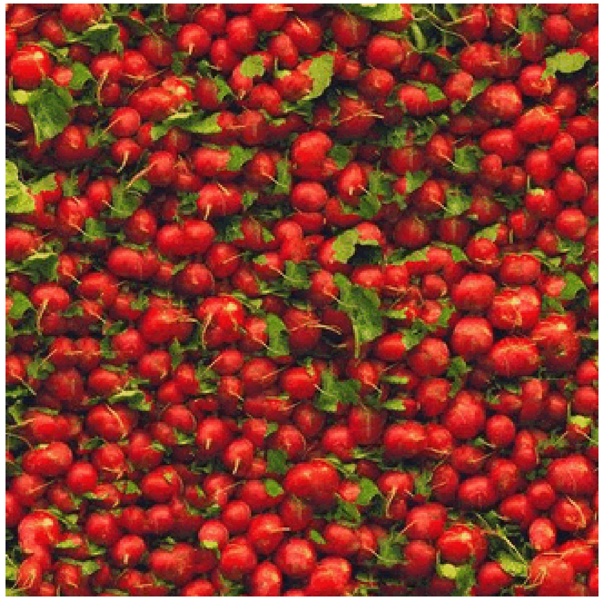
Cutting Images: Graphs and Boundary Finding

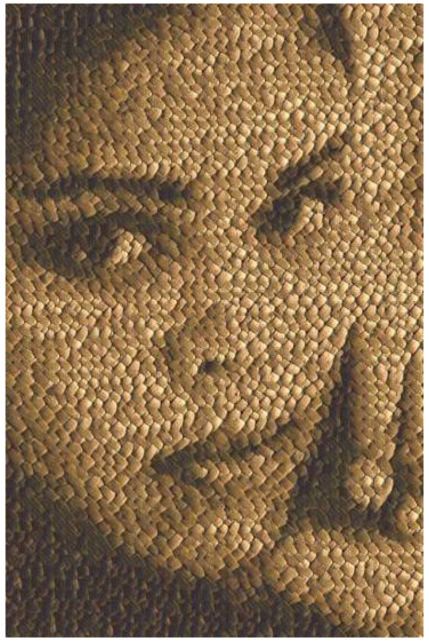
“The Double Secret”, Magritte

Computational Photography

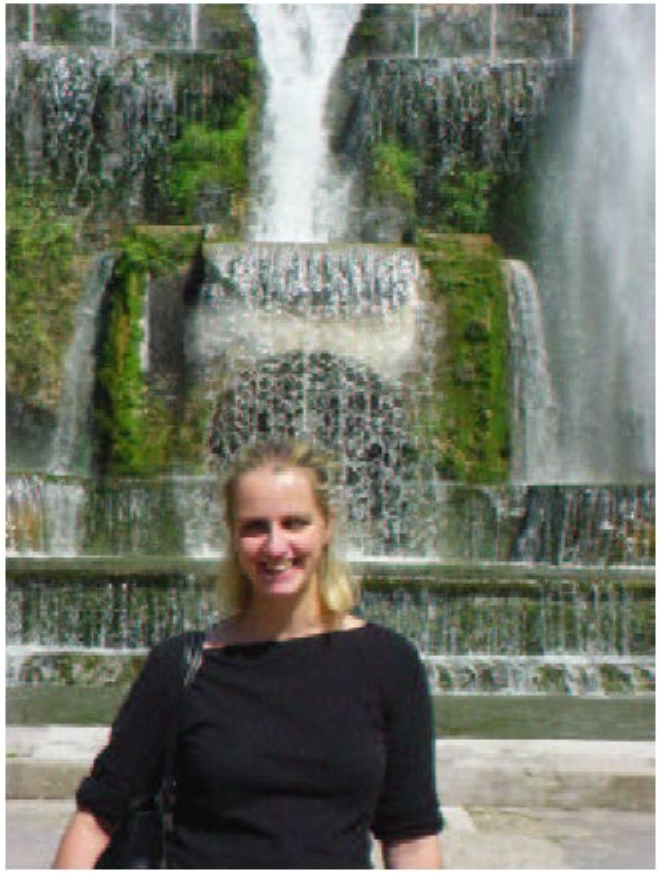
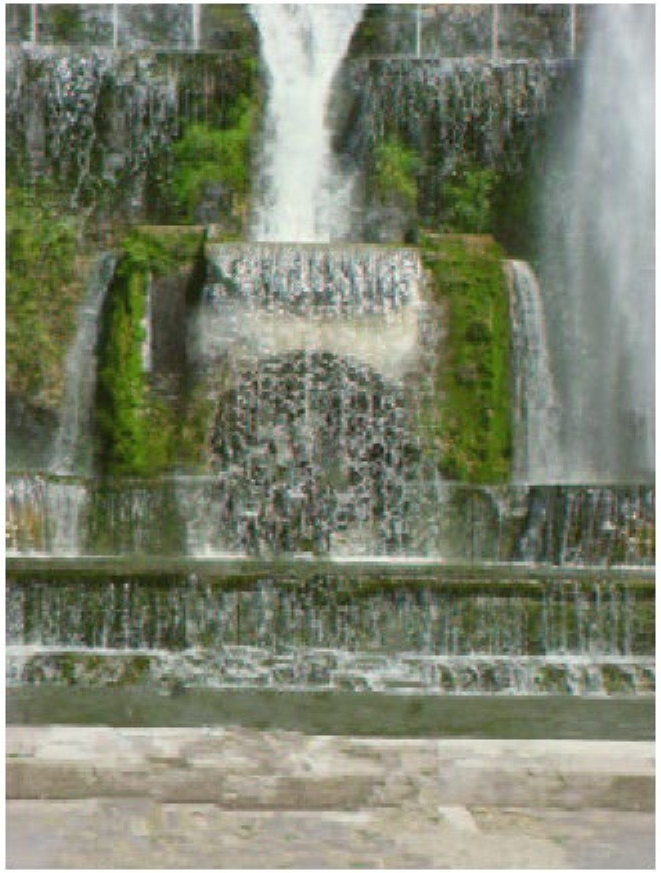
Derek Hoiem, University of Illinois

Last class: texture synthesis and transfer



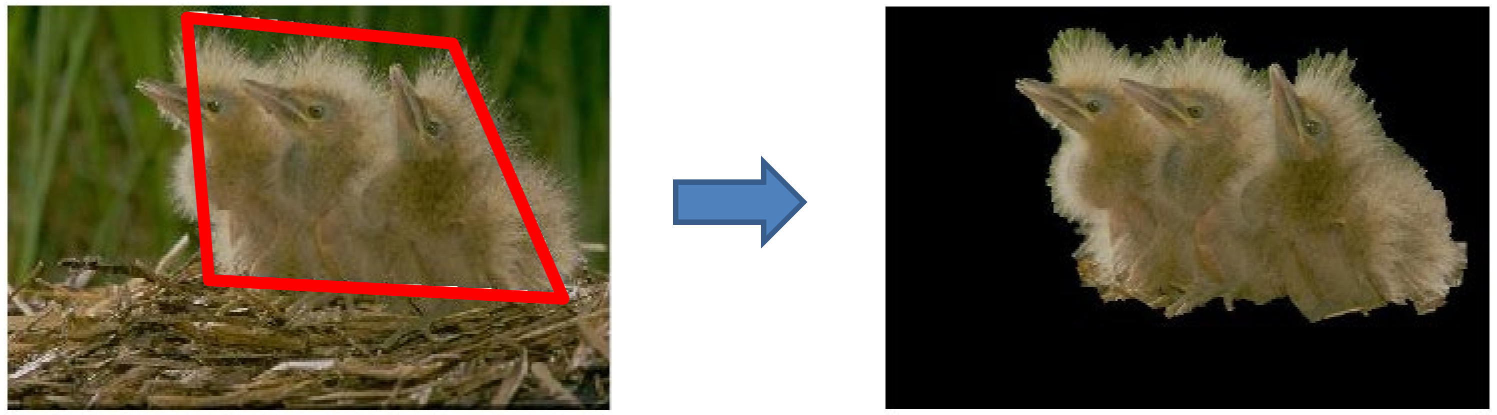


Last class: in-painting

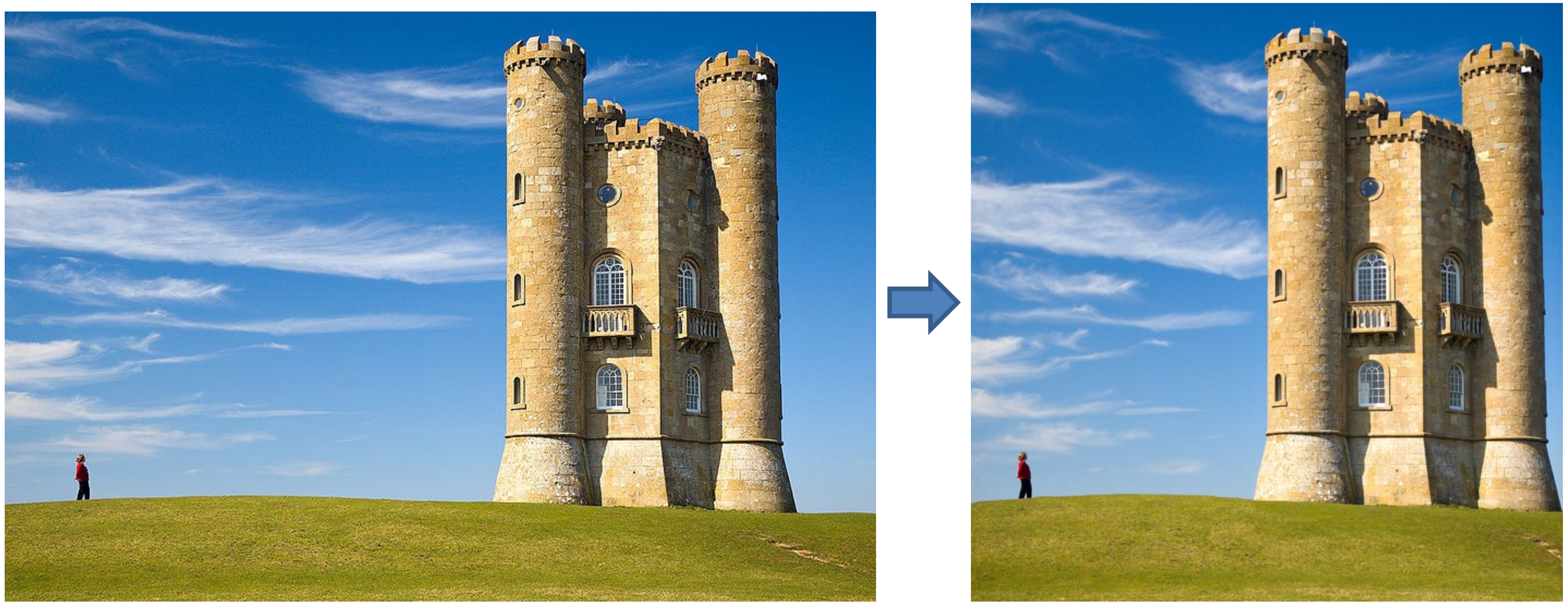
This Lecture: Finding Seams and Boundaries

Segmentation



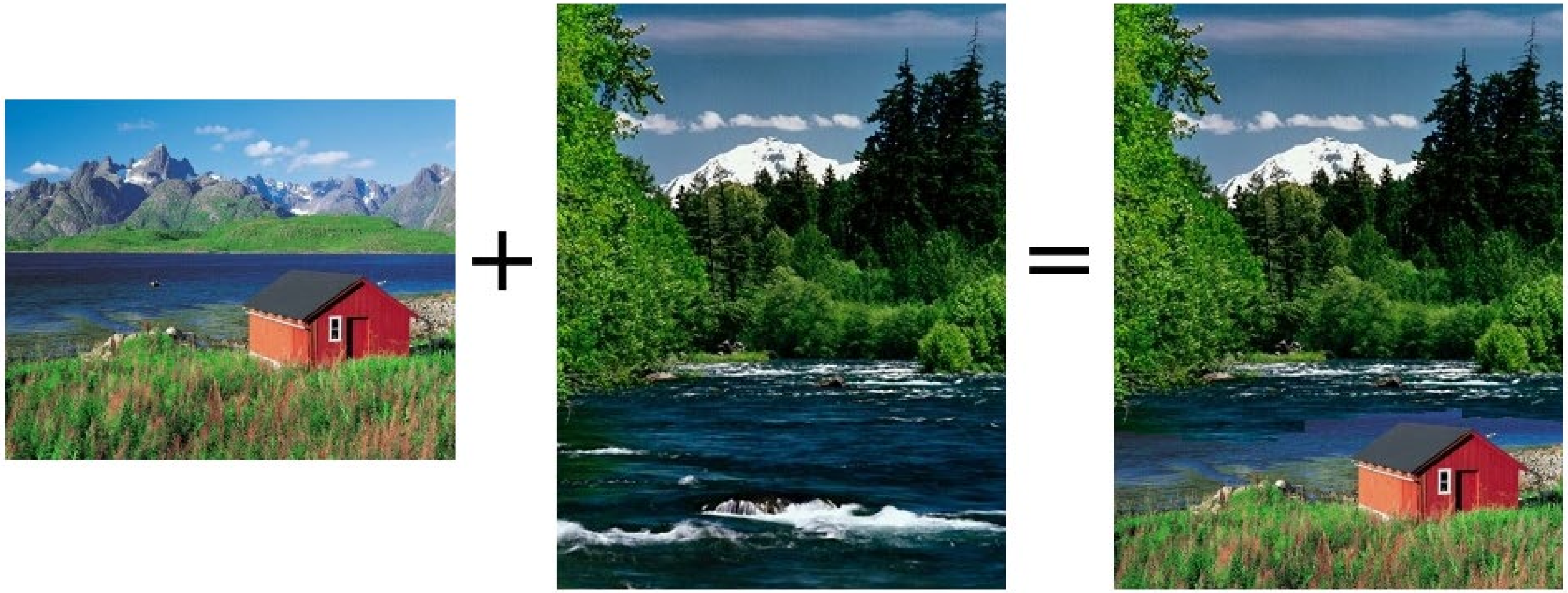
This Lecture: Finding Seams and Boundaries

Retargeting



This Lecture: Finding Seams and Boundaries

Stitching



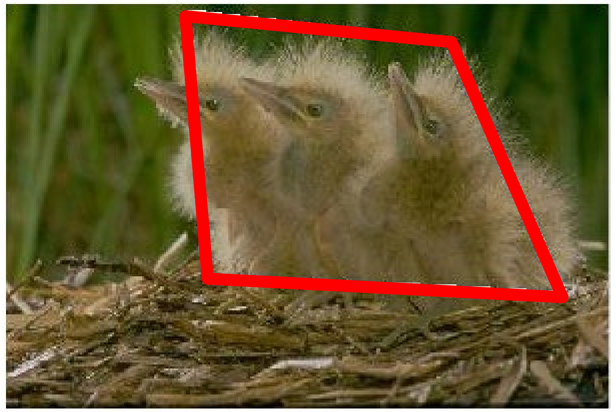
This Lecture: Finding seams and boundaries

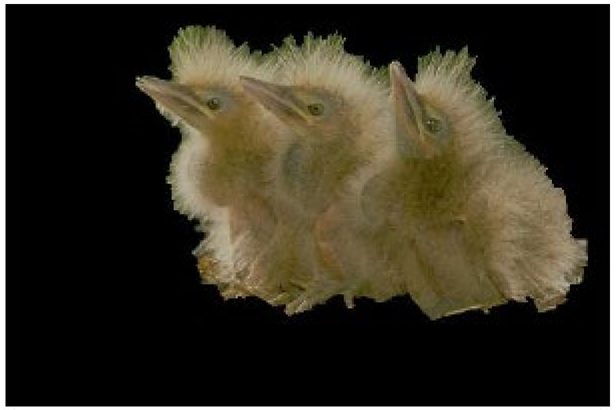
Fundamental Concept: The Image as a Graph

- Intelligent Scissors: Good boundary = short path
- Graph cuts: Good region has low cutting cost

Semi-automated segmentation

User provides imprecise and incomplete specification of region – your algorithm has to read his/her mind.





Key problems

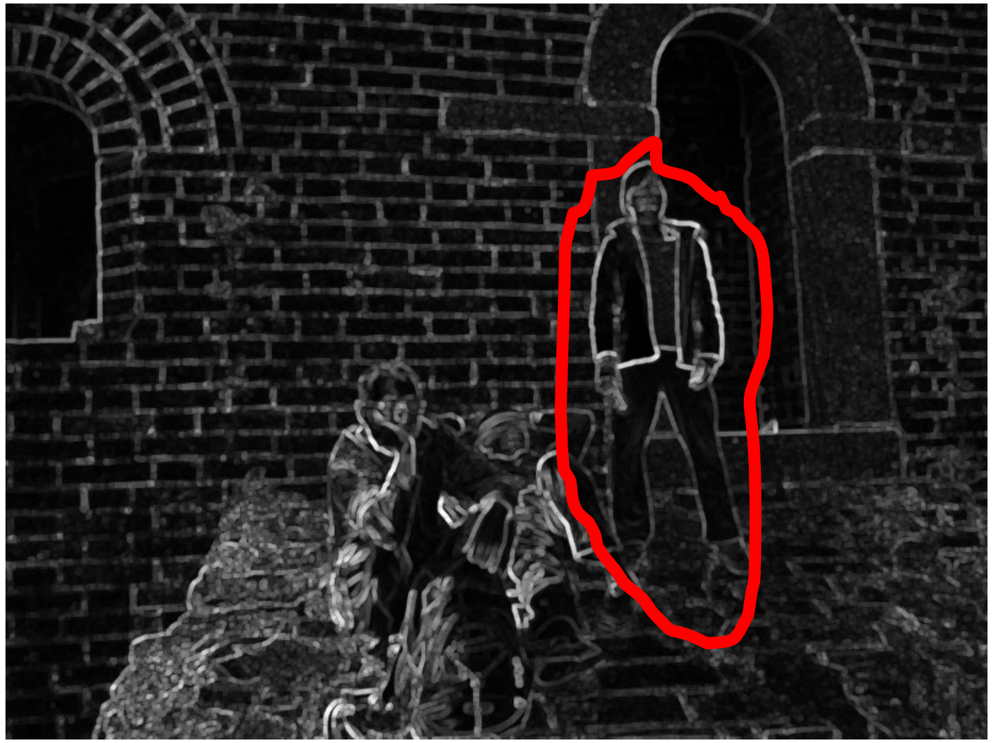
1. What groups of pixels form cohesive regions?
2. What pixels are likely to be on the boundary of regions?
3. Which region is the user trying to select?

What makes a good region?

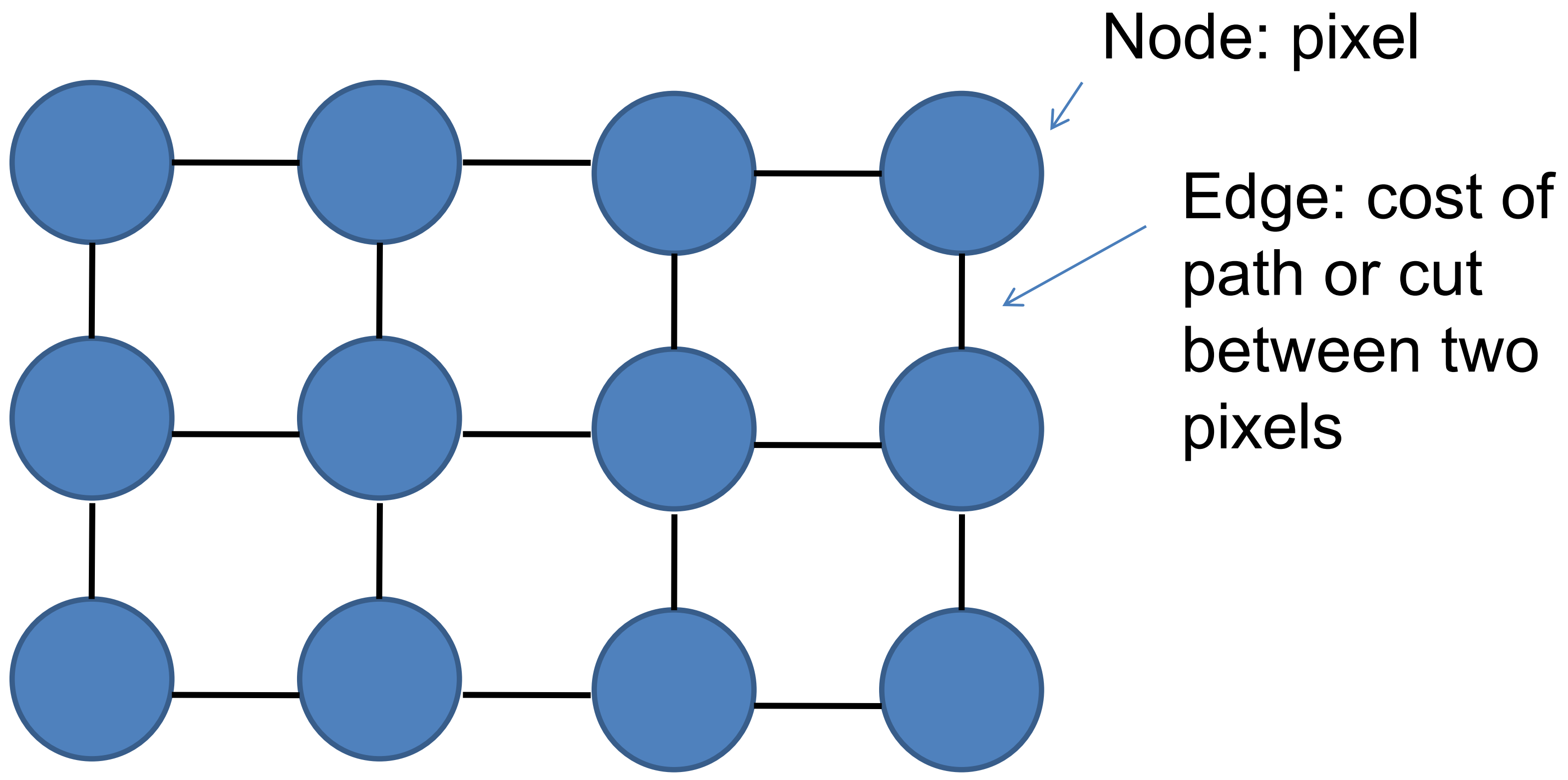
- Contains small range of color/texture
- Looks different than background
- Compact

What makes a good boundary?

- High gradient along boundary
- Gradient in right direction
- Smooth

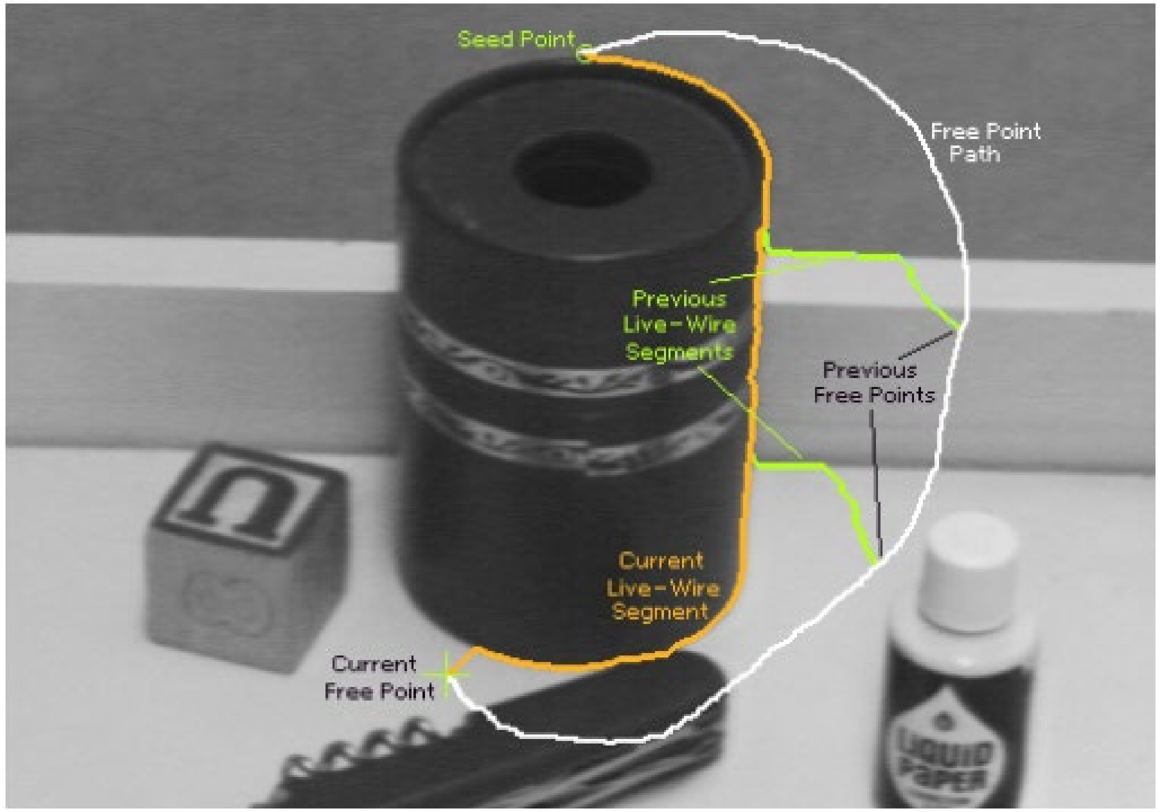


The Image as a Graph



Intelligent Scissors

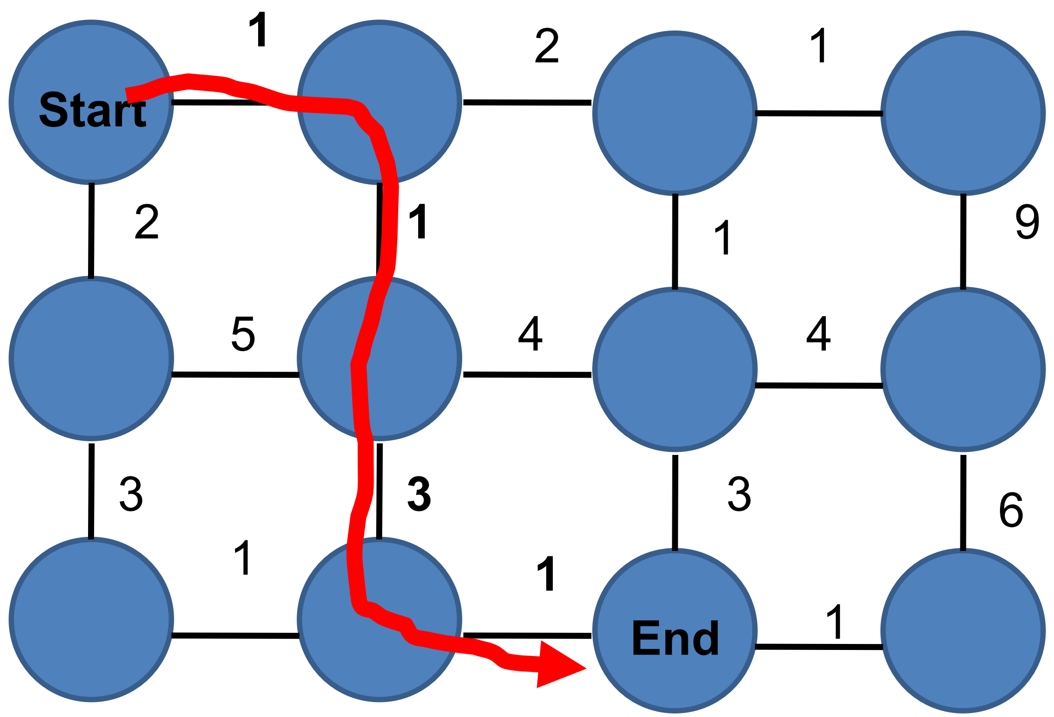
Mortenson and Barrett (SIGGRAPH 1995)



Intelligent Scissors

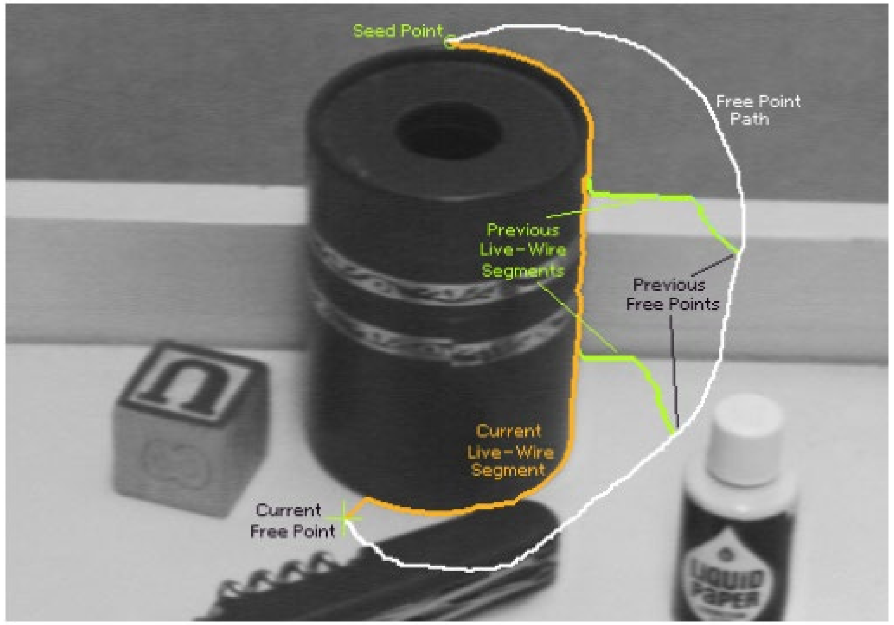
Mortenson and Barrett (SIGGRAPH 1995)

A good image boundary has a short path through the graph.



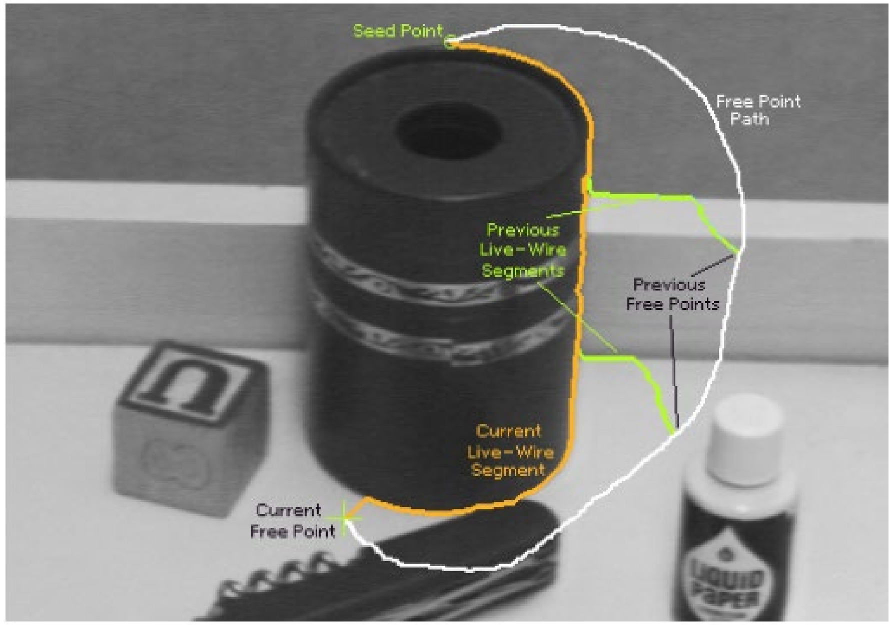
Intelligent Scissors

- Formulation: find good boundary between seed points
- Challenges
  - Minimize interaction time
  - Define what makes a good boundary
  - Efficiently find it



Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
4. Get path from seed to cursor, choose new seed, repeat

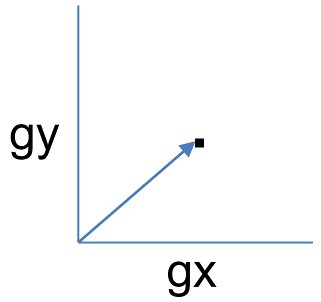


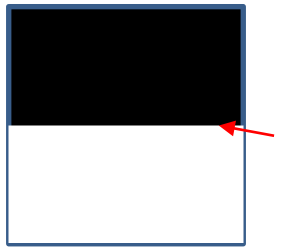
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
   a) Lower if edge is present (e.g., with edge(im, ‘canny’))
   b) Lower if gradient is strong
   c) Lower if gradient is in direction across the boundary

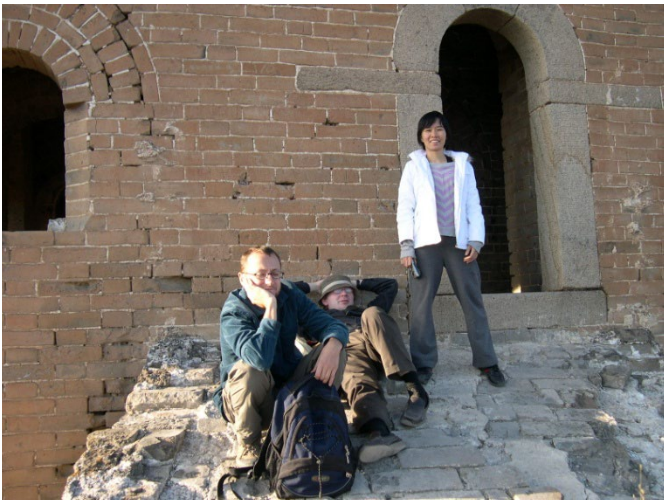
Gradient magnitude and orientation:

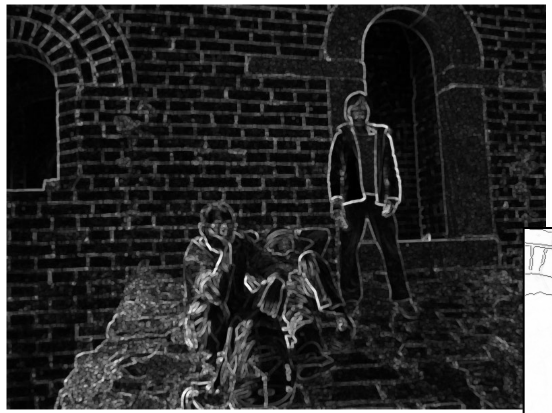
```
gx = filter(im, [-1, 0, 1])
gy = filter(im, [-1, 0, 1].transpose)
mag = sqrt(gx**2 + gy**2)
theta = atan2(-gy, gx)
```



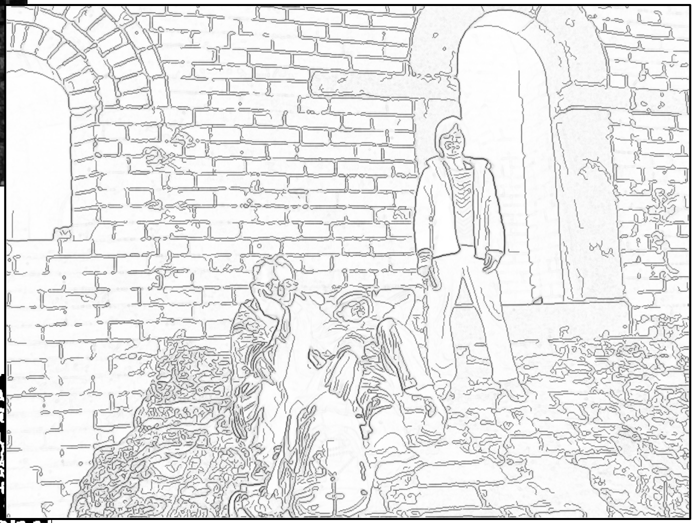


Gradients, Edges, and Path Cost

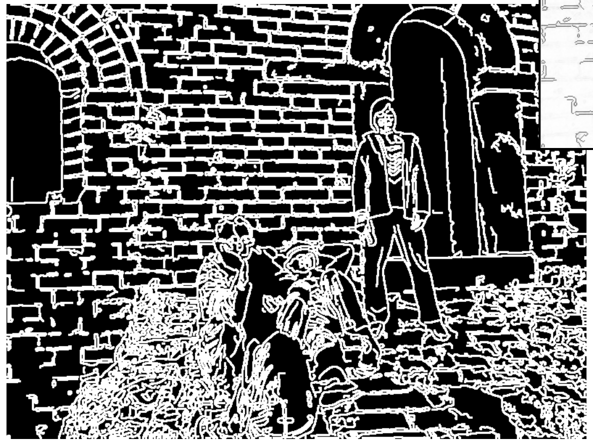




Gradient Magnitude



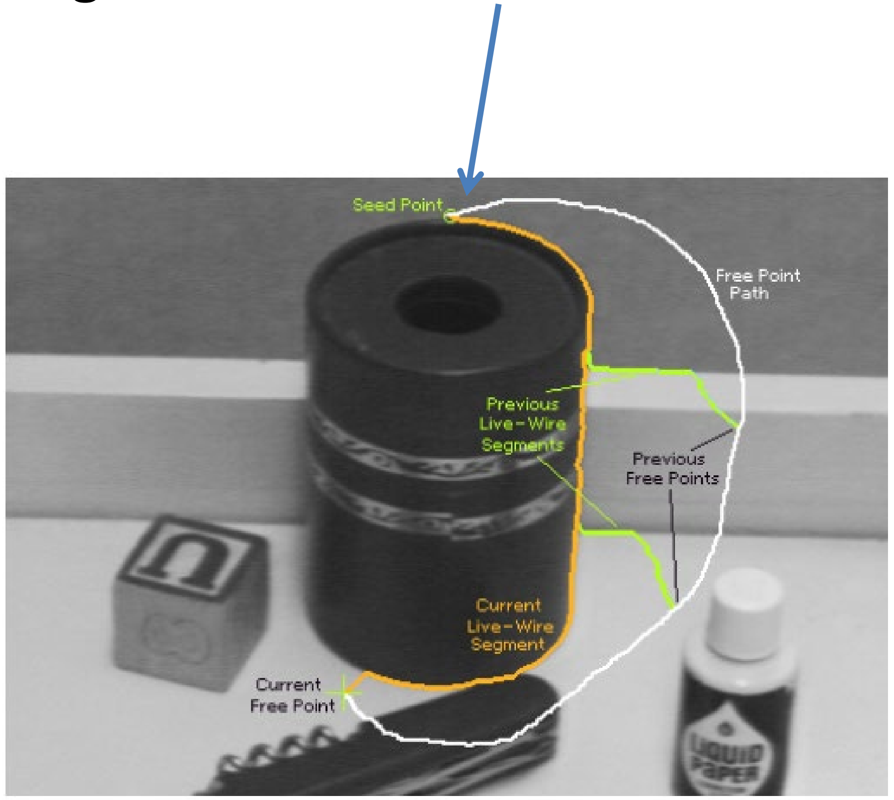
Path Cost



Edge Image

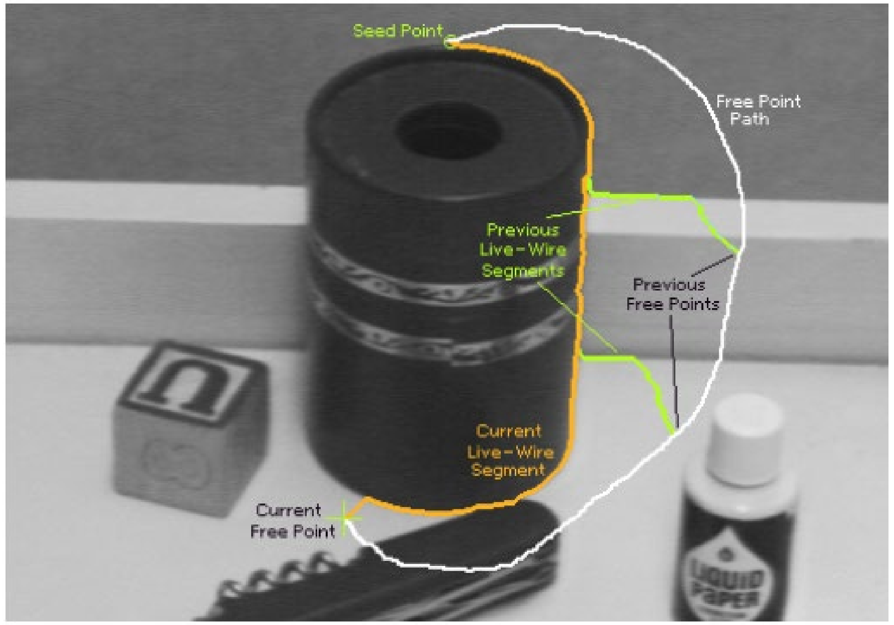
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
   – Snapping



Intelligent Scissors: method

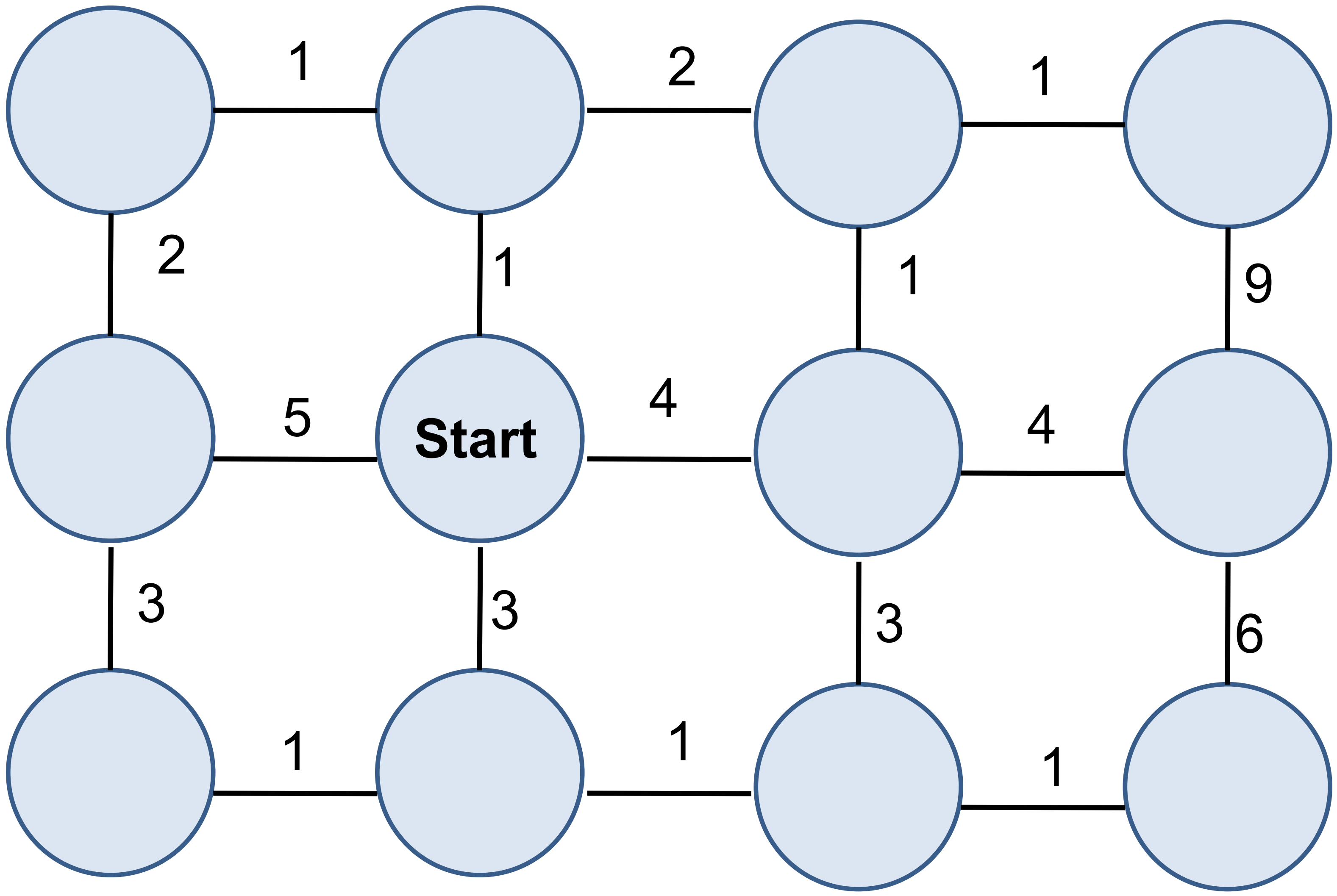
1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
   - Djikstra’s shortest path algorithm



Djikstra's shortest path algorithm

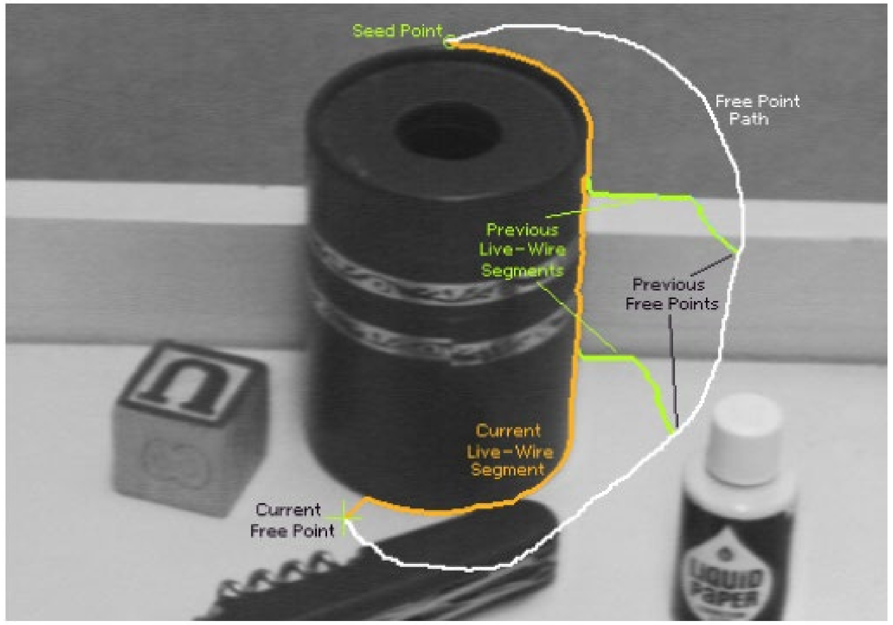
```
Initialize, given seed $s$:
• Compute $cost_2(q, r)$ % cost for boundary from pixel
  $q$ to neighboring pixel $r$
• cost($s$) = 0  %  total cost from seed to this point
• A = {$s$}  %  set to be expanded
• E = { } % set of expanded pixels
• P(q) % pointer to pixel that leads to $q$

Loop while A is not empty
1. $q$ = pixel in A with lowest cost
2. Add q to E
3. for each pixel $r$ in neighborhood of $q$ that is not
   in E
   a) cost_tmp = cost($q$) + $cost_2(q,r)$
   b) if ($r$ is not in A) OR (cost_tmp < cost($r$))
       i. cost($r$) = cost_tmp
       ii. P($r$) = $q$
       iii.Add $r$ to A
```

Example of Djikstra's algorithm



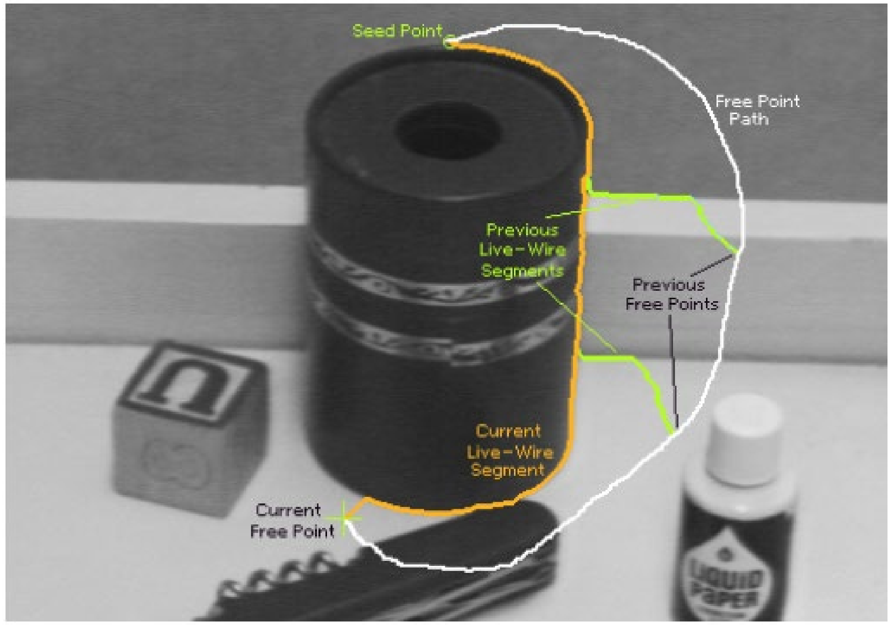
Intelligent Scissors: method

1. Define boundary cost between neighboring pixels
2. User specifies a starting point (seed)
3. Compute lowest cost from seed to each other pixel
4. Get new seed, get path between seeds, repeat

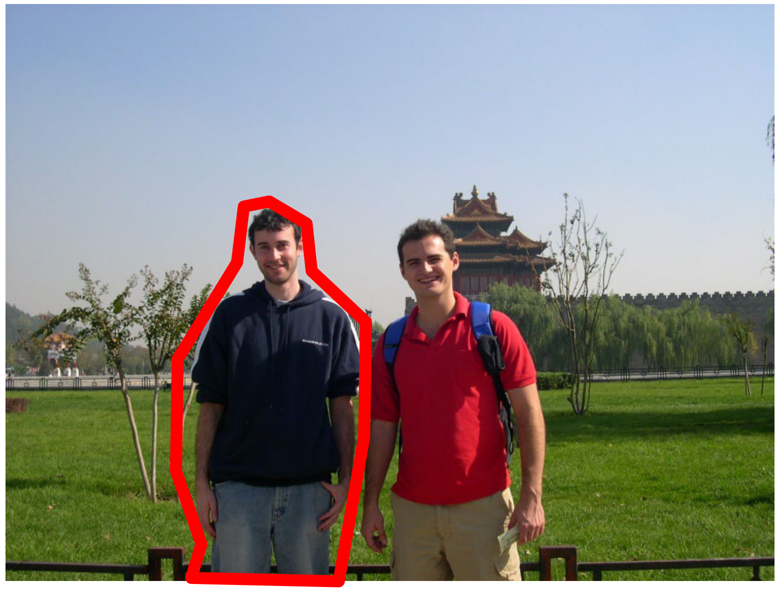


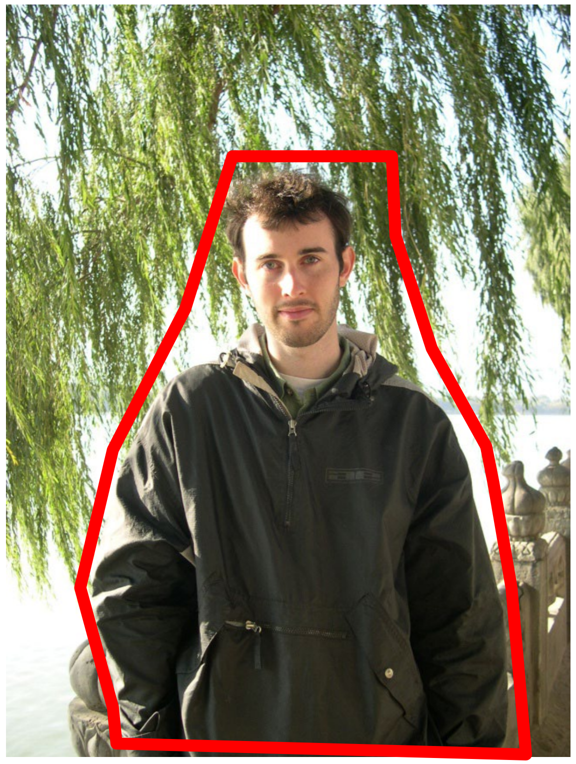
Intelligent Scissors: improving interaction

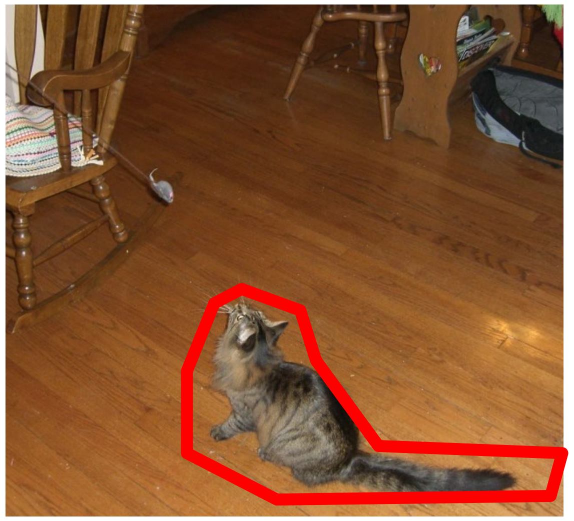
1. Snap to low-cost pixel within small window around cursor when placing first seed
2. Automatically adjust to boundary as user drags
3. Freeze stable boundary points to make new seeds



Where will intelligent scissors work well, or have problems?







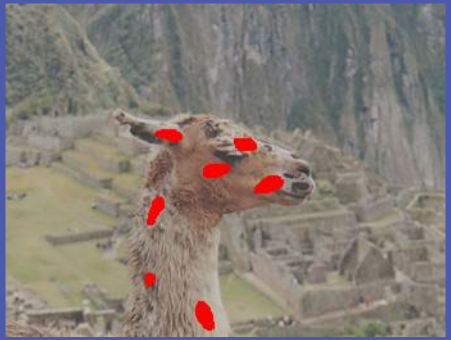
Grab cuts and graph cuts

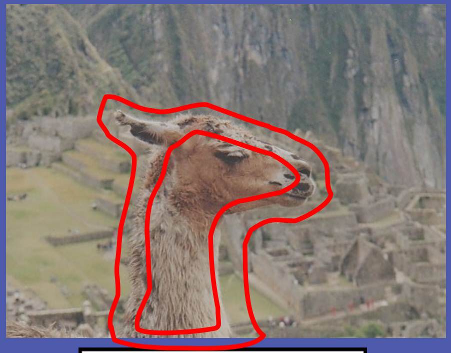
Magic Wand
(198?)

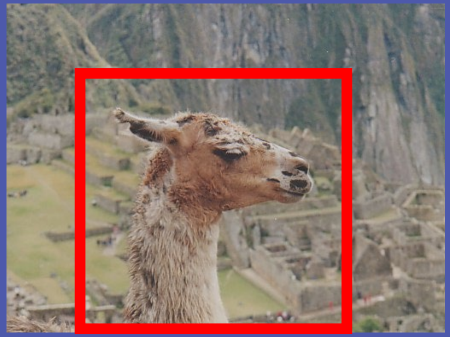
Intelligent Scissors
Mortensen and Barrett (1995)

GrabCut
Rother et al. (2004)

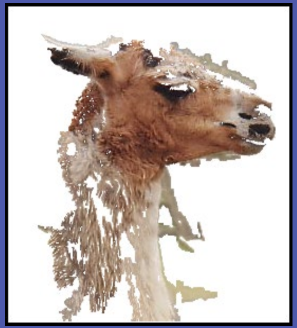
User
Input

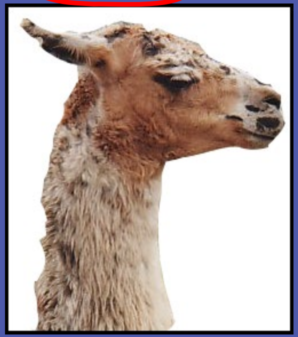


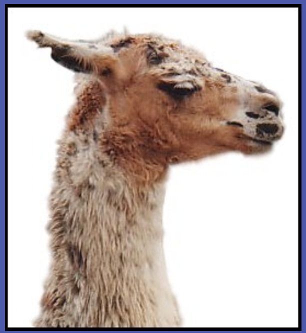




Result







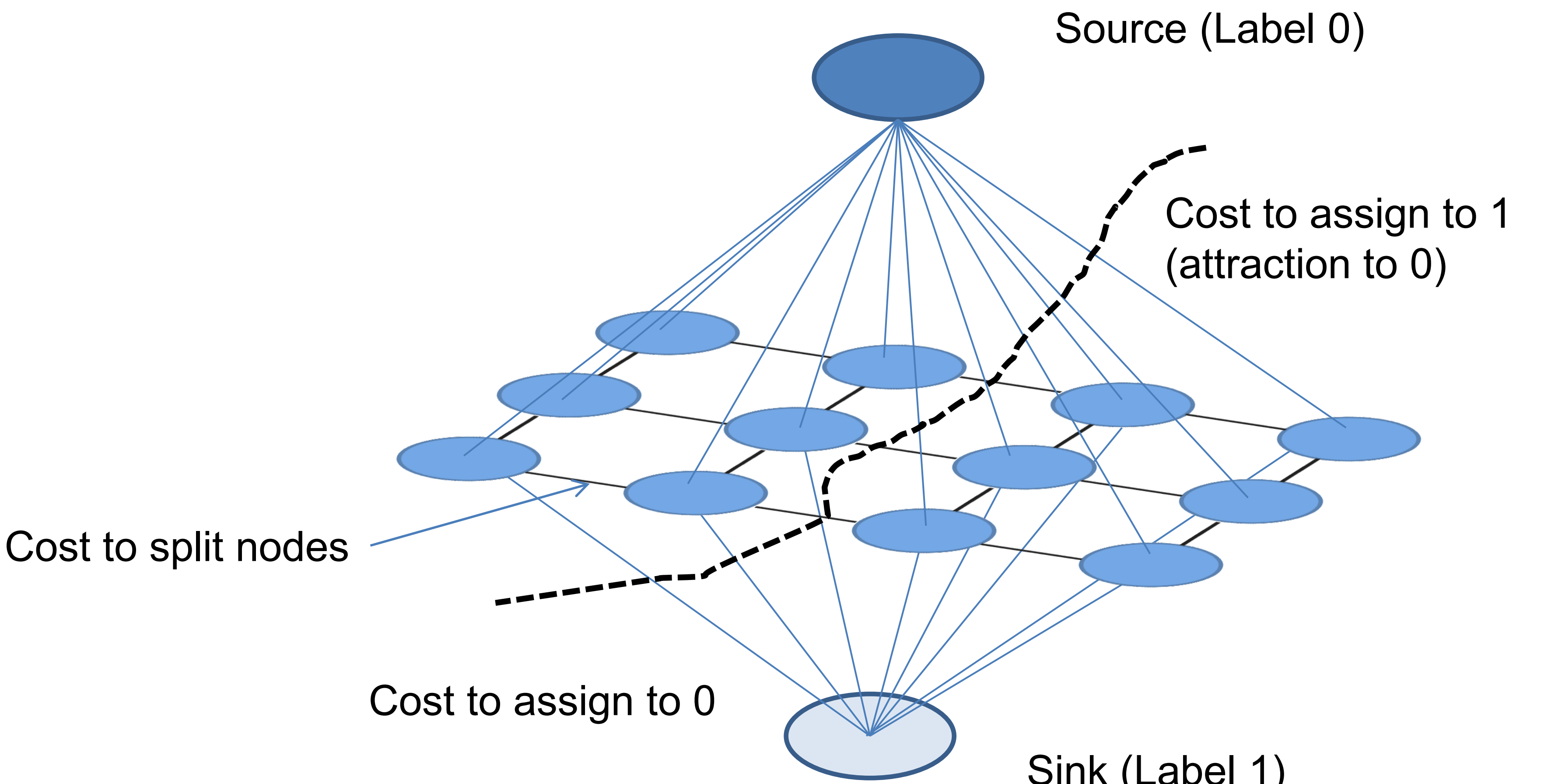
Regions

Boundary

Regions & Boundary

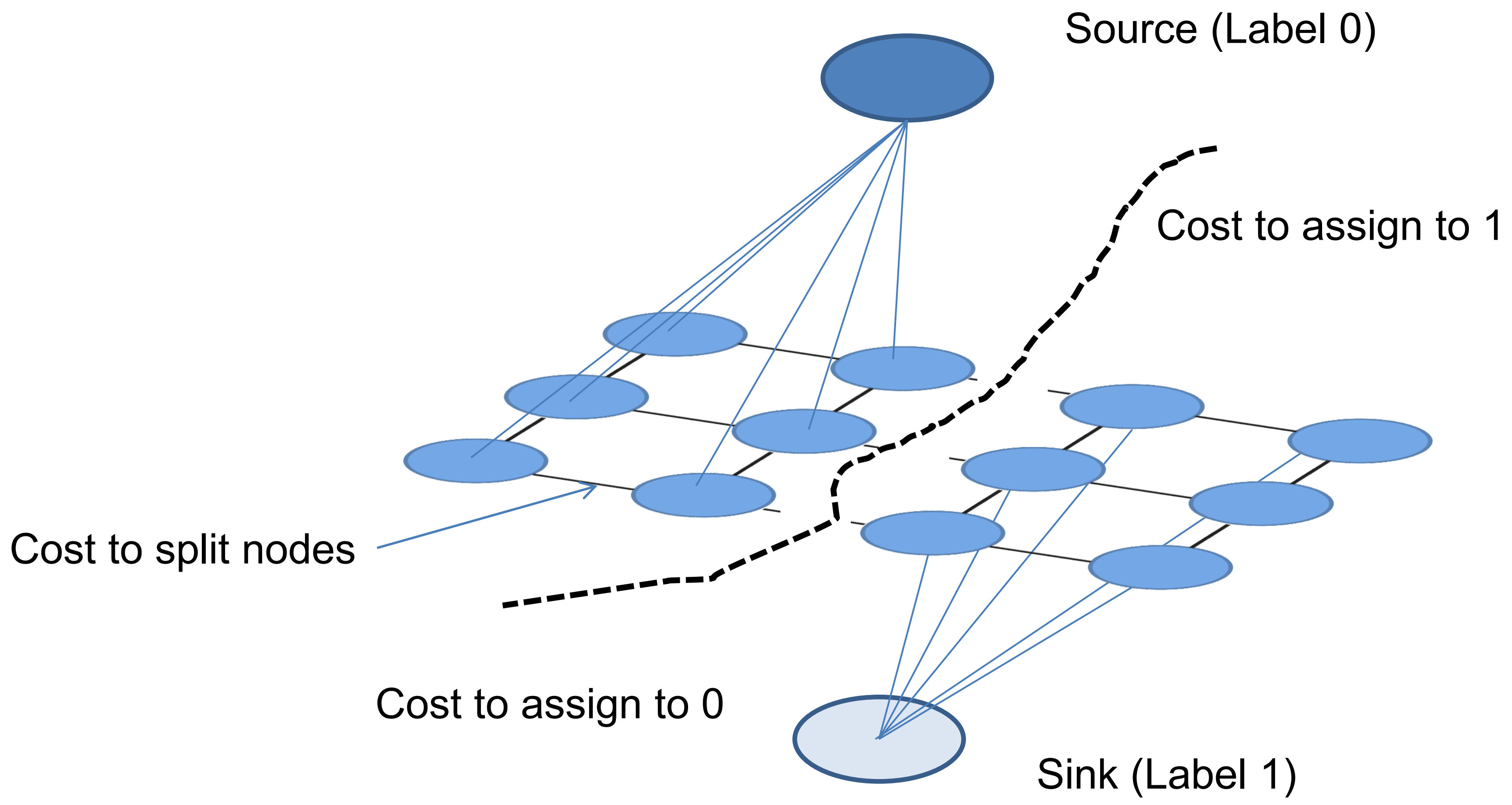
Source: Rother

Segmentation with graph cuts



$$Energy(\mathbf{y};\theta,data) = \sum_i \psi_1(y_i;\theta,data) \sum_{i,j\in edges} \psi_2(y_i,y_j;\theta,data)$$

Segmentation with graph cuts



$$Energy(\mathbf{y};\theta,data)=\sum_{i}\psi_1(y_i;\theta,data)\ \sum_{i,j\in edges}\psi_2(y_i,y_j;\theta,data)$$

Graph cuts segmentation

1. Define graph
   - usually 4-connected or 8-connected
2. Set weights to foreground/background
   - Color histogram or mixture of Gaussians for background and foreground

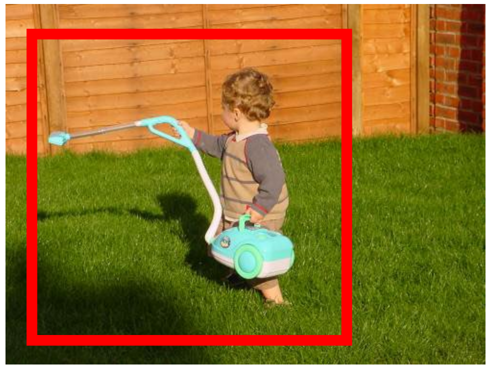
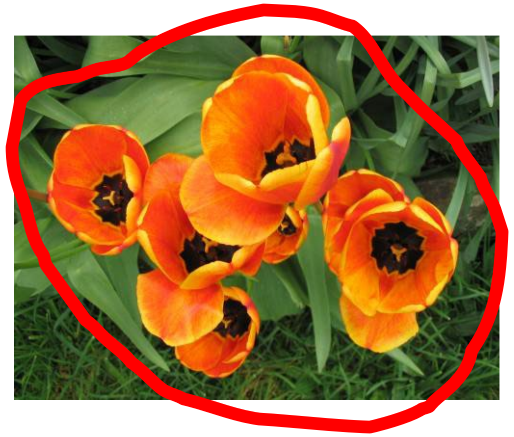
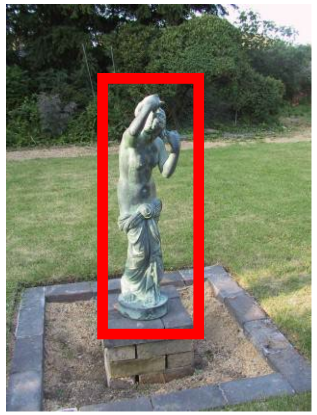
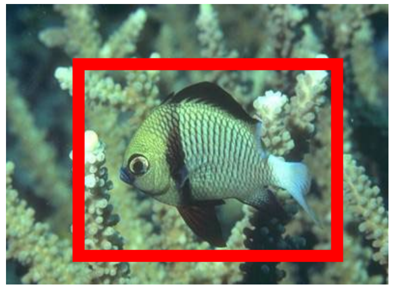
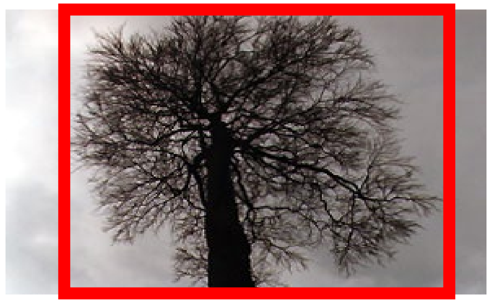
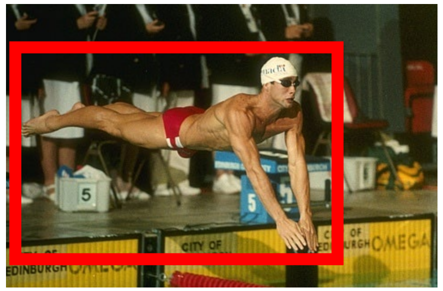
$$unary\_potential(x) = -\log\left(\frac{P(c(x);\theta_{foreground})}{P(c(x);\theta_{background})}\right)$$

3. Set weights for edges between pixels

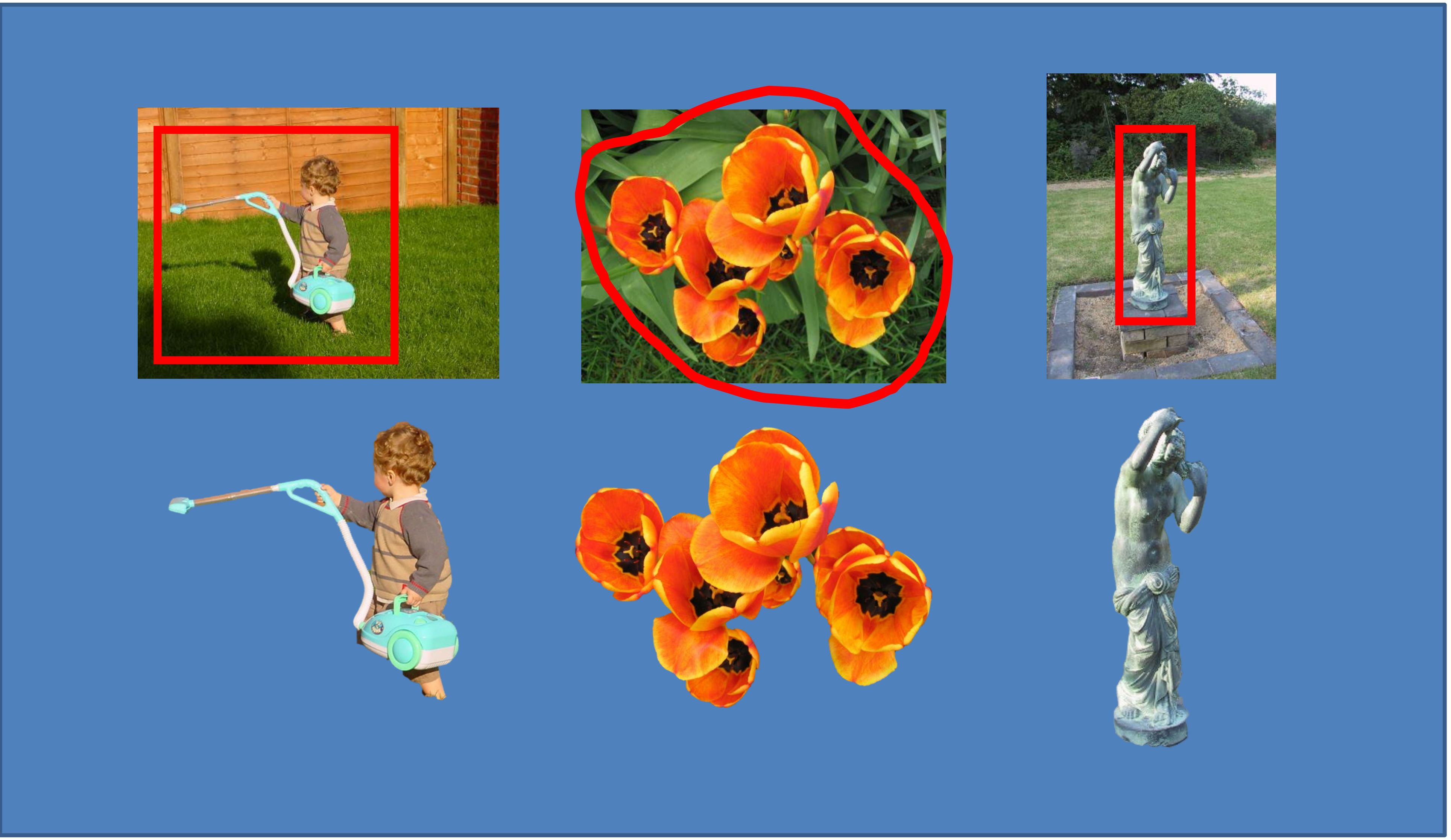
$$edge\_potential(x,y) = k_1 + k_2 \exp\left\{\frac{-\left\|c(x)-c(y)\right\|^2}{2\sigma^2}\right\}$$

4. Apply min-cut/max-flow algorithm
5. Return to 2, using current labels to compute foreground, background models

What is easy or hard about these cases for graphcut-based segmentation?

Easier examples



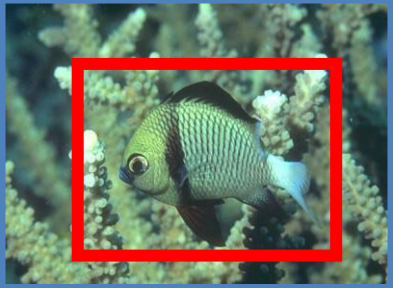
More difficult Examples

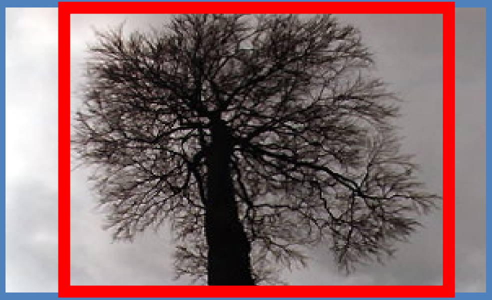
Camouflage & Low Contrast

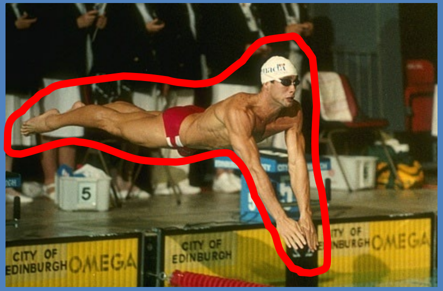
Fine structure

Harder Case

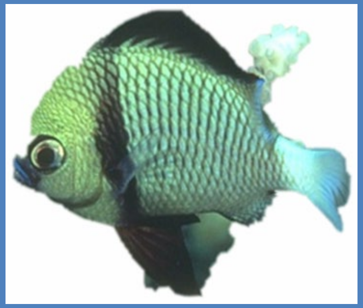
Initial Rectangle

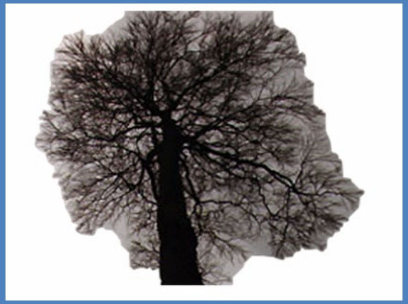


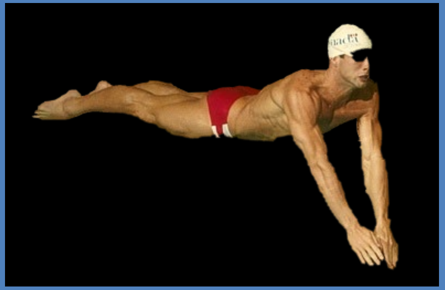




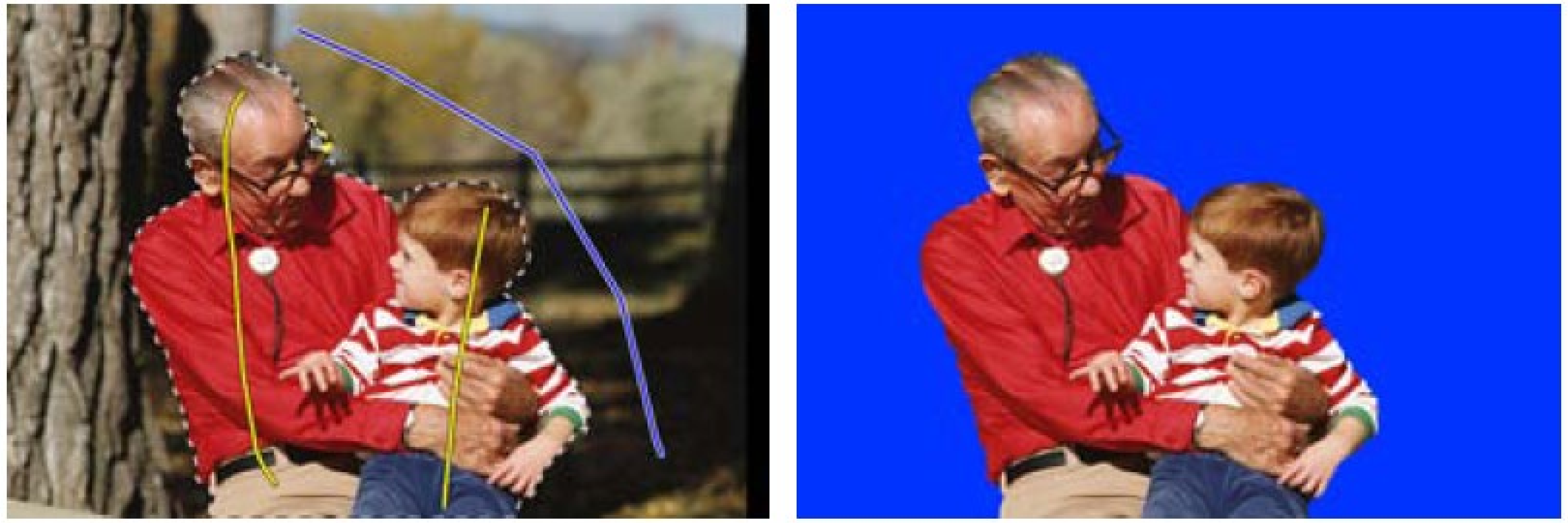
Initial Result

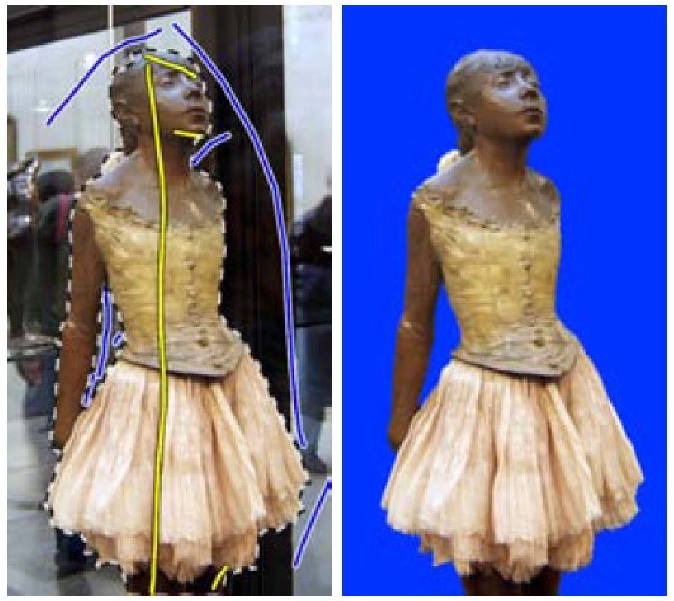


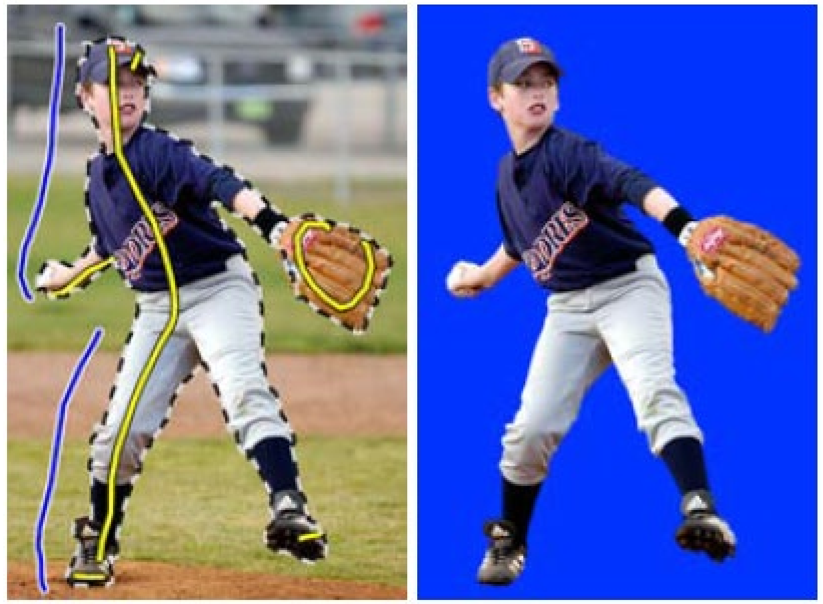




Lazy Snapping (Li et al. SG 2004)







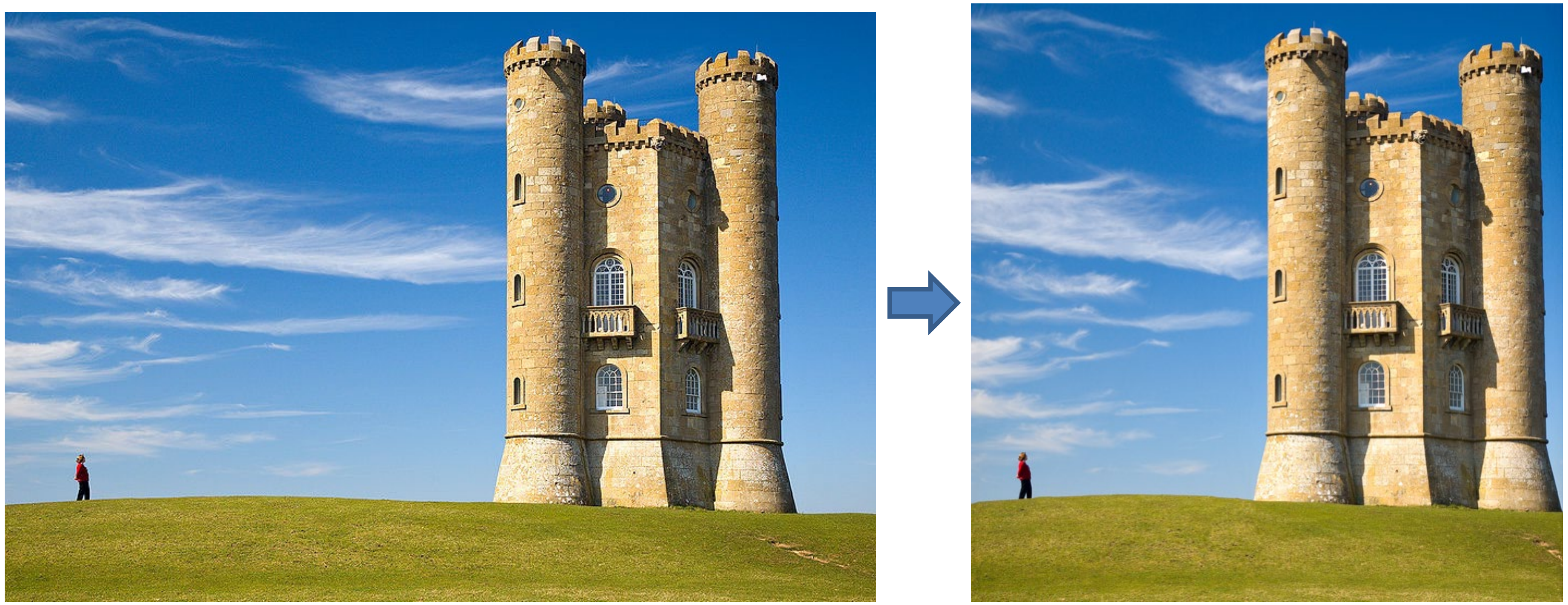
Limitations of Graph Cuts

- Requires associative graphs
  - Connected nodes should prefer to have the same label

- Is optimal only for binary problems

Other applications: Seam Carving

Seam Carving – Avidan and Shamir (2007)

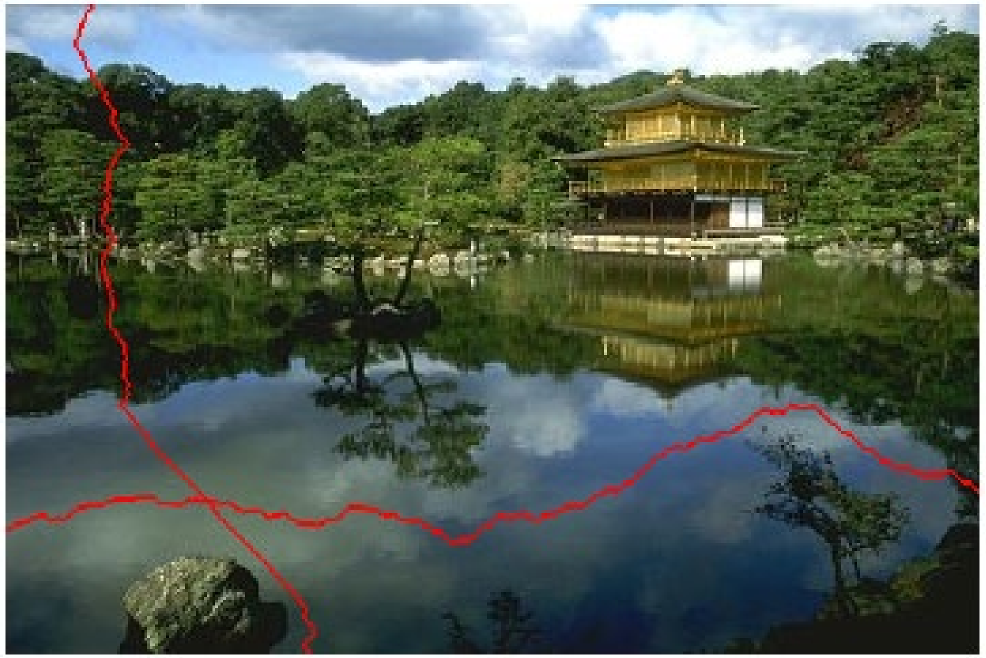


Demo: https://www.aryan.app/seam-carving/

Other applications: Seam Carving

- Find shortest path from top to bottom (or left to right), where cost = gradient magnitude

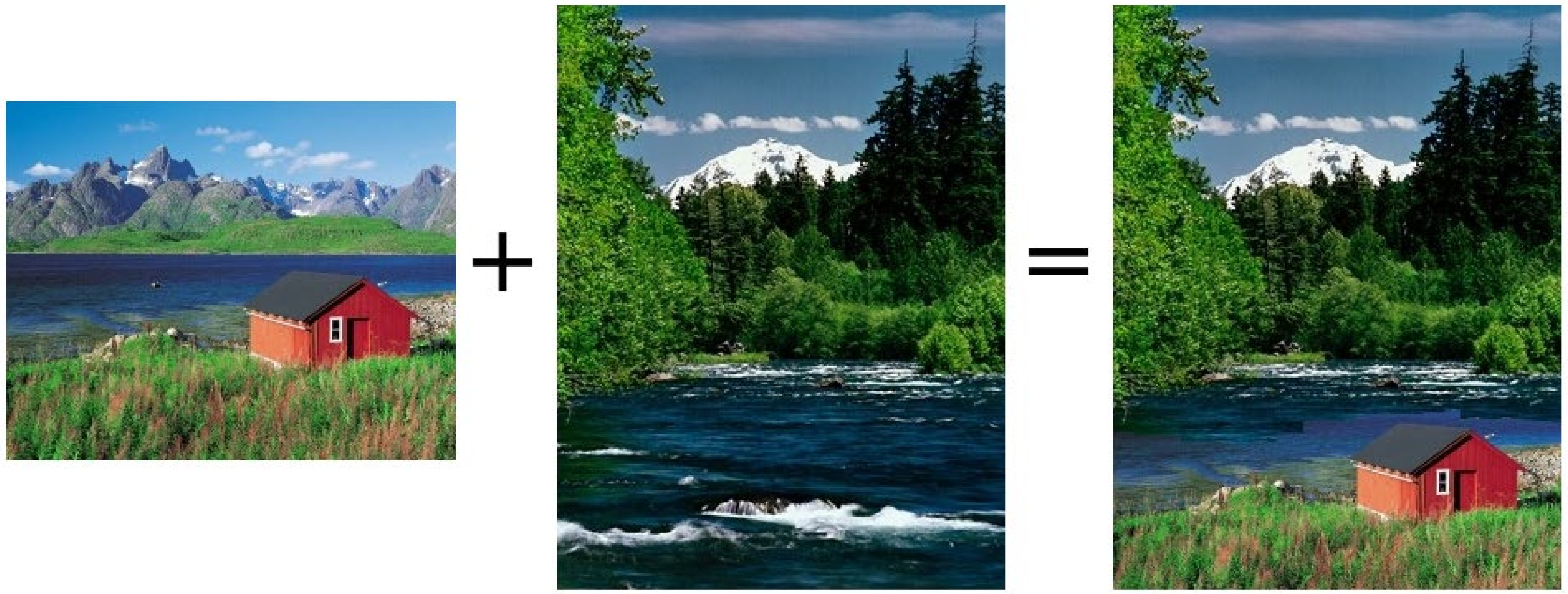
http://www.youtube.com/watch?v=6NcIJXTlugc



Seam Carving – Avidan and Shamir (2007)

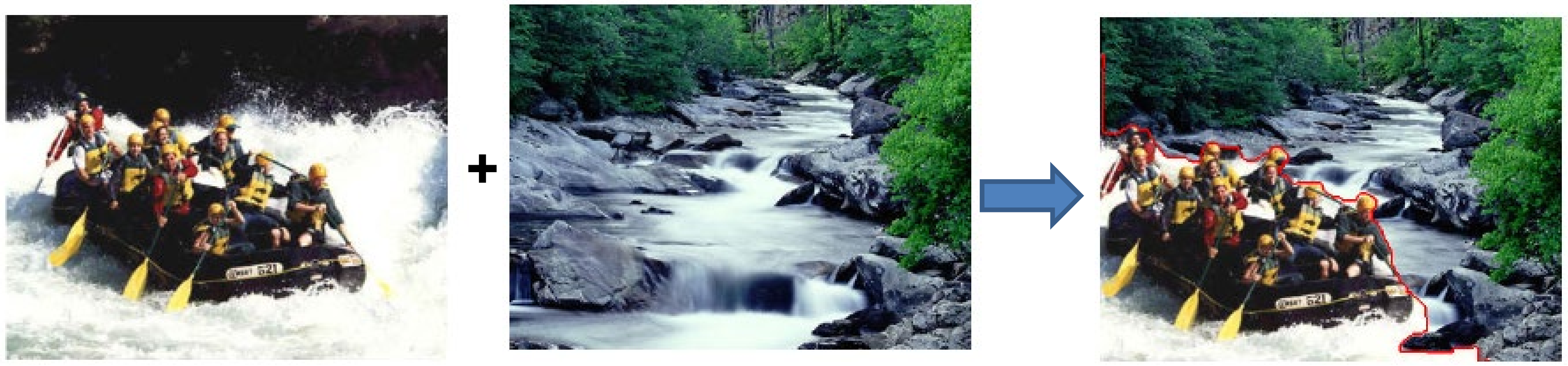
Other applications: stitching

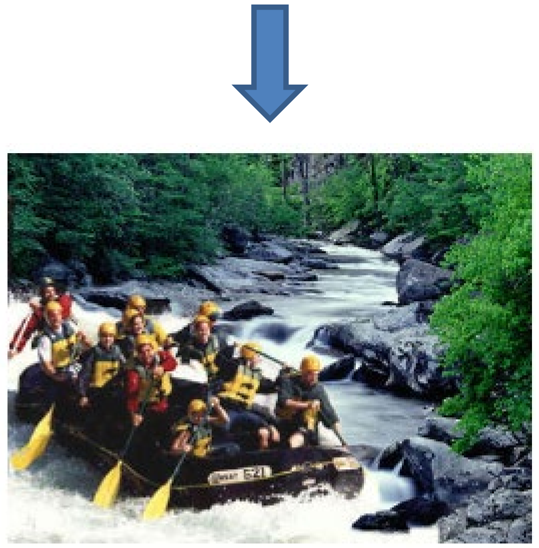
Graphcut Textures – Kwatra et al. SIGGRAPH 2003



Other applications: stitching

Graphcut Textures – Kwatra et al. SIGGRAPH 2003





Ideal boundary:

1. Similar color in both images
2. High gradient in both images

Summary of big ideas

- Treat image as a graph
  - Pixels are nodes
  - Between-pixel edge weights based on gradients
  - Sometimes per-pixel weights for affinity to foreground/background

- Good boundaries are a short path through the graph (Intelligent Scissors, Seam Carving)

- Good regions are produced by a low-cost cut (GrabCuts, Graph Cut Stitching)

Take-home questions

1. What would be the result in “Intelligent Scissors” if all of the edge costs were set to 1?

2. How could you change boundary costs for graph cuts to work better for objects with many thin parts?

Next class

- Compositing and blending