CMCS427 Notes on moving the camera

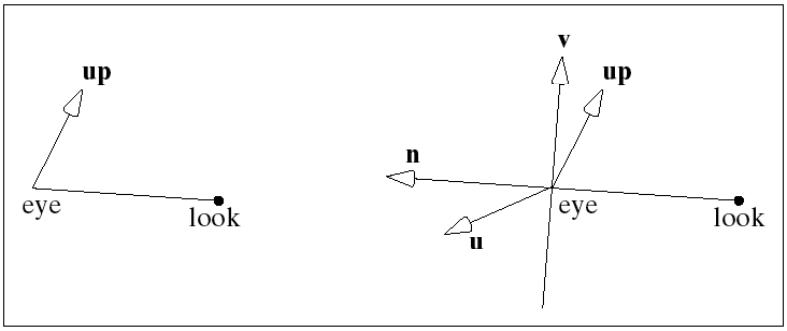
NOTE: Here eye = e = at, and look=d=lookAt, with e and d the notation on the slides.
And u,v,n are the camera vectors that we call xc, yc and zc.

Lecture:

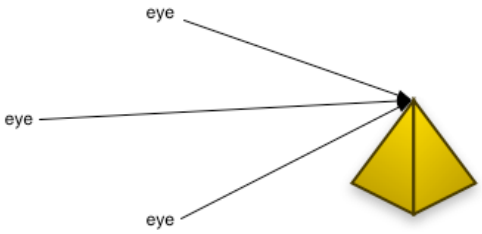
I. Animating the camera

We can change:

Position	change eye in camera(at, lookAt, up)
Direction	change lookAt
Zoom	change FOV in perspective()



II. Making the change



A. Change *eye* alone
Camera moves but continues to look at same place

B. Change *look* alone
What happens?

C. Change *up*
What happens?

III. Animating the camera along a parametric curve

If we want to use a parametric curve to animate the camera, we can use this code:

in idle callback:	t += Delta_t;

in display callback:	x = px(t);
	y = py(t);
	z = pz(t);
	camera(x,y,z, 0, 0, 0, 0,1,0);

Now the camera moves by the parametric curve but continues to look at the origin.

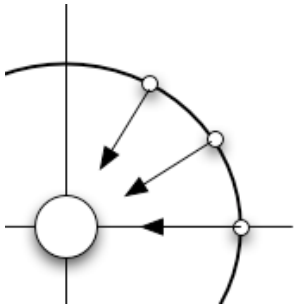
IV. Cases in camera animation (we filled out these examples in class).

A. A satellite viewing the earth
Assume circling in x,y plane, radius R

look = < , , >

eye = < , , >

up = < , , >

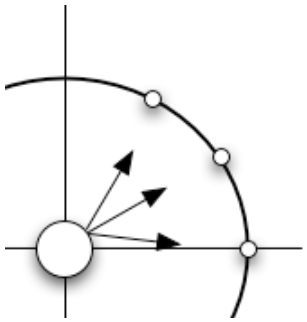


B. Watching a satellite orbit at a radius R
when we are at position (x,y,z) on the planet

look = < , , >

eye = < , , >

up = < , , >

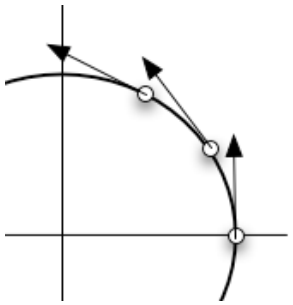


C. A person on a carnival ride looking
straight ahead (this time in x,z plane)

look = < , , >

eye = < , , >

up = < , , >



D. A cannon ball ride watching the target
(Picture on the board)

look = < , , >

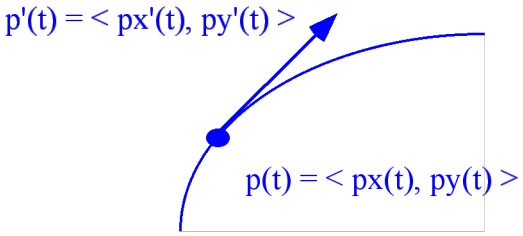
eye = < , , >

up = < , , >

E. A video game character controlled by the keyboard on x,z plane (no jumps)
Character moves forward at position p with forward vector v
k – accelerates forward
, - accelerates backward
j – turns left
l – turns right

Notes on differential geometry and Frenet frames
Case D suggested a different analysis based on differential geometry concepts.

Using the tangent vector:
If we want to look ahead, so our view follows the curve, we need the tangent vector for the curve. For a parametric vector curve this is the coordinate-wise derivative of the curve with respect to t.



For curve p(t) = < px(t), py(t) > the tangent function is p'(t) = < px'(t), py'(t) >

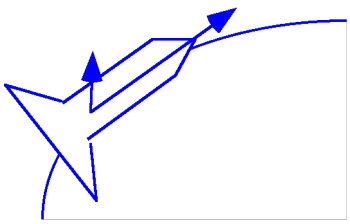
For the 2d circle we have p'(t) = < -2π sin 2πt, 2π cos 2πt >

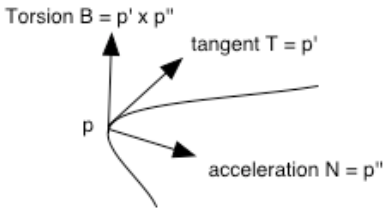
Now to animate we do this:

in idle callback: t += Delta_t;

in display callback:
x = px(t); dx = px'(t);
y = py(t); dy = py'(t);
z = pz(t); dz = pz'(t);
camera(x,y,z, x + dx, y + dy, z + dz, 0,1,0);

The Frenet frame
The Frenet frame is a local coordinate system defined on a parametric curve. It has u,v,n vectors like the synthetic camera and gives a sense of local orientation.





Note: the book defines the Frenet frame on pages 316-318. The terms it uses can be better understood by using slightly more intuitive terms as below:

What the book calls "acceleration" is often called "curvature vector"
What the book calls "binormal" is often called "torsion vector"

Math alert: This technique is advanced. if we want to animate along a 3D curve in space we may not want our head to always point up wrt gravity. Consider a flight simulator with a plane making a sharp bank; the pilot's head should bank with the plane. Then we need some nasty differential geometry to compute the torsion vector wrt the curve, which we use as the UP vector. Here's what we do:

- We take the first derivative of the curve to get the tangent vector $T(t) = p'(t)$

- We take the second derivative of the curve to get the acceleration vector $N(t) = p''(t)$. The acceleration vector points in the direction that the curve is turning; its length is the rate of change (tighter the curve, longer the vector).

  NOTE: the book takes an extra step to compute B and N as a safety measure, but for our purposes this approach is fine.

- We take the cross-product of T and B to get the binormal vector $B(t) = p'(t) \times p''(t)$. The biormal or torsion vector is the local up for the curve; it points in towards the bank.

Example: for the helix (skipping the 2π to make the equations simple).

$p(t) = < \cos t, \sin t, ht >$ $\quad B(t) = p'(t) \times p''(t) = \begin{vmatrix} i & j & k \\ -\sin & \cos & h \\ -\cos & -\sin & 0 \end{vmatrix}$

$T(t) = p'(t) = < -\sin t, \cos t, h >$ $\quad = < -h \sin, h \cos, \sin^2 + \cos^2 >$

$N(t) = p''(t) = < -\cos t, -\sin t, 0>$ $\quad = < - h \sin, h \cos, 1 >$

Now we stick B(t) in for the UP vector in camera.

We use the same techniques if we want to bank a shape that we are looking at (not from). Assume we have a model of a jet fighter. We need to rotate the fighter so the nose points along the tangent vector and the cockpit points up along the torsion vector (otherwise the shape moves always with the same orientation, say pointing down the X-axis.)