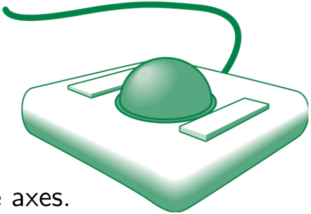
02561 Computer Graphics

Virtual trackball

Jeppe Revall Frisvad

November 2023

Let the orbiting of a camera follow the rotation of a sphere



- Rotating a sphere makes intuitive sense.
- But trackballs are not commonly available.
- Let us make a virtual trackball and attach it to the mouse (or a touch interface).
- Keep track of rotation angles θ_x, θ_y for two coordinate axes.
- Calculate a view matrix from two rotation matrices: $\boldsymbol{V} = \boldsymbol{R}_x(\theta_x)\,\boldsymbol{R}_y(\theta_y)\,.$
- Implement event handlers for mouse (or touch) events:
  - Use onmousedown/`touchstart` to register that an event started at position x_0, y_0.
  - Use onmouseup/touchend to register that an event ended.
  - Use onmousemove/touchmove to register new positions x, y and calculate changes:

$$\Delta\theta_x = \omega(x - x_0)\,, \quad \Delta\theta_y = \omega(y - y_0)\,,$$

where ω is the angular velocity. Then update angles θ_x, θ_y and positions x_0, y_0:

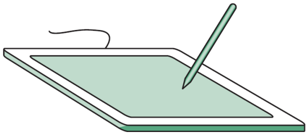
$$\begin{aligned} \theta_x &:= \theta_x + \Delta\theta_x\,, \quad x_0 := x \\ \theta_y &:= \theta_y + \Delta\theta_y\,, \quad y_0 := y\,. \end{aligned}$$

How to use the touch interface

- A touch event has a list of target touches (to support multi-touch).
- Each touch can be used as a mouse event.
- We can then connect a touch interface with a mouse interface using:

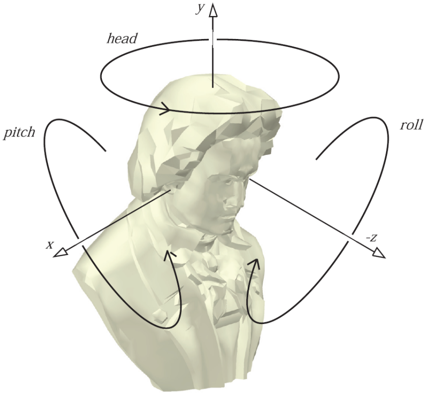
```
canvas.addEventListener("touchstart", function (ev) {
  ev.preventDefault();
  if (ev.targetTouches.length === 1) {
    var touch = ev.targetTouches[0];
    touch.preventDefault = function () { };
    touch.button = 0;
    canvas.onmousedown(touch);
    this.addEventListener("touchmove", roll, false);
    this.addEventListener("touchend", release, false);

    function roll(e) {
      touch = e.targetTouches[0];
      canvas.onmousemove(touch);
    }
    function release() {
      canvas.onmouseup(touch);
      this.removeEventListener("touchmove", roll);
      this.removeEventListener("touchend", release);
    }
  }
});
```



Euler angle rotation and gimbal lock

- Euler angle rotation is concatenation of rotations around fixed coordinate axes.
- The order of rotation is typically head, pitch, roll: $\boldsymbol{V} = \boldsymbol{R}_z(\theta_z)\,\boldsymbol{R}_x(\theta_x)\,\boldsymbol{R}_y(\theta_y)$.
- When rotating sequentially, we may after head rotation $\boldsymbol{R}_y(\theta_y)$ end up with pitch and roll being rotations around the same axis.
- We then lose one degree of freedom. This is called gimbal lock.
- To avoid gimbal lock, we need to avoid rotation around fixed axes.
- Quaternions [Hamilton 1844] provide a convenient axis-angle representation for rotations.



References

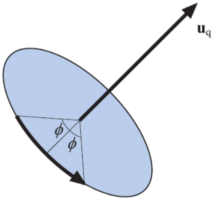
- Akenine-Möller, T., Haines, E., Hoffman, N., Pesce, A., Iwanicki, M., and Hillaire, S. *Real-Time Rendering*, fourth edition. CRC Press, 2018.
- Hamilton, W. R. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science (3rd series) 25*, pp. 10–13. 1844.

What is a quaternion?

- Generalization of complex numbers for representation of rotations.
- Specified as a 4-vector with a vector imaginary part $\boldsymbol{q}_v$ and a scalar real part q_w:

$$\hat{\boldsymbol{q}} = (\boldsymbol{q}_v, q_w) = iq_x + jq_y + kq_z + q_w\,, \quad \boldsymbol{q}_v = (q_x, q_y, q_z)\,, \quad i^2 = j^2 = k^2 = ijk = -1\,.$$

Multiplication:	$\hat{\boldsymbol{q}}\hat{\boldsymbol{r}}$	$=$	$(\boldsymbol{q}_v \times \boldsymbol{r}_v + r_w\boldsymbol{q}_v + q_w\boldsymbol{r}_v, q_w r_w - \boldsymbol{q}_v \cdot \boldsymbol{r}_v)\,.$
Addition:	$\hat{\boldsymbol{q}} + \hat{\boldsymbol{r}}$	$=$	$(\boldsymbol{q}_v + \boldsymbol{r}_v, q_w + r_w)\,.$
Conjugate:	$\hat{\boldsymbol{q}}^*$	$=$	$(-\boldsymbol{q}_v, q_w)\,.$
Norm:	$\mathrm{norm}(\hat{\boldsymbol{q}})$	$=$	$\sqrt{\hat{\boldsymbol{q}}\hat{\boldsymbol{q}}^*} = \sqrt{\boldsymbol{q}_v \cdot \boldsymbol{q}_v + q_w^2}\,.$
Identity:	$\hat{\boldsymbol{i}}$	$=$	$(\boldsymbol{0}, 1)\,.$
Inverse:	$\hat{\boldsymbol{q}}^{-1}$	$=$	$\hat{\boldsymbol{q}}^*/[\mathrm{norm}(\hat{\boldsymbol{q}})]^2\,.$

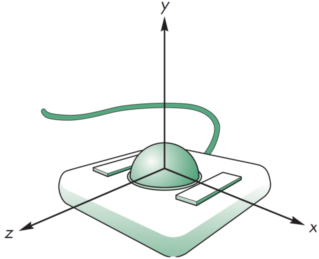


- Unit quaternion specifying rotation of 2ϕ radians around an axis $\boldsymbol{u}_q$:

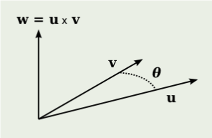
$$\hat{\boldsymbol{q}} = (sin\phi\, \boldsymbol{u}_q, \cos\phi)\,.$$

- Use homogeneous coordinates to convert a position or direction vector $\boldsymbol{p} = (p_x, p_y, p_z, p_w)$ to a quaternion $\hat{\boldsymbol{p}}$.
- Apply quaternion rotation using $\hat{\boldsymbol{q}}\hat{\boldsymbol{p}}\hat{\boldsymbol{q}}^{-1}$. For a unit quaternion, use $\hat{\boldsymbol{q}}\hat{\boldsymbol{p}}\hat{\boldsymbol{q}}^*$.

The quaternion trackball



- When rotating a trackball, the rotation of the sphere corresponds to the rotation of the virtual object.
- We can get the rotation by recording the start and end positions ($\boldsymbol{u}$ and $\boldsymbol{v}$) of the finger rotating the ball in the trackball frame (xyz).
- If we have two points on a unit sphere, we can make a unit quaternion that rotates from one point $\boldsymbol{u}$ to the other point $\boldsymbol{v}$:



$$\hat{\boldsymbol{q}} = \left(\sin\frac{\theta}{2}\boldsymbol{w}, \cos\frac{\theta}{2}\right) = \left(\frac{\boldsymbol{u}\times\boldsymbol{v}}{\sqrt{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}}, \frac{1}{2}\sqrt{2(1+\boldsymbol{u}\cdot\boldsymbol{v})}\right).$$

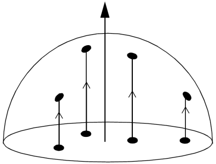
```
q_inc = q_inc.make_rot_vec2vec(normalize(u), normalize(v));
```

- For a quaternion trackball, keep track of two quaternions instead of two angles:
  - $\hat{\boldsymbol{q}}_{\text{rot}}$ the rotation away from the default view.
  - $\hat{\boldsymbol{q}}_{\text{inc}}$ the incremental rotation due to a mouse/touch event.
- Two quaternion rotations are concatenated by multiplication: $\hat{\boldsymbol{q}}_{\text{rot}} := \hat{\boldsymbol{q}}_{\text{rot}}\hat{\boldsymbol{q}}_{\text{inc}}$.

Mapping a flat mouse/touch interface to a sphere

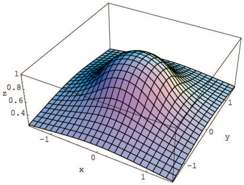
- The mouse/touch interface provides 2D screen space positions.
- Convert to $\boldsymbol{p} = (p_x, p_y) \in [0,1]^2$ as in the drawing program of Worksheet 2.
- We can then use orthographic projection with $d = \sqrt{p_x^2 + p_y^2}$ to get positions on the unit sphere:

  $$\boldsymbol{u} = \left(p_x, p_y, \sqrt{1 - d^2}\right)\ , \text{ for } d \leq \frac{1}{\sqrt{2}}.$$



- To retain an effect when the mouse/touch is outside the unit circle in the canvas, we use a hyperbolic sheet:

  $$\boldsymbol{u} = \left(p_x, p_y, \frac{1}{2d}\right)\ , \text{ for } d > \frac{1}{\sqrt{2}}.$$



- We can use this mapping to get the last mouse/touch position $\boldsymbol{u}$ and the current mouse/touch position $\boldsymbol{v}$ from which we can calculate $\hat{\boldsymbol{q}}_{\text{inc}}$.

Reference

- Henriksen, K., Sporring, J., and Hornbæk, K. Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics 10*(2), pp. 206–216. 2004.

Using a quaternion trackball for view transformation

- Initialize the trackball quaternions $\hat{\boldsymbol{q}}_{\text{rot}}$ and $\hat{\boldsymbol{q}}_{\text{inc}}$ to identity quaternions:

```
var q_rot = new Quaternion();
var q_inc = new Quaternion();
```

- Initialize eye point to $\boldsymbol{e} = (0, 0, z_{\text{eye}})$, look-at point to $\boldsymbol{a} = (0, 0, 0)$, and up vector to $\boldsymbol{y} = (0, 1, 0)$.
- Before rendering, apply $\hat{\boldsymbol{q}}_{\text{rot}}$ to the eye point and to the up vector, and get the view matrix using the look-at function:

  $\boldsymbol{V}$ = lookAt(q_rot.apply($\boldsymbol{e}$), $\boldsymbol{a}$, q_rot.apply($\boldsymbol{y}$));

- What if we want to start with arbitrary $\boldsymbol{e}$, $\boldsymbol{a}$, and $\boldsymbol{y}$?
- We still need to rotate around the origin, so we use $\boldsymbol{z} = \boldsymbol{e} - \boldsymbol{a}$ and

```
q_rot = q_rot.make_rot_vec2vec(vec3(0, 0, 1), normalize(z));
```

- Now, initially, $z_{\text{eye}} = \|\boldsymbol{z}\|$ and we update $\boldsymbol{y}$ to $\hat{\boldsymbol{q}}_{\text{rot}}^{-1}$ applied to $\boldsymbol{y}$, and then

  $\boldsymbol{V}$ = lookAt(add(q_rot.apply(vec3(0, 0, z_{eye})), $\boldsymbol{a}$), $\boldsymbol{a}$, q_rot.apply($\boldsymbol{y}$));

Dollying and panning

- Dollying is moving radially toward or away from the look-at point $\boldsymbol{a}$.
- We dolly by interactively updating z_{eye} using $(z_{\text{eye}} := z_{\text{eye}} + \Delta z_{\text{eye}})$
  - Mouse: difference (Δ) in the y-coordinate of the mouse position since last update.
  - Touch: difference (Δ) in distance between two touch positions since last update.
- Panning is moving the camera in a plane parallel to the image plane.
- Store a 2D vector $(x_{\text{pan}}, y_{\text{pan}})$ specifying displacement along $\boldsymbol{x} = (1, 0, 0)$ and up vector $\boldsymbol{y}$ after applying trackball rotation $\hat{\boldsymbol{q}}_{\text{rot}}$ to both.
- We pan by interactively updating $x_{\text{pan}}, y_{\text{pan}}$ using
  - Mouse: differences in the xy-coordinates of the mouse position since last udpate.
  - Touch: differences in the xy-coordinates of one touch position since last update, but in a two-touch event where the two touches moved in similar directions.
- We now replace the look-at point $\boldsymbol{a}$ by

$$\boldsymbol{c} = \boldsymbol{a} - \left(x_{\text{pan}}\, \hat{\boldsymbol{q}}_{\text{rot}} \boldsymbol{x} \hat{\boldsymbol{q}}_{\text{rot}}^{-1} + y_{\text{pan}}\, \hat{\boldsymbol{q}}_{\text{rot}} \boldsymbol{y} \hat{\boldsymbol{q}}_{\text{rot}}^{-1}\right).$$

  and then

  $\boldsymbol{V}$ = `lookAt(add(q_rot.apply(vec3(0, 0,` z_{eye}`)),` $\boldsymbol{c}$`),` $\boldsymbol{c}$`, q_rot.apply(`$\boldsymbol{y}$`));`

Spinning

- Normally, we update $\hat{\boldsymbol{q}}_{\text{rot}}$ by multiplication with $\hat{\boldsymbol{q}}_{\text{inc}}$ for every `onmousemove`/`touchmove` called during a trackball event.
- We could instead perform this update in our render loop (if some time passed since last update, e.g. 20 ms). This would enable spinning of the camera.
- How to stop the spin?
  - Stop the spin by setting $\hat{\boldsymbol{q}}_{\text{inc}}$ to an identity quaternion (`qinc.setIdentity();`).
- When to stop the spin? Use onmouseup and
  - Stop spinning if the last mouse/touch position was the same as the current mouse/touch position when the trackball event ended.
  - Stop spinning if some time (e.g. 20 ms) passed since the last mouse/touch position was recorded.