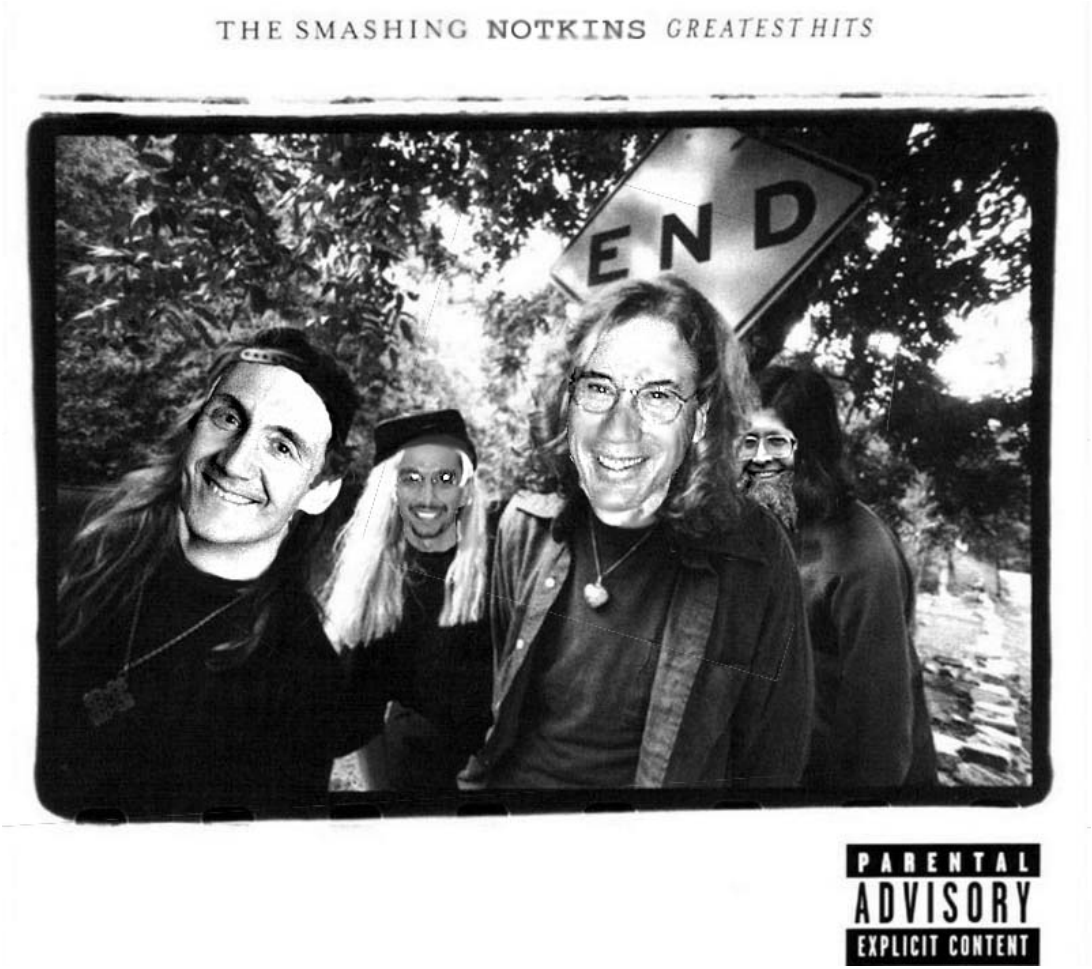
Lecture 4

Image Scissors (for Fun and Profit)

From: “Edward Scissorhands”

Project 1: Image Scissors

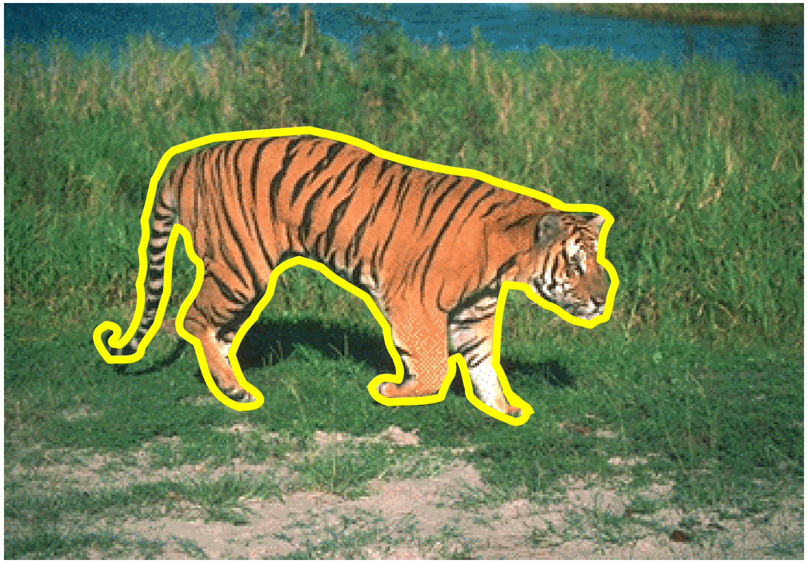


By Melissa Garcia, CSE 455 (2003)

Read:

- Intelligent Scissors, Mortensen et. al, SIGGRAPH 1995

Extracting objects



How could this be done?

- Manually? Tedious…
- Automatically? (“Image segmentation”) Too hard…
- Solution: Do it *semi-automatically*

Intelligent Scissors (demo)

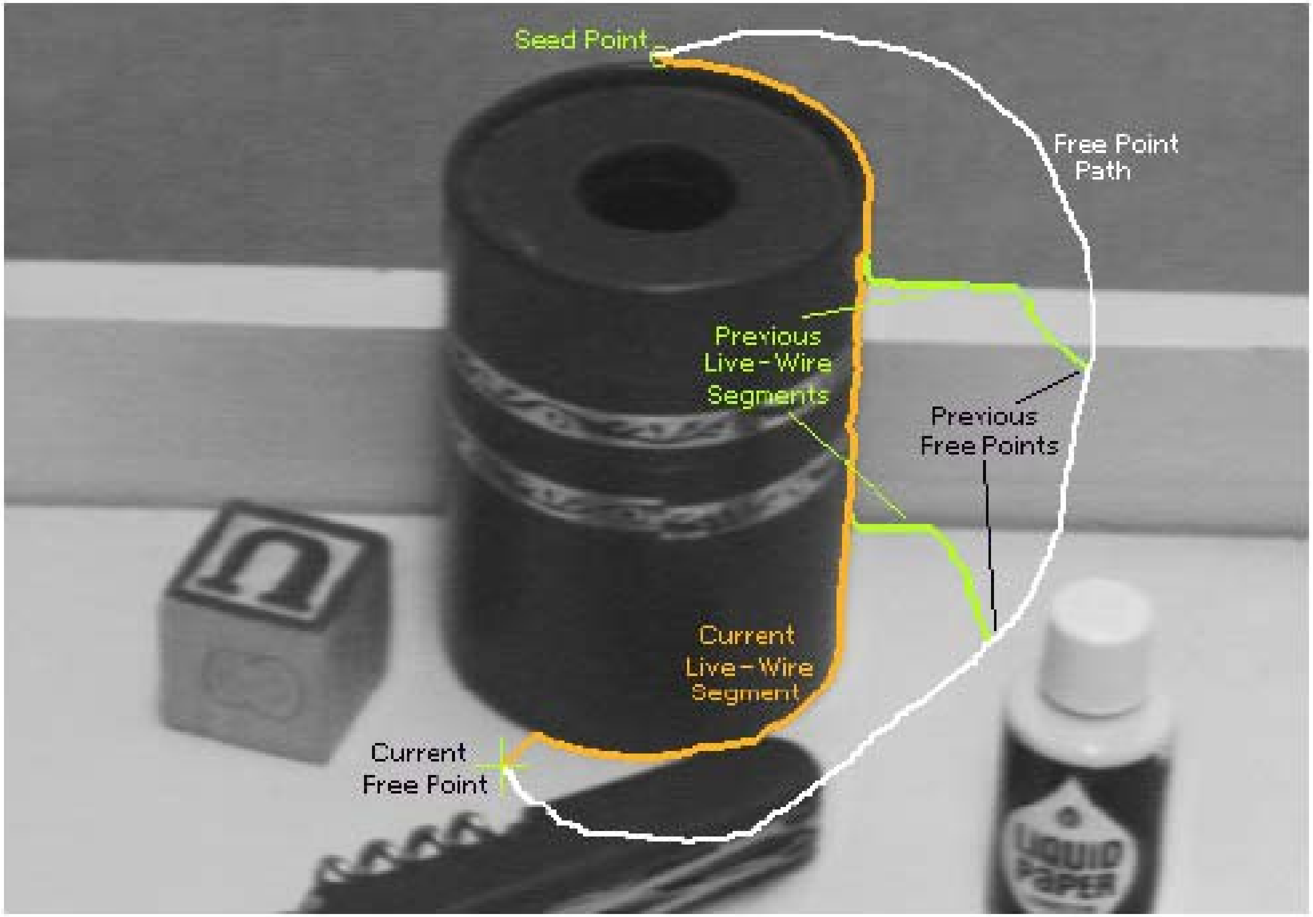


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

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

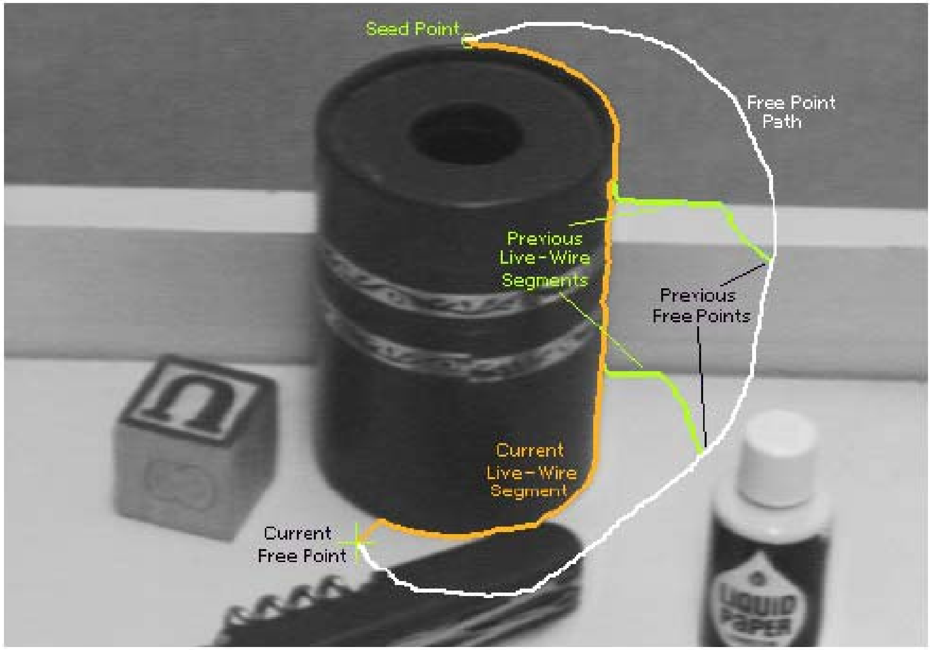


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

Basic Idea

- Define edge score for each pixel
  - edge pixels have low cost
- Compute lowest cost paths from seed to all other pixels
- Given mouse position, output 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

Intelligent Scissors

Basic Idea

- Define edge score for each pixel
  - edge pixels have low cost
- Compute lowest cost paths from seed to all other pixels
- Given mouse position, output 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

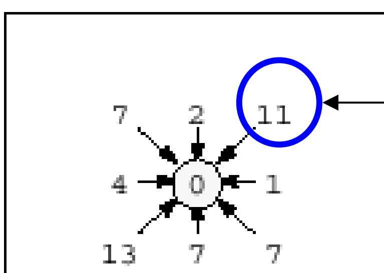
Computing shortest paths (basic idea)

Graph Search Algorithm

- Computes minimum cost path from seed to *all other pixels*

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

$$8 * \text{length} = 8\sqrt{2} \approx 11$$



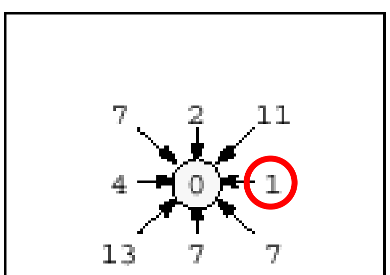
Computing shortest paths (basic idea)

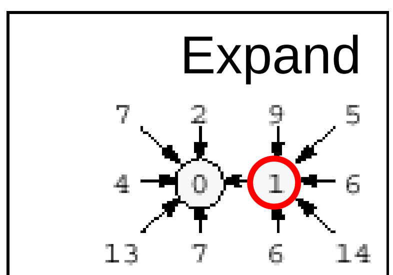
Graph Search Algorithm

- Computes minimum cost path from seed to *all other pixels*

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

Find smallest

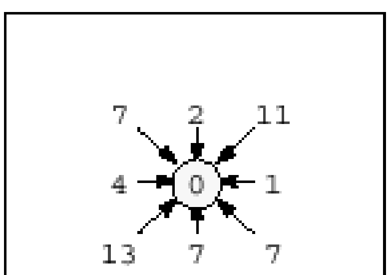


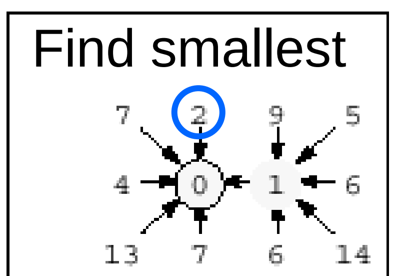


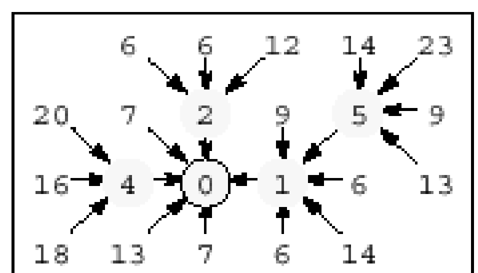
Computing shortest paths (basic idea)

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

Expand and so on…

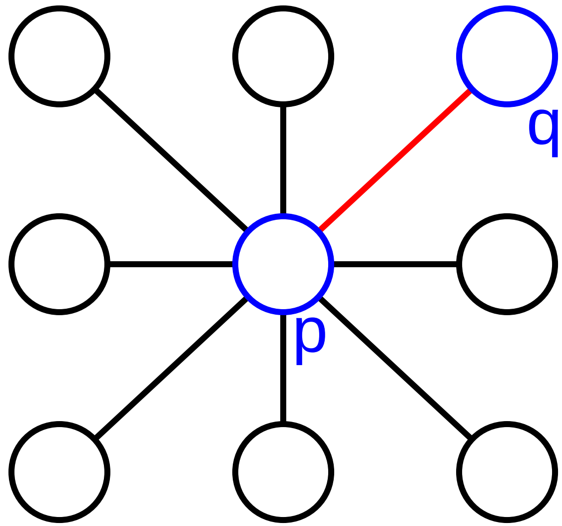






Let's look at this more closely

Treat the image as a graph



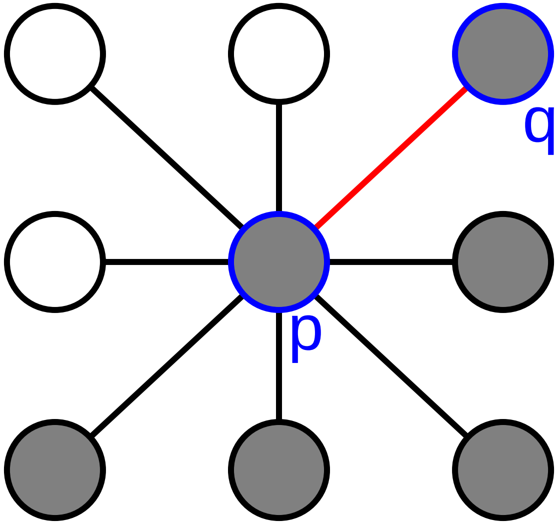
Graph

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

Note: each *link* has a cost

- different than the figure before where each pixel had a cost

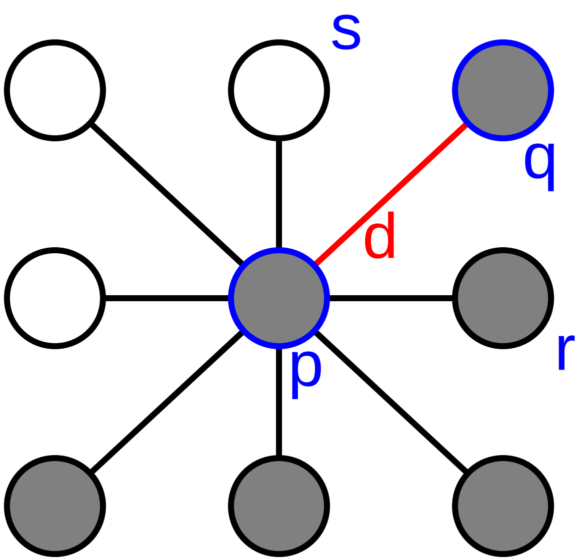
Defining the costs for shortest paths



Want to hug image edges: how to define cost of a link?

- the link should follow the intensity edge

Defining the costs for shortest paths

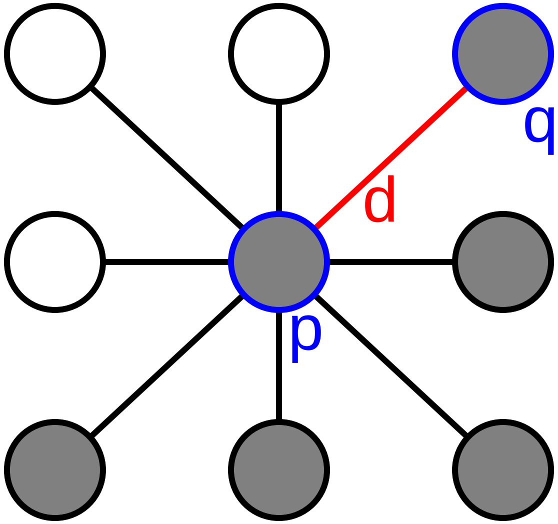


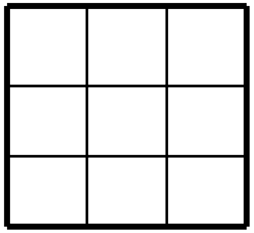
Want to hug image edges: how to define cost of a link?

- the link should follow the intensity edge
  - want intensity to change rapidly perpendicular to the link

- Define d = $\frac{1}{\sqrt{2}}$ |intensity of s – intensity of r|

Defining the costs for shortest paths





H_d

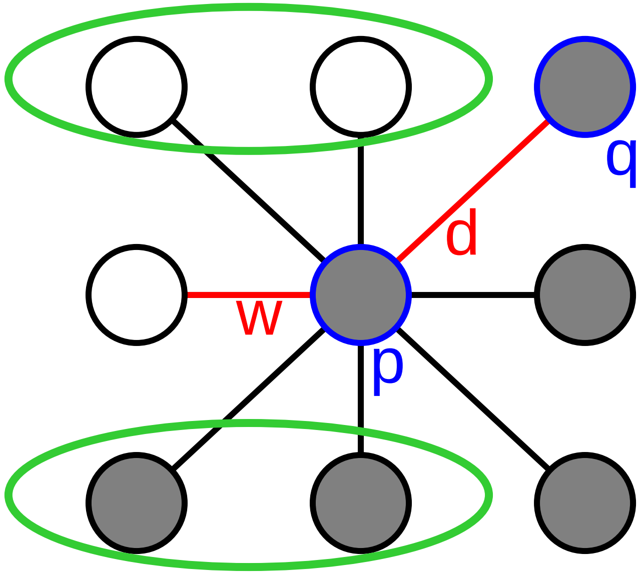
d can be computed using a cross-correlation filter

- assume it is centered at p

Defining the costs for shortest paths

$$\frac{1}{4}\begin{bmatrix} 1 & 1 & \\ & & \\ -1 & -1 & \end{bmatrix}$$

H_w



$$\frac{1}{\sqrt{2}}\begin{bmatrix} & 1 & \\ & & -1 \\ & & \end{bmatrix}$$

H_d

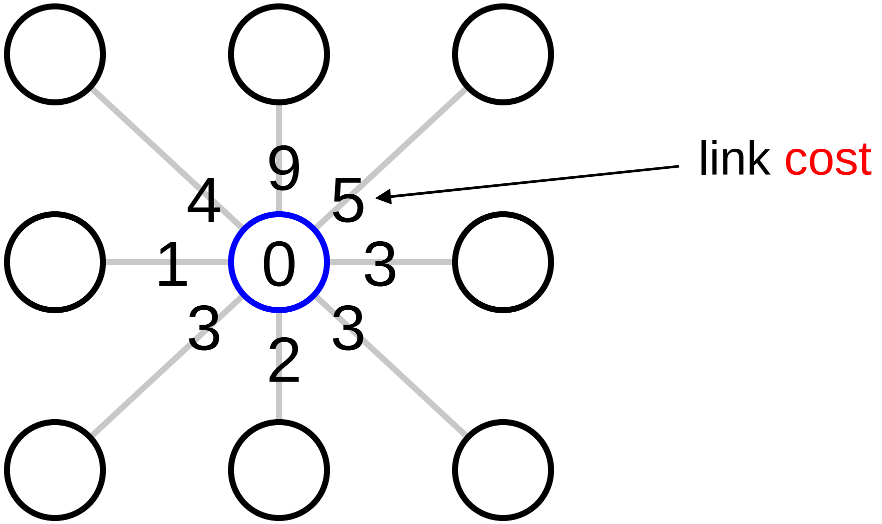
d can be computed using a cross-correlation filter

- assume it is centered at p

Cost of a link

- Want edges to have minimum cost, so define link cost as:
  - cost = (max-|filter response|)*length
  - where max = maximum |filter response| over all pixels in the image
- Note that cost is scaled by length of link. Why?

Dijkstra's shortest path algorithm



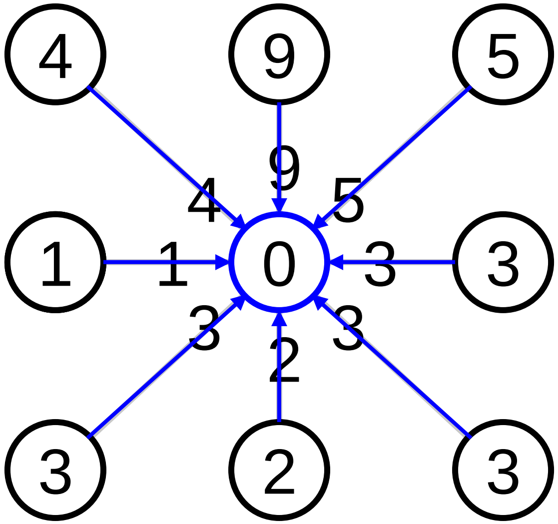
Algorithm

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) + cost_{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

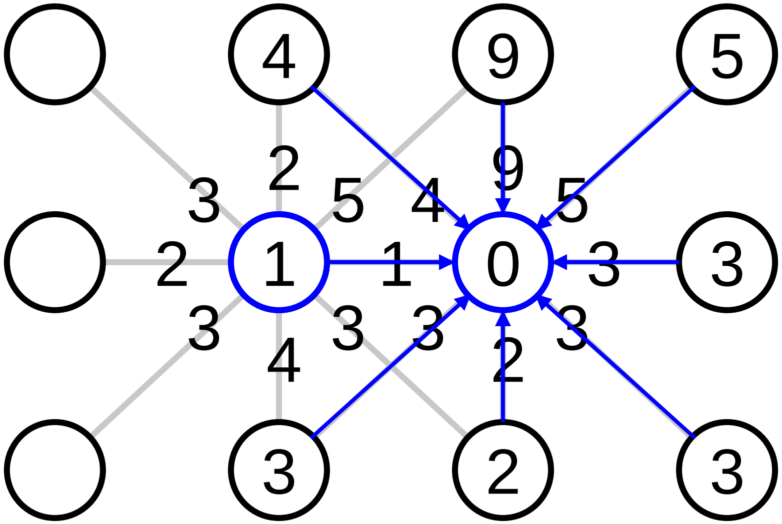
   » set cost(q) = min(cost(p) + $cost_{pq}$, cost(q))

      » if q's cost changed, make q point back to p

   » insert q on the ACTIVE list (if not already there)

3. set r = node with minimum cost on the ACTIVE list

Dijkstra's shortest path algorithm



Algorithm

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

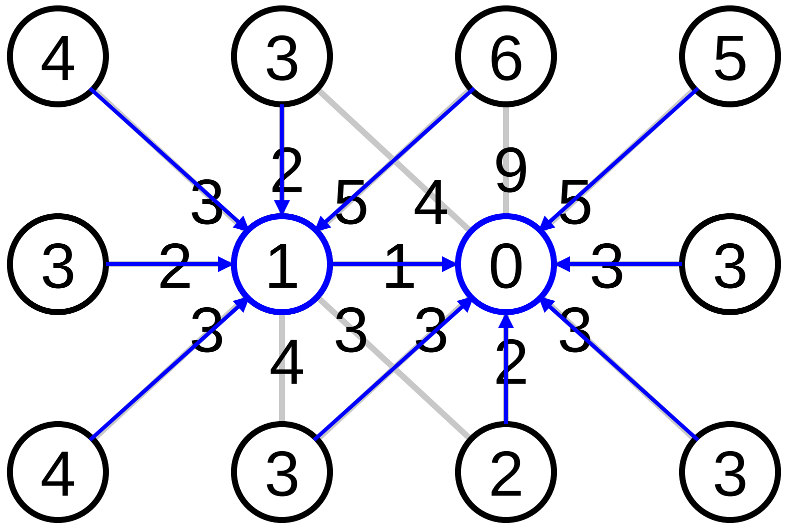
   for each of p's neighbors q that are not expanded

   » set cost(q) = min(cost(p) + $cost_{pq}$, cost(q))

   » if q's cost changed, make q point back to p

   » insert 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) + $cost_{pq}$, cost(q))

   » if q's cost changed, make q point back to p

   » insert 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

Analysis of Dijkstra's Algorithm

Suppose the image contains N pixels

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) + $cost_{pq}$, cost(q)) ← O(N) total

   » if q's cost changed, make q point back to p

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

N times

Total time = $N(O(N)) + O(N) = O(N^2)$

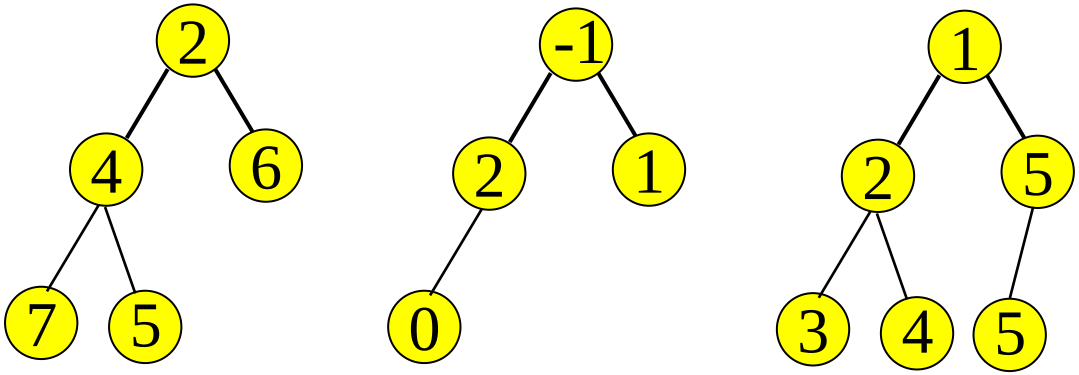
Quadratic! Can we do better?

Recall from Data Structures: Priority Queue (Heap)

A binary heap is a binary tree that is:

1. Complete: the tree is completely filled except possibly the bottom level, which is filled from left to right
2. Satisfies the heap order property: every node is smaller than (or equal to) its children

Therefore, the root node is always the smallest in a heap

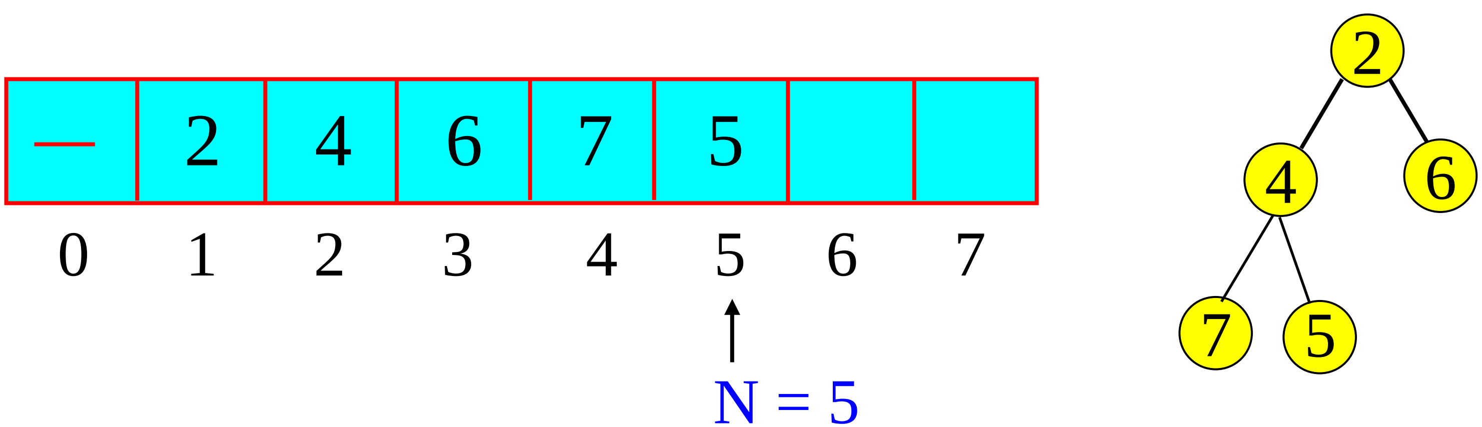


Which of these is not a heap?

Array Implementation of Priority Queues

Array Implementation:

- Root node = A[1]
- Children of A[i] = A[2i], A[2i + 1]
- Keep track of current size N (number of nodes)



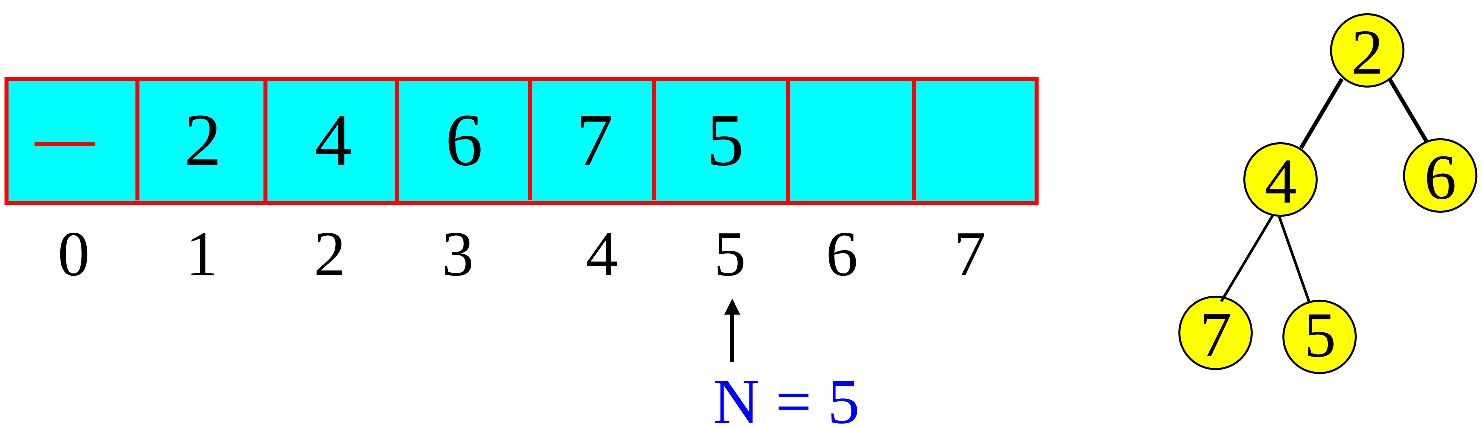
Priority Queue Operations

ExtractMin: return the element with the minimum cost from a priority queue: O(1) time

Insert: insert an element into a priority queue: O(log N)

Update: update an existing element in a priority queue: O(log N)

IsEmpty: return true if a priority queue is empty: O(1)



Dijkstra's Algorithm and Priority Queues

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) + $cost_{pq}$, cost(q)) ← Update

      » if q's cost changed, make q point back to p

   » insert q on the ACTIVE list (if not already there) ← Insert

3. set r = node with minimum cost on the ACTIVE list ← ExtractMin
4. repeat Step 2 for p = r

Dijkstra's Algorithm with a Priority Queue

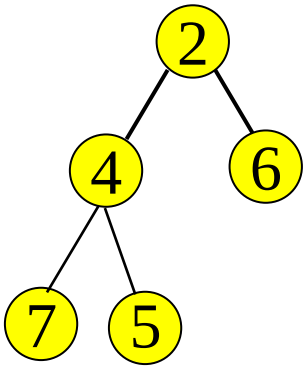
Use a priority queue to store active nodes with key = cost

N times:

Select the active node *r* with the lowest cost → ExtractMin

N times:

cost(q) = min(cost(p) + cost_{pq}, cost(q)) → Update



Total run time = ?

Dijkstra's Algorithm with a Priority Queue

Use a priority queue to store active nodes with key = cost

N times:

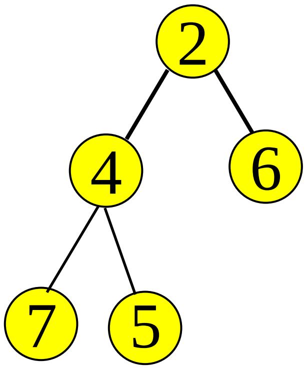
Select the active node *r* with the *lowest cost*

ExtractMin

N times:

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

Update



Total run time = O(N log N + N log N) = O(N log N)

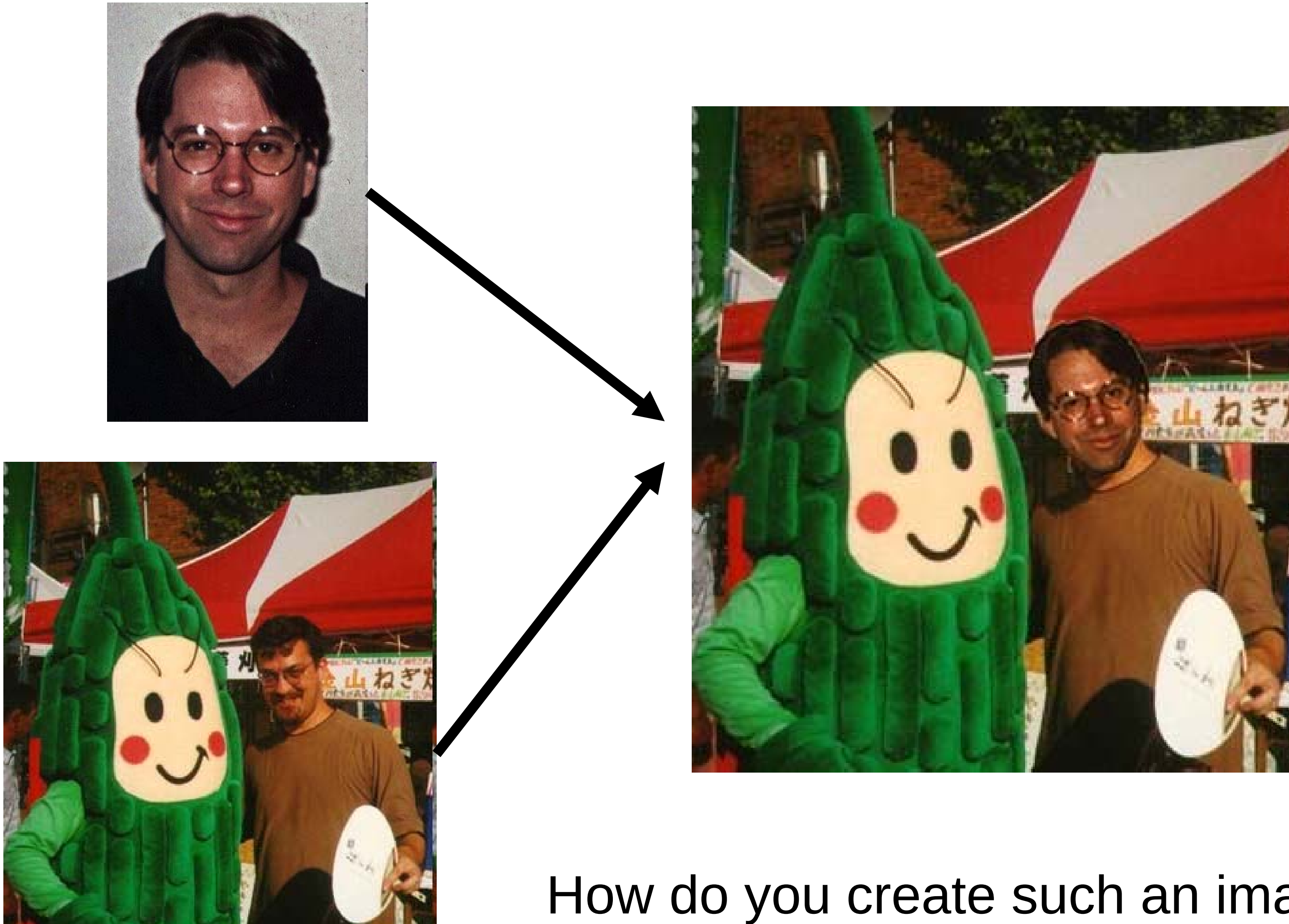
Better than Quadratic!

Summary: Dijkstra's shortest path algorithm

Properties

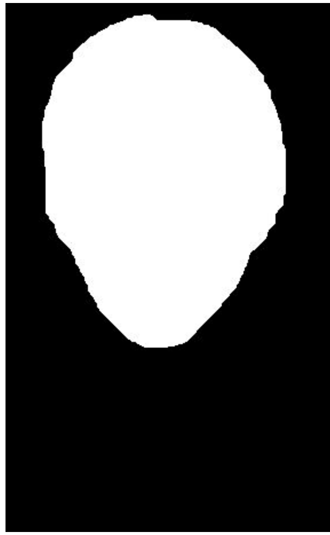
- Computes the minimum cost path from the seed to every node in the graph. Set of minimum cost paths forms a *tree*
- Running time with N pixels:
  - $O(N^2)$ time if you use an active list
  - $O(N \log N)$ if you use an active priority queue (heap)
  - Takes fraction of a second for a typical (640x480) image
- Once the tree is computed once, can extract optimal path from any point to seed in $O(N)$ time.
  - Runs in real time as the mouse moves
- What happens if the user specifies a new seed?

Creating Composite Images using Scissors



How do you create such an image?

Using Image Scissors to extract an Object

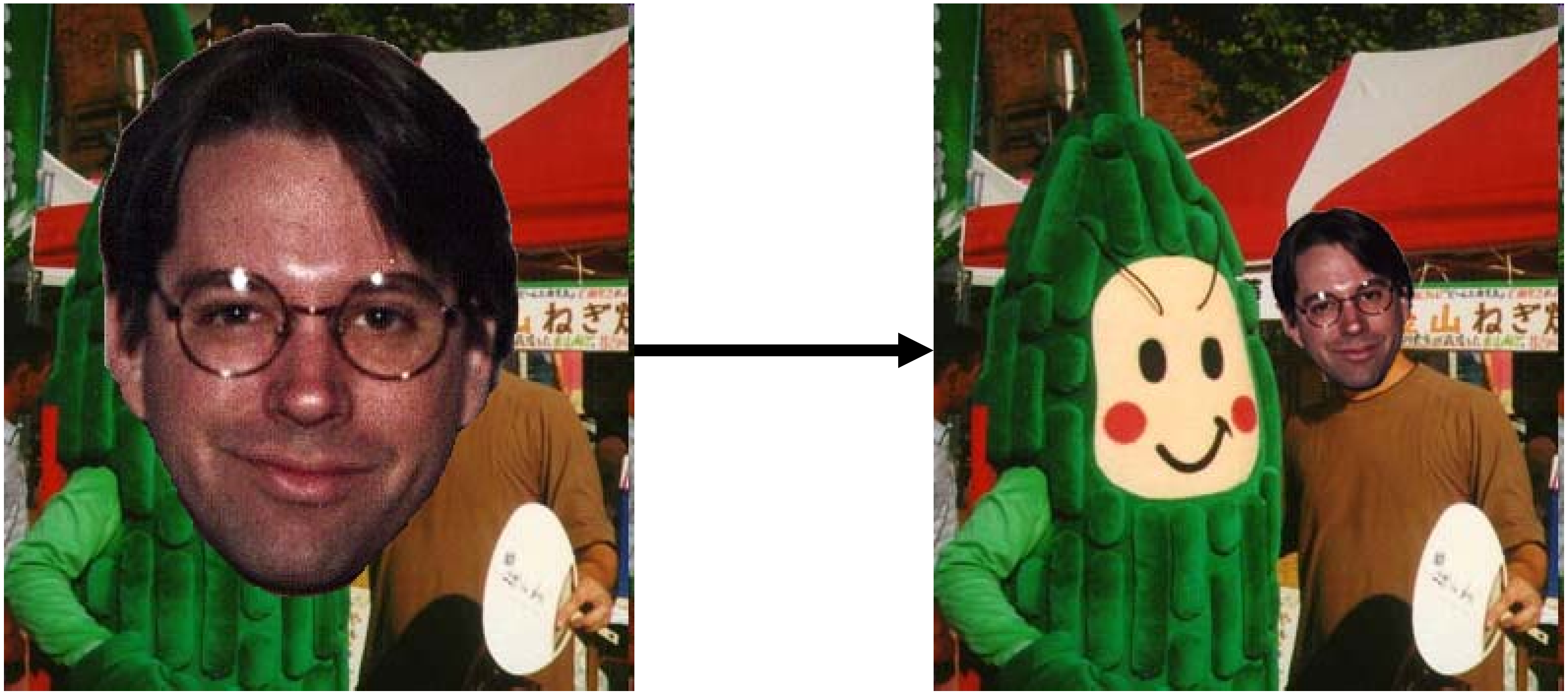


Mask from Image Scissors

Composite image using Photoshop

Shape transformation in Photoshop

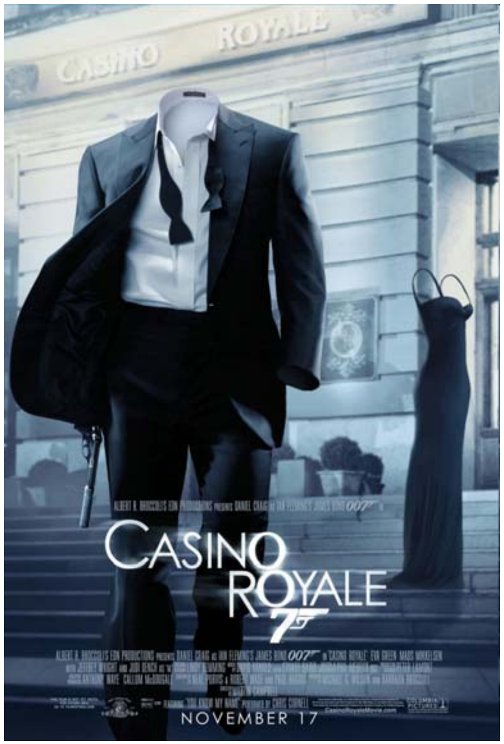
Rotate and scale



Color matching in Photoshop

Adjust color balance

Other Examples (from past CSE 455)



Your masterpiece here

Next Time: Cameras and Image Formation

- Things to do:
  - Project 1 will be assigned today (on web)
    - Use Sieg 327 if possible – all required software is installed on computers there
    - Contact Jiun-Hung if you have questions
    - Start early!
  - Read Chap. 2 in text

Have a great weekend!