CS4670: Computer Vision

Noah Snavely

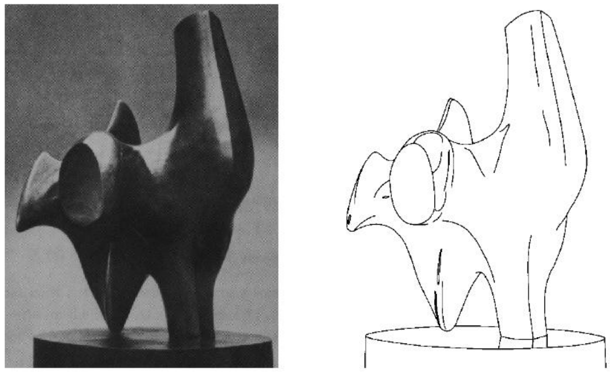
Lecture 2: Edge detection

From [Sandlot Science]

Announcements

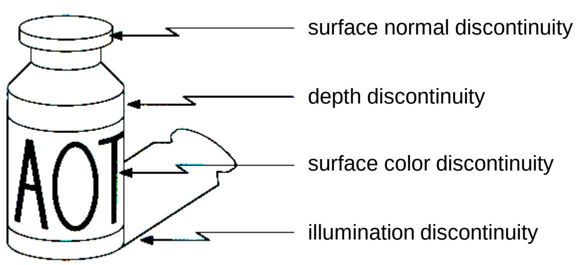
- Project 1 released, due Friday, September 7

Edge detection



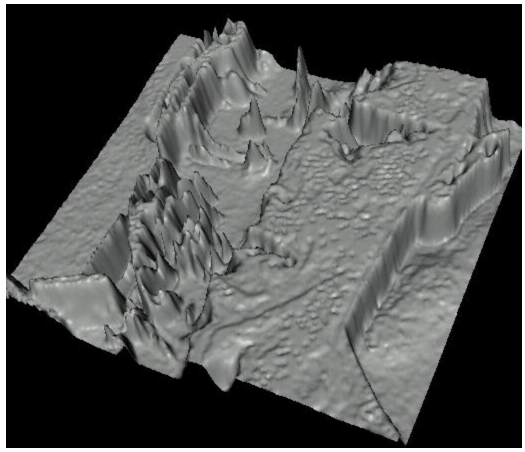
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

Origin of Edges



- Edges are caused by a variety of factors

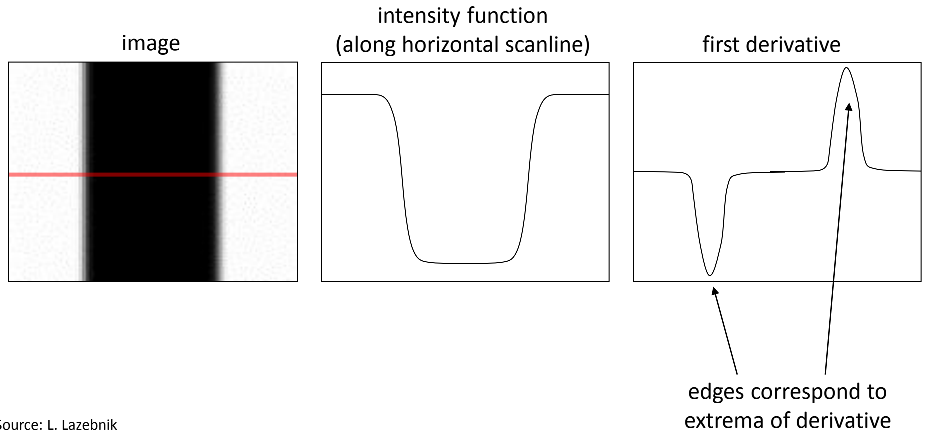
Images as functions…



- Edges look like steep cliffs

Characterizing edges

- An edge is a place of rapid change in the image intensity function



Source: L. Lazebnik

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f*, then compute the derivative
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x,y] \approx F[x+1,y] - F[x,y]$$

How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:

H_x

$\frac{\partial f}{\partial y}$:

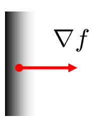
H_y

Source: S. Seitz

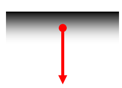
Image gradient

- The *gradient* of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

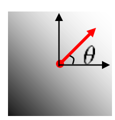
The gradient points in the direction of most rapid increase in intensity



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

The *edge strength* is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

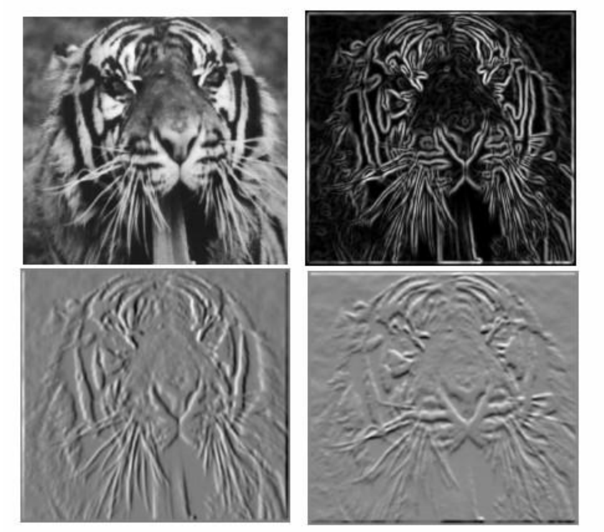
The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

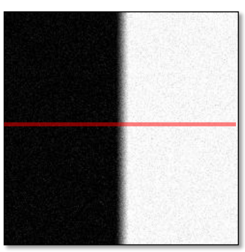
Source: Steve Seitz

Image gradient

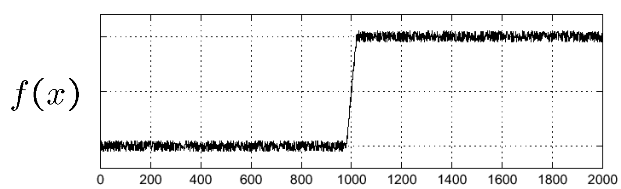


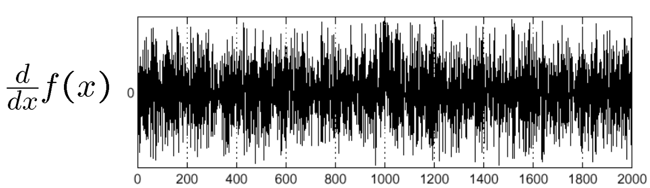
Source: L. Lazebnik

Effects of noise



Noisy input image

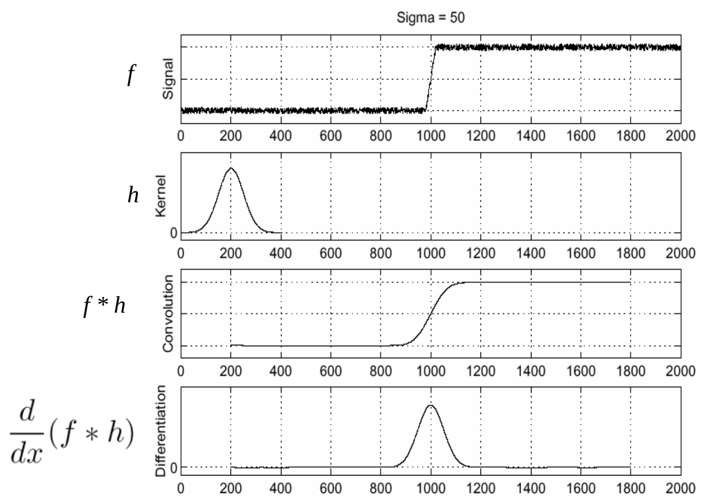




Where is the edge?

Source: S. Seitz

Solution: smooth first

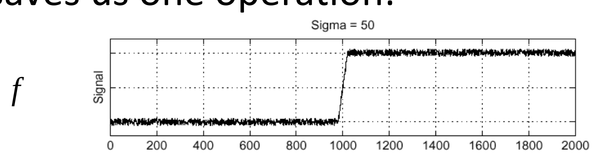


To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

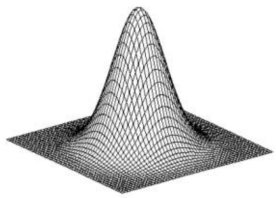
Associative property of convolution

- Differentiation is convolution, and convolution is associative: $\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$
- This saves us one operation:



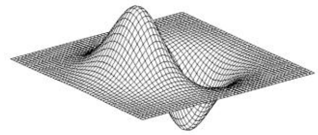
Source: S. Seitz

2D edge detection filters



Gaussian

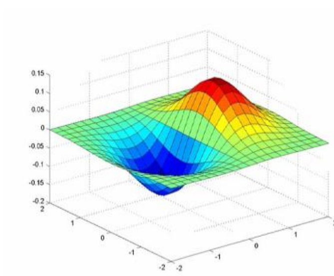
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



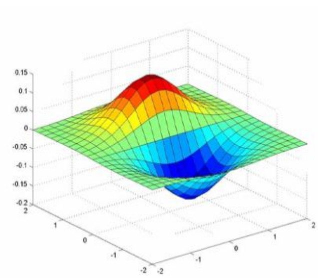
derivative of Gaussian (*x*)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

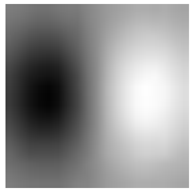
Derivative of Gaussian filter

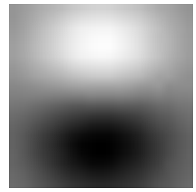


x-direction



y-direction





The Sobel operator

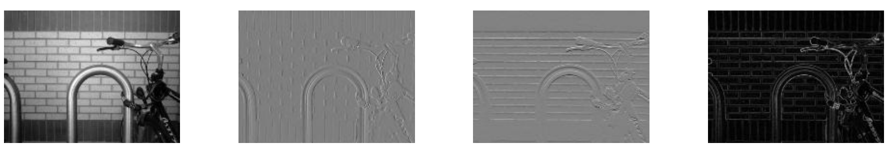
- Common approximation of derivative of Gaussian

$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$s_x \qquad\qquad s_y$$

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value

Sobel operator: example



Source: Wikipedia

Example

- original image (Lena)

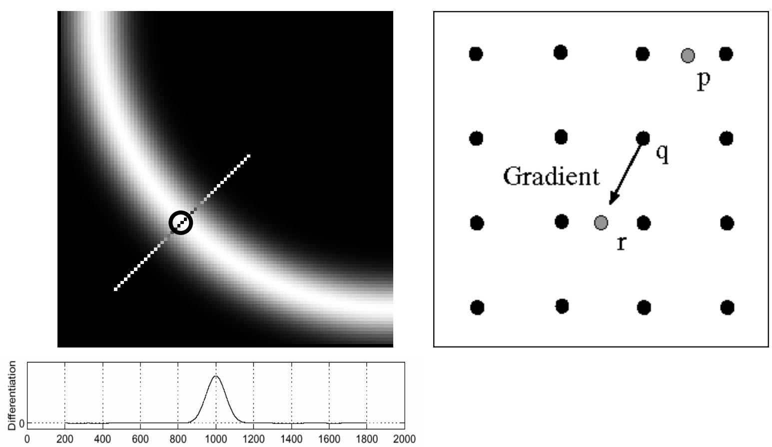
Finding edges

gradient magnitude

Finding edges

thresholding

Non-maximum supression



- Check if pixel is local maximum along gradient direction
  - requires *interpolating* pixels p and r

Finding edges

thresholding

Finding edges

thinning

(non-maximum suppression)

Canny edge detector

MATLAB: `edge(image,'canny')`

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
4. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

Source: D. Lowe, L. Fei-Fei

Canny edge detector

- Still one of the most widely used edge detectors in computer vision

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

- Depends on several parameters:

σ : width of the Gaussian blur

high threshold
low threshold

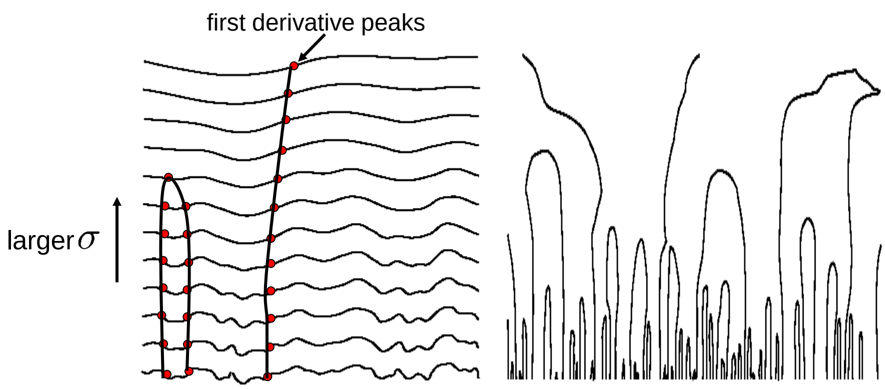
Canny edge detector

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - large σ detects "large-scale" edges
  - small σ detects fine edges

Source: S. Seitz

Scale space (Witkin 83)



Gaussian filtered signal

- Properties of scale space (w/ Gaussian smoothing)
  - edge position may shift with increasing scale (σ)
  - two edges may merge with increasing scale
  - an edge may ***not*** split into two with increasing scale

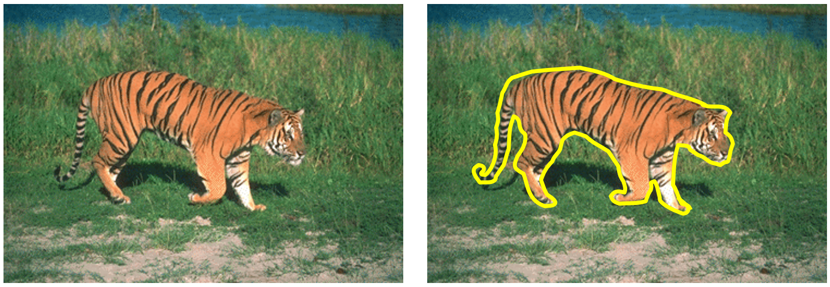
Questions?

Image Scissors

Aging Helen Mirren

- Today's Readings
  - Intelligent Scissors, Mortensen et. al, SIGGRAPH 1995

Extracting objects



- How could this be done?
  - hard to do manually
  - hard to do automatically ("image segmentation")
  - pretty easy to do *semi-automatically*

Intelligent Scissors (demo)

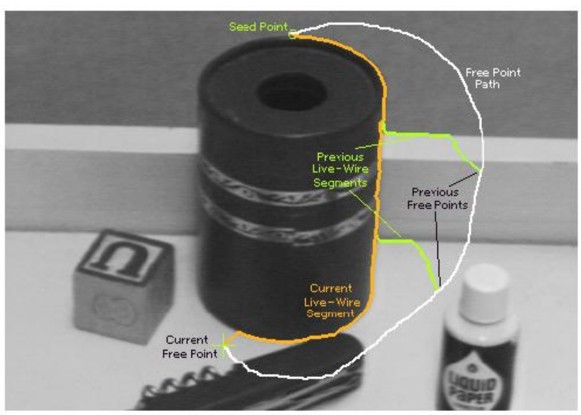
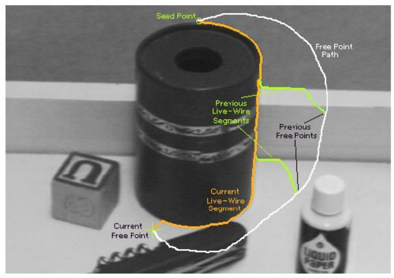


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

- Approach answers basic question
  - Q: how to find a path from seed to mouse that follows object boundary as closely as possible?
  - A: define a path that stays as close as possible to edges



Intelligent Scissors

- Basic Idea
  - Define edge score for each pixel
    - edge pixels have low cost
  - Find lowest cost path from seed to mouse

11	13	12	9	5	8	3	1	2	4	10
14	11	7	4	2	5	8	4	6	3	8
11	6	3	5	7	9	12	11	10	7	4
7	4	6	11	13	18	17	14	8	5	2
6	2	7	10	15	15	21	19	8	3	5
8	3	4	7	9	13	14	15	9	5	6
11	5	2	8	3	4	5	7	2	5	9
12	4	2	1	5	6	3	2	4	8	12
10	9	7	5	9	8	5	3	7	8	15

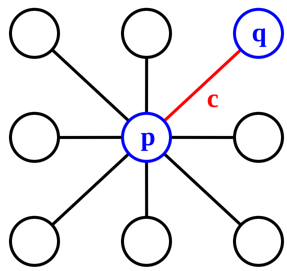
mouse

seed

Questions
- How to define costs?
- How to find the path?

Let's look at this more closely

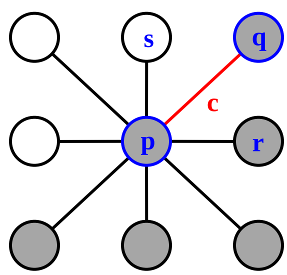
- Treat the image as a graph



Graph
- node for every pixel **p**
- link between every adjacent pair of pixels, **p,q**
- cost **c** for each link

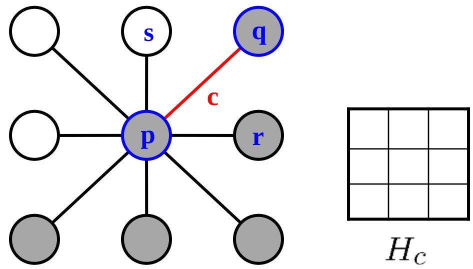
Note: each *link* has a cost
- this is a little different than the figure before where each pixel had a cost

Defining the costs



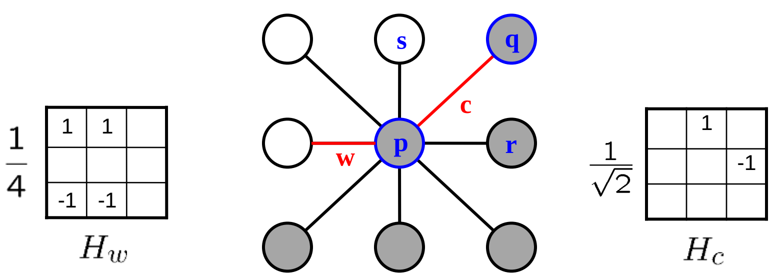
Want to hug image edges: how to define cost of a link?
- good (low-cost) links follow the intensity edge
  - want intensity to change rapidly ⊥ to the link
- $c \approx - \frac{1}{\sqrt{2}}$ |intensity of r – intensity of s|

Defining the costs



- c can be computed using a cross-correlation filter
  - assume it is centered at p

Defining the costs



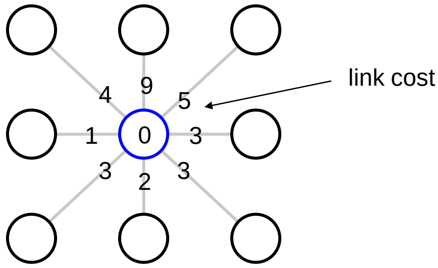
c can be computed using a cross-correlation filter

- assume it is centered at p

A couple more modifications

- Scale the filter response by length of link c. Why?
- Make c positive
  - Set c = (max-|filter response|*length)
  - where max = maximum |filter response|*length over all pixels in the image

Dijkstra’s shortest path algorithm



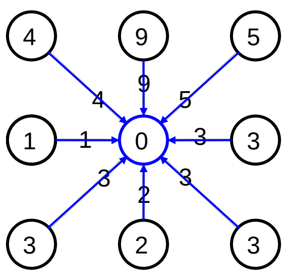
Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

   for each of p’s neighbors q that are not expanded

   set cost(q) = min(cost(p) + c_{pq}, cost(q))

Dijkstra’s shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

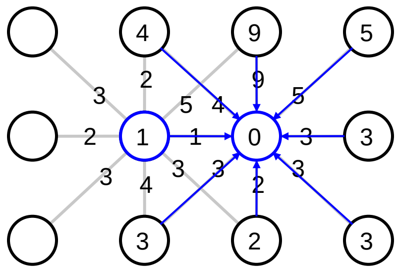
   for each of p’s neighbors q that are not expanded

   set cost(q) = min(cost(p) + c_{pq}, cost(q))

   if q’s cost changed, make q point back to p

   put q on the ACTIVE list (if not already there)

Dijkstra’s shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

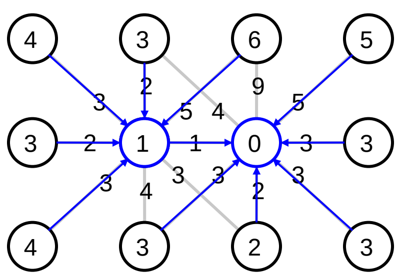
   for each of p’s neighbors q that are not expanded

      set cost(q) = min(cost(p) + c_{pq}, cost(q))

      if q’s cost changed, make q point back to p

      put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra’s shortest path algorithm



Algorithm

1. init node costs to ∞, set p = seed point, cost(p) = 0
2. expand p as follows:

   for each of p’s neighbors q that are not expanded

      set cost(q) = min(cost(p) + c_{pq}, cost(q))

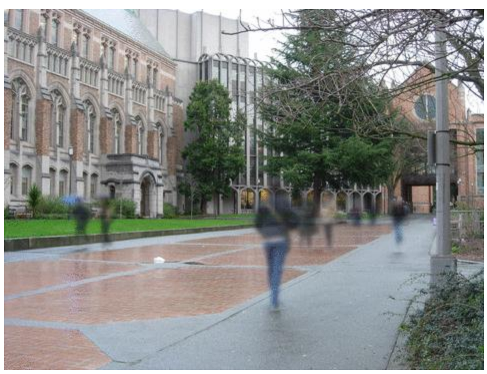
      if q’s cost changed, make q point back to p

      put q on the ACTIVE list (if not already there)
3. set r = node with minimum cost on the ACTIVE list
4. repeat Step 2 for p = r

Dijkstra's shortest path algorithm

- Properties
  - It computes the minimum cost path from the seed to every node in the graph. This set of minimum paths is represented as a *tree*
  - Running time, with N pixels:
    - $O(N^2)$ time if you use an active list
    - O(N log N) if you use an active priority queue (heap)
    - takes fraction of a second for a typical (640x480) image
  - Once this tree is computed once, we can extract the optimal path from any point to the seed in O(N) time.
    - it runs in real time as the mouse moves
  - What happens when the user specifies a new seed?

Example Result



Peter Davis

Questions?