+cross(b.yzw, a.yzw)));
}
```

Vertex Shader II

```
// inverse quaternion
vec4 invq(vec4 a)
{ return(vec4(a.x, -a.yzw)/dot(a,a)); }

  void main() {
  vec3 axis = rquat.yxw;
  float theta = rquat.x;
  vec4 r, p;
  p = vec4(0.0, vPosition.xyz);  // input point quaternion
  p = multq(rquat, multq(p, invq(rquat))); // rotated point quaternion
  gl_Position = vec4( p.yzw, 1.0); // back to homogeneous coordinates
  color = vColor;
}
```