Witkin and Kass --
How they might have made Luxo, Jr.

Animation involves
- **character design**
- **character motion**
- **personality and appeal**

Look at character motion
- **Squash and stretch**
- **anticipation and followthrough**

Can we do these by computer?

Key observations
- **You can only accelerate by pushing off of something else**
- **When you land, there is extra momentum**

Plan
- **Understand and build off of the underlying physics.**

Physically based character animation

User wants to specify

1. **What the character has to do**
   e.g. jump from here to there

2. **How the motion should be performed**
   e.g. don’t waste energy,
   come down hard enough to splatter what you land on

3. **What is the physical structure of the character?**

4. **What physical resources are available**
   e.g. a floor to push off of
   muscles on the character
   ...

Problem Formulation -- The SpaceTime Method

Particle position at time t is x(t)
force on particle is f(t).

$$mx'' - f - mg = 0$$

equation of motion for particle.

We typically know f(t) and x(t), x'(t) at some t = t0 and then we solve.

Suppose we want the particle to fly from a fixed place a at t=t0 to a fixed place b at t=t1 at the minimum cost in energy (~ f^2).

Then, we have the problem of finding f(t) such that x(t0) = a, x(t1) = b, and

R = integral(t0,t1) |f(t)|^2 dt

is minimized.

Formulating a solution

Can represent everything discretely, having x_i ($i = 0$, ... N) for positions and f_i (($i = 0$, ... N) for forces.

Then,

$x' = (x_i - x_{i-1})/h, \ldots)$

$x'' = (x_{i+1} - 2x_i + x_{i-1})/h^2$

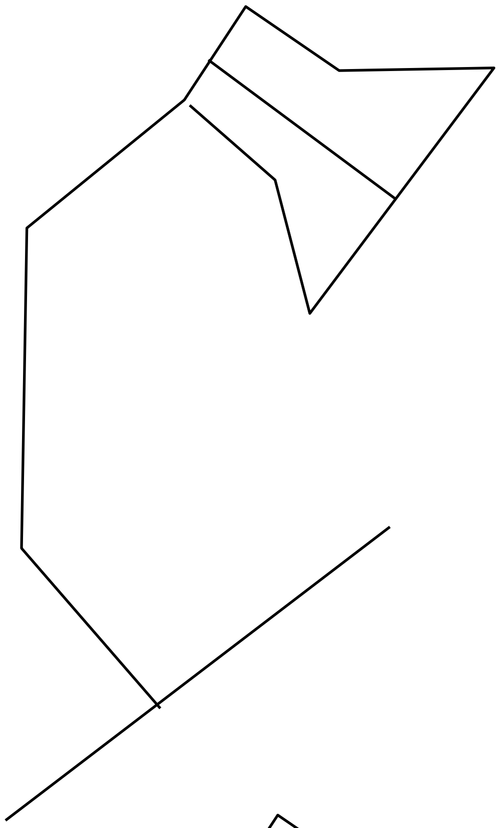
and integrals become summations.

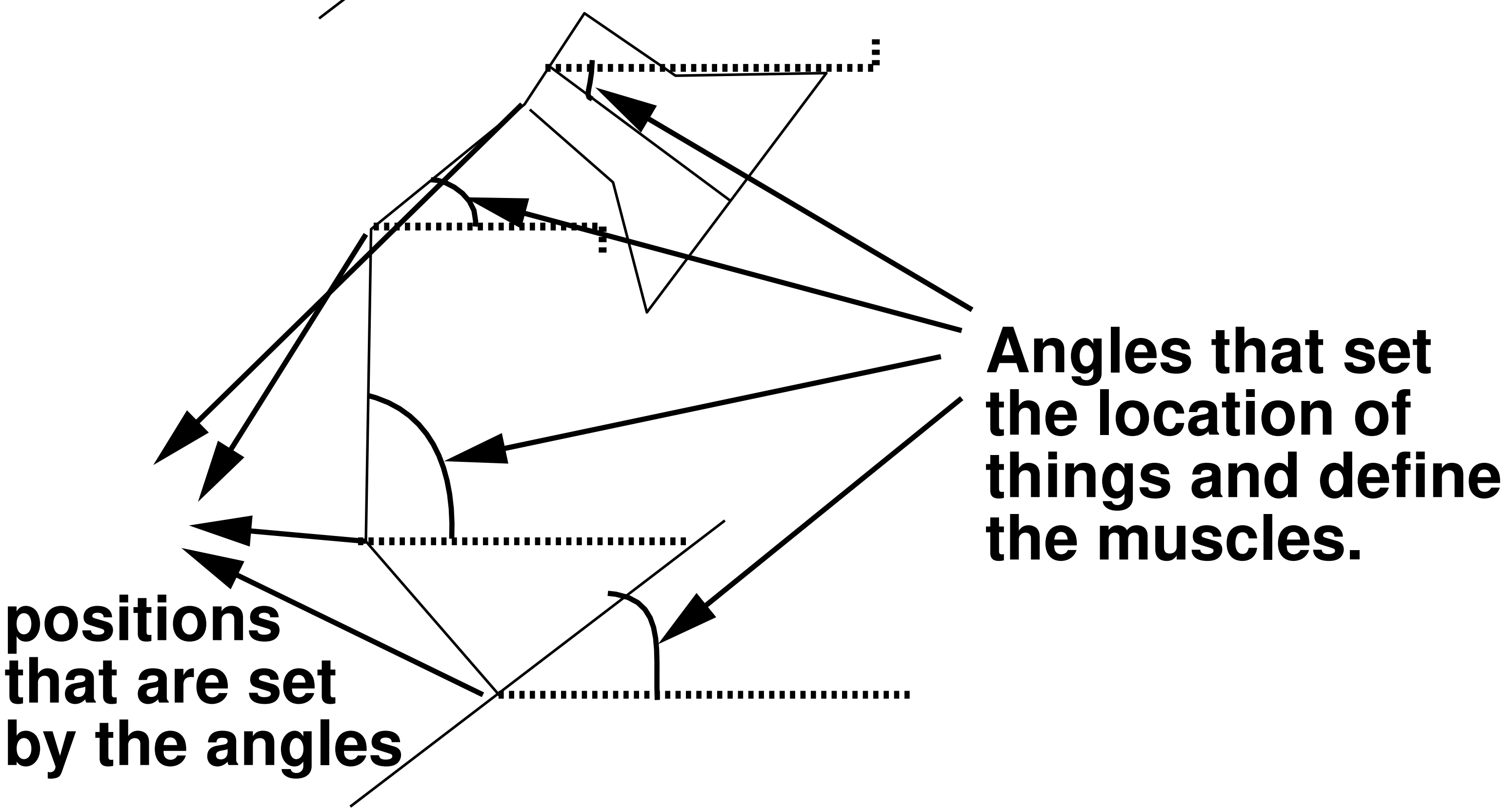
$R = h \sum_{i} f_1^2$

When done, we get a system of constraints, some linear, some non-linear. that govern the situation.

Solve the system of equations and a set of positions and forces result.

Defining objects (enter Luxo)





Added constraints

Kinetic energy becomes a more complex thing needing to account for energies of the links

Momentum has angular components...

We want spring constants to define how the springs(i.e. muscles) work.

More conservation expressions.

End result, it’s harder to solve, but the techniques work.

Image Morphing

We've seen it in

- Composite News
- T2
- Black or White (Michael Jackson)
- too many commercials

Simple ideas

- key frame animation.

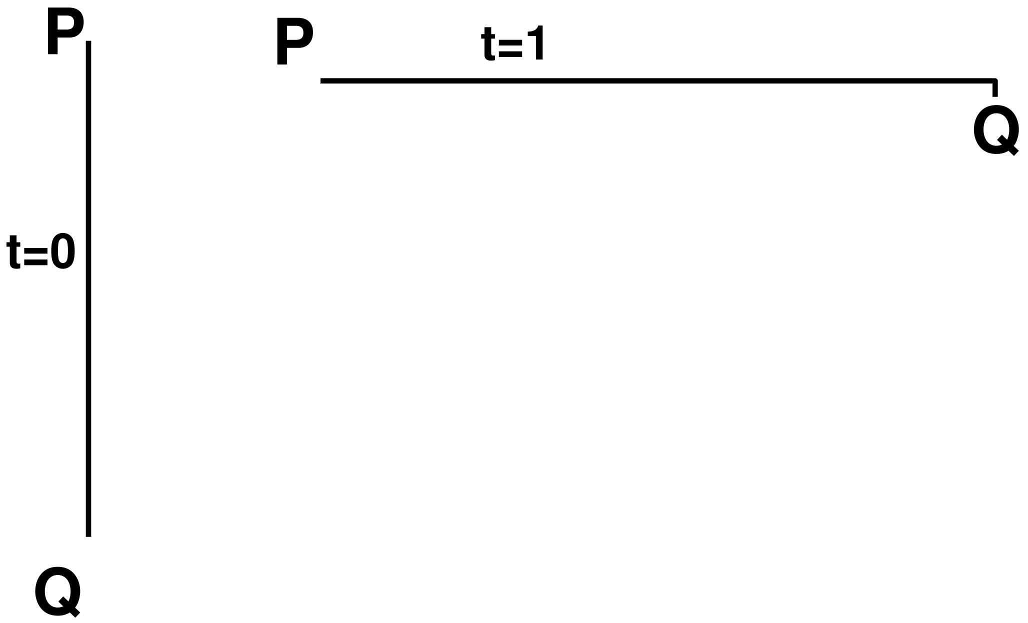
  If A is here at time 0 and there at time 1, it should be halfway between at time .5

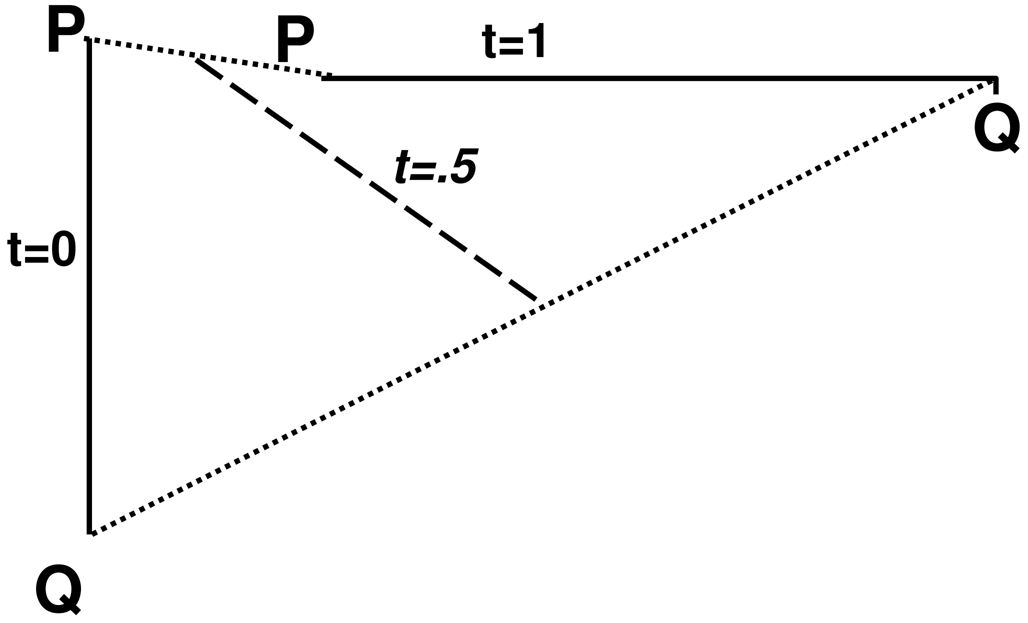
- pixel based cross–dissolve

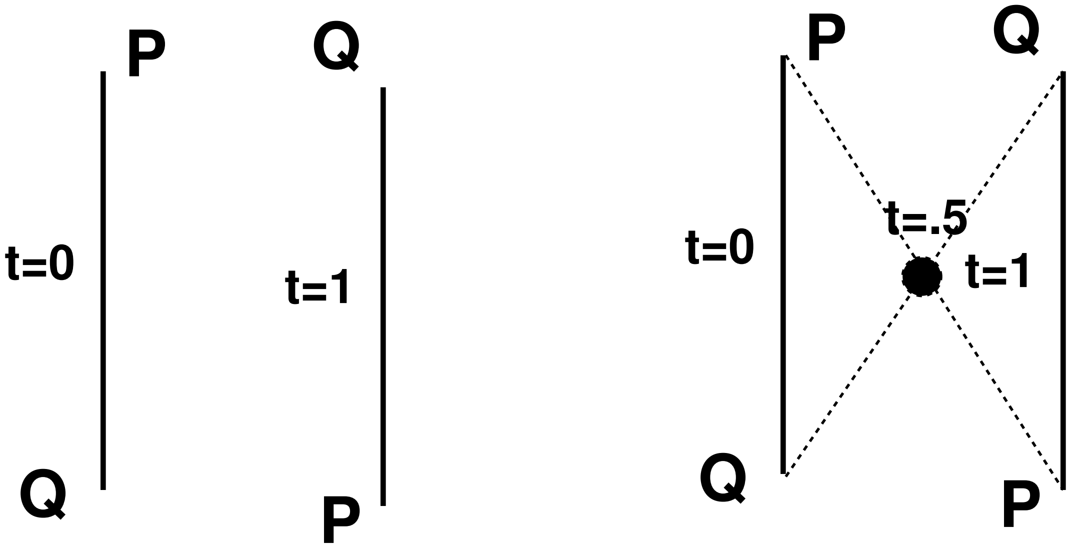
  if you start from image A and end at image B, you show image $(1-t)A + tB$ at time t.

- image morphing

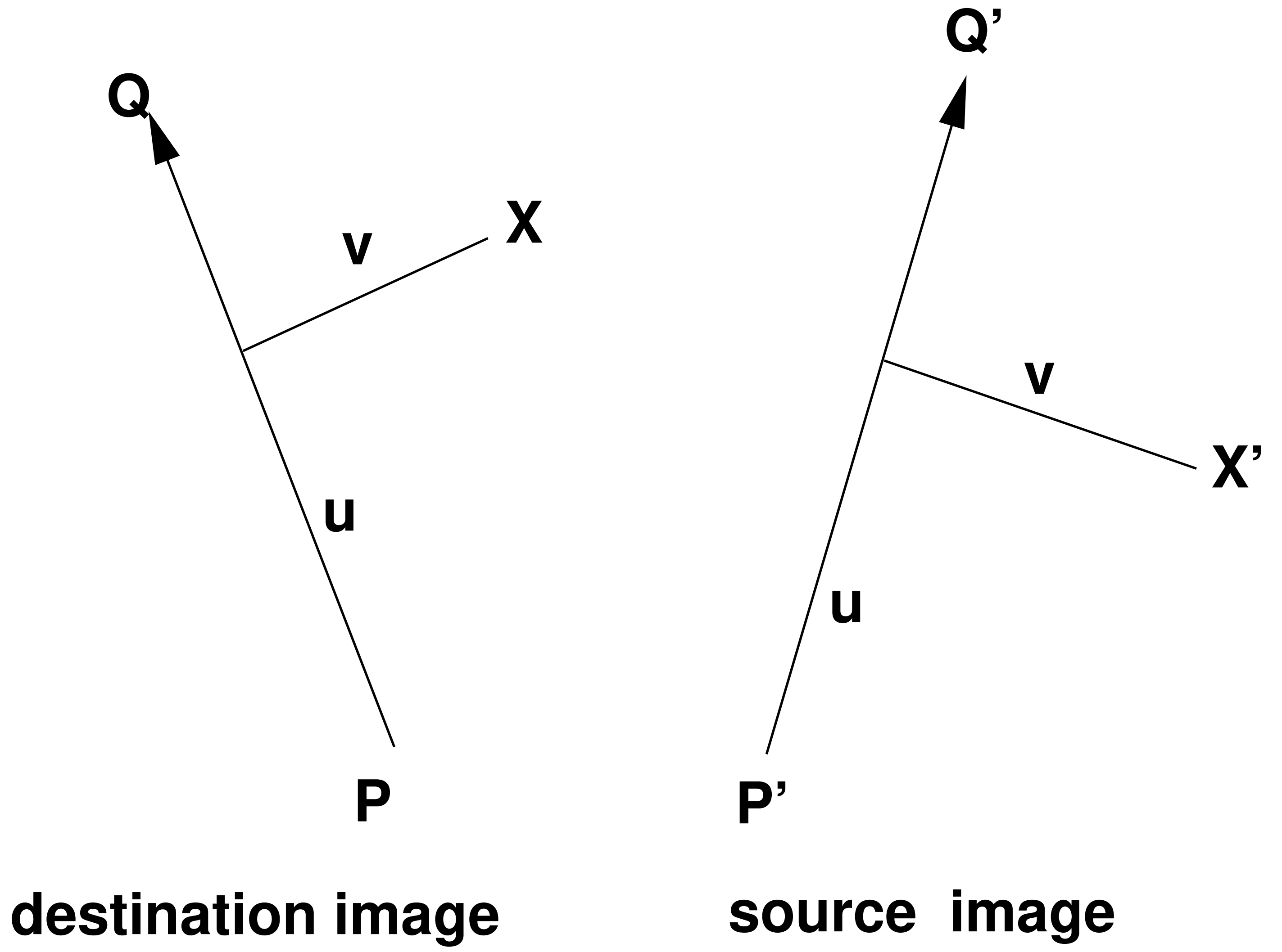
  combine these 2 ideas.


P
t=0
Q
P
t=1
Q


P
t=0
Q
P
t=1
Q
t=.5


P
t=0
Q
Q
t=1
P
P
Q
t=.5
t=0
t=1
Q
P

Transforming between 2 lines



Point X in the destination image corresponds to point X' in the destination image.

Given PQ, we found the perpendicular from X to PQ and measured the values u and v in that coordinate system.

We then plotted X' as coordinates u,v in the P'Q' coordinate system.

Mathematically,

$$u = ((X-P) * (Q-P)) / ||Q - P||^2$$

$$v = ((X-P) * Perp(Q-P))/||Q - P||$$

$$X' = P' + u * (Q' - P') + (v * Perp(Q' - P'))/||Q'-P'||$$

where Perp(.) returns a vector perpendicular to and of the same length as the input vector.

Once we can compute X from X', we sample the source image at X' to determine the value of the destination image at X.

The algorithm transforms each pixel by rotation, translation and/or scaling.

Pixels along the line P'Q' become pixels along the line PQ.

u value is normalized by length of line
v coordinate is distance in pixels

So, the image is scaled along the direction of the lines by the ratio of the lengths of the lines.

Example images

Can get pure rotation if both lines are the same length.

Can get translation if lines are same length and orientation.

Can't get uniform scaling or shear.

Transforming by Multiple pairs of lines

Compute Xi' for each pair of lines.

Measure Di = Xi' – Xi, displacement of ith X

Compute a weighted average of these displacements.

weight = ((length^p)/(a+dist))^b

where length is the length of the line
dist is the distance from the pixel to the line

a,b,p are constants tweaked to the desired impact of the line.

a=0 makes points on a line go exactly where the user wants them. Larger a causes warping which displaces the line but gives more continuity.

larger values of b cause the effect of a line on a pixel to drop off at greater distances. b = 0 makes all lines have the same weight. Typically, b in [0.5,2].

p is typically in [0,1].

Pseudo code

```
DSUM = (0,0)
weightsum = 0
For each line PiQi
        calculate u,v based on Pi’Qi’
        calculate Xi’ based on u,v,Pi’Qi’
        calculate displacement Di = Xi’–Xi
        dist = shortest distance from X to PiQi
        weight = ((length^p)/(a+dist))^b
        DSUM += Di * weight
        weightsum  += weight
X’ = X + DSUM/weightsum
destinationImage(X) = sourceImage(X’)
```

Morphing between 2 images

To blend between I0 and I1...

Define corresponding lines in I0 and I1

Each intermediate frame I is defined by creating a new set of line segments by interpolating the lines from their positions in I0 to their positions in I1.

Both Images I0 and I1 are distorted toward the position of the lines in I.

The two resulting images are cross–dissolved.

We start at I0 (no changes have occurred)
end at I1 (all changes have occurred)

and along the way morph our way along.

Lines can be interpolated from endpoints (as above) or from center point, orientation and length.

1st method can have line shrink to a point.
2nd method is non–intuitive to the user.