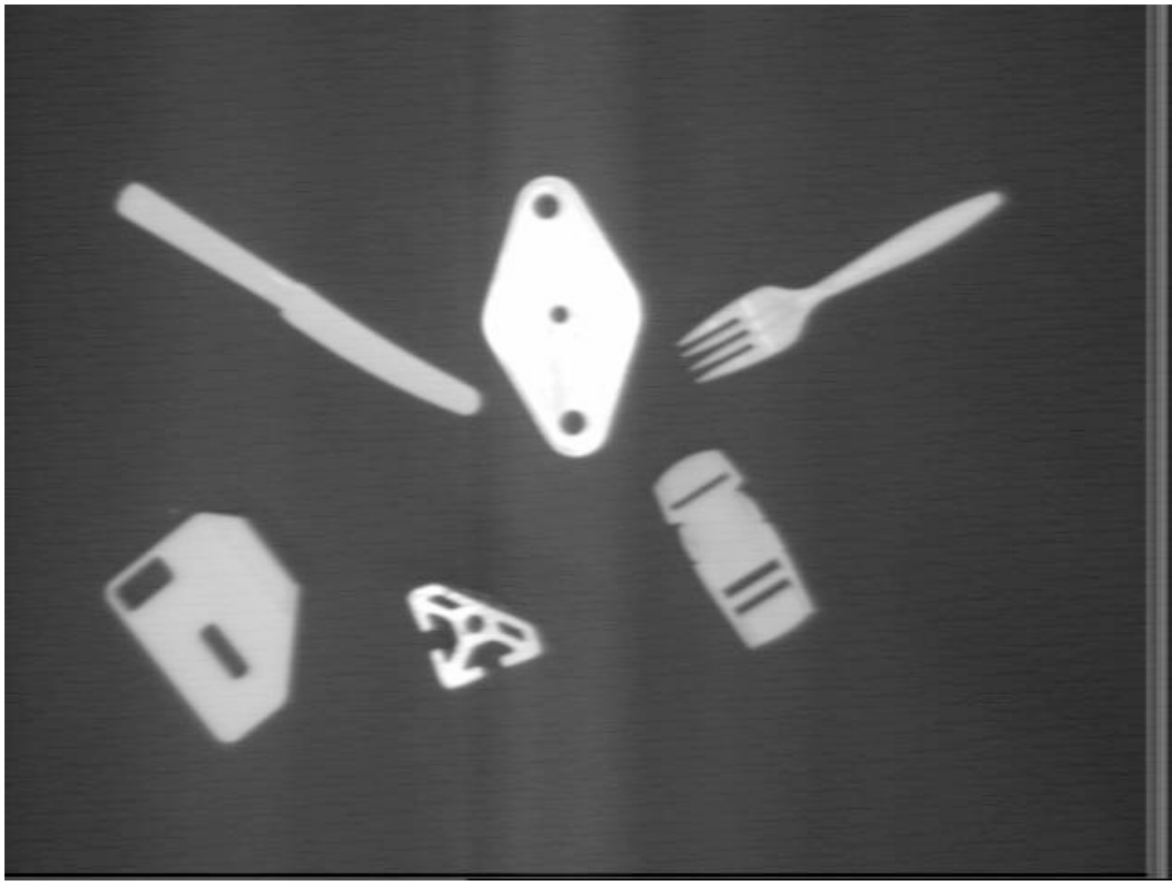
Computational Photography

Week 4, Spring 2009

Instructor: Prof. Ko Nishino

Multiple Objects

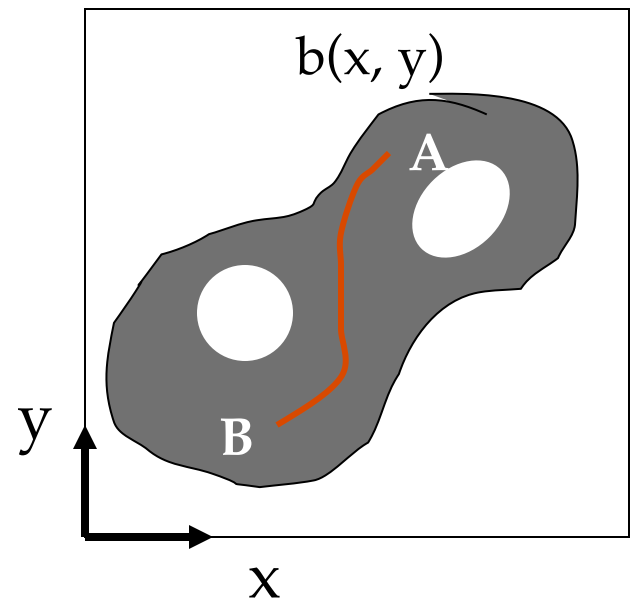


Need to SEGMENT image into separate COMPONENTS (regions)

- (Non-trivial !)

Connected Components

Maximal Set of Connected points



A & B are connected: Path exists between A & B along which b(x,y) is constant.

Connected Component Labeling

Region Growing Algorithm:

(a)Start with "SEED" point where $b(x,y) = 1$

(b) Assign LABEL to seed point

(c)Assign SAME LABEL to its Neighbors with $b(x,y) = 1$

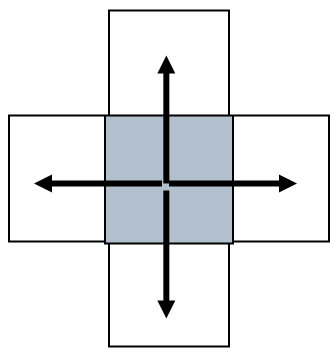
(d) Assign SAME LABEL to Neighbors of Neighbors

Terminates when a component is completely labeled.

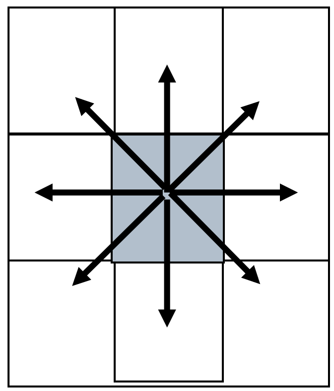
Then, pick another UNLABELED seed point.

What do we mean by Neighbors?

Connectedness:



4-connectedness



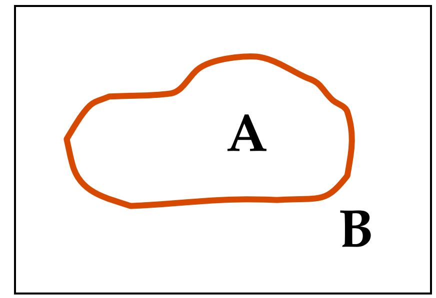
8-connectedness

Neither is perfect!

What do we mean by Neighbors?

- Jordan's Curve Theorem:

  Closed Curve -> 2 connected regions



- Consider:

0	1	0
1	0	1
0	1	0

(4-C)

B1	O1	B1
O2	B2	O3
B1	O4	B1

Hole without Closed curve!

(8-C)

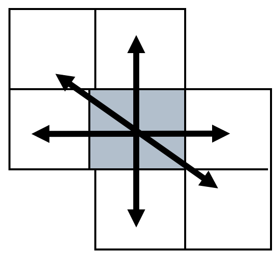
B	O	B
O	B	O
B	O	B

Connected Backgrounds with closed Ring!

Solution to Neighborhood Problem

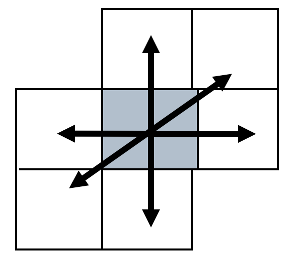
- Introduce Asymmetry

(a)



OR

(b)



- Using (b)

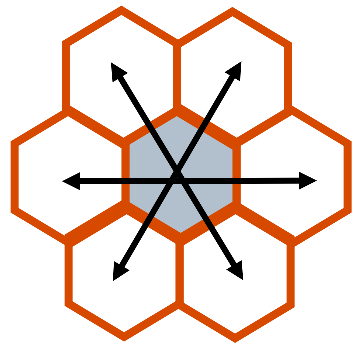
0	1	0
1	0	1
0	1	0

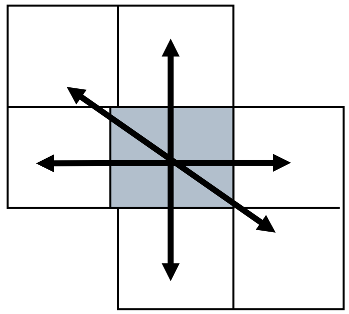
B	O1	B
O1	B	O2
B	O2	B

Two separate
Line Segments

We'll use (a) in the latter examples (see why)

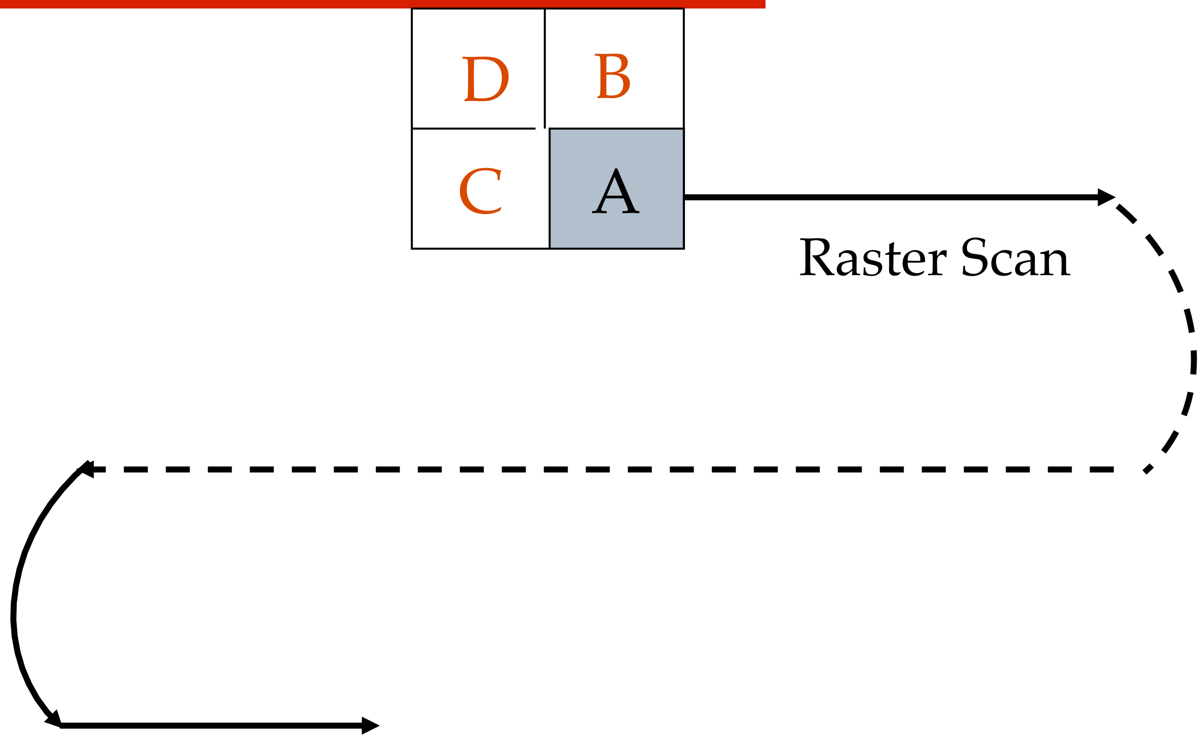
Hexagonal Tessellation





Asymmetry makes a SQUARE grid like HEXAGONAL grid

Sequential Labeling Algorithm



Note: We want to label A.

B, C, D are already labeled.

Sequential Labeling Algorithm

0	0
0	1

-> label(A) = new label

X	X
X	0

-> label(A) = "background"

D	X
X	1

-> label(A) = label(D)

0	0
C	1

-> label(A) = label(C)

0	B
0	1

-> label(A) = label(B)

0	B
C	1

-> If
label(B) = label(C)
then,
label(A) = label(B)

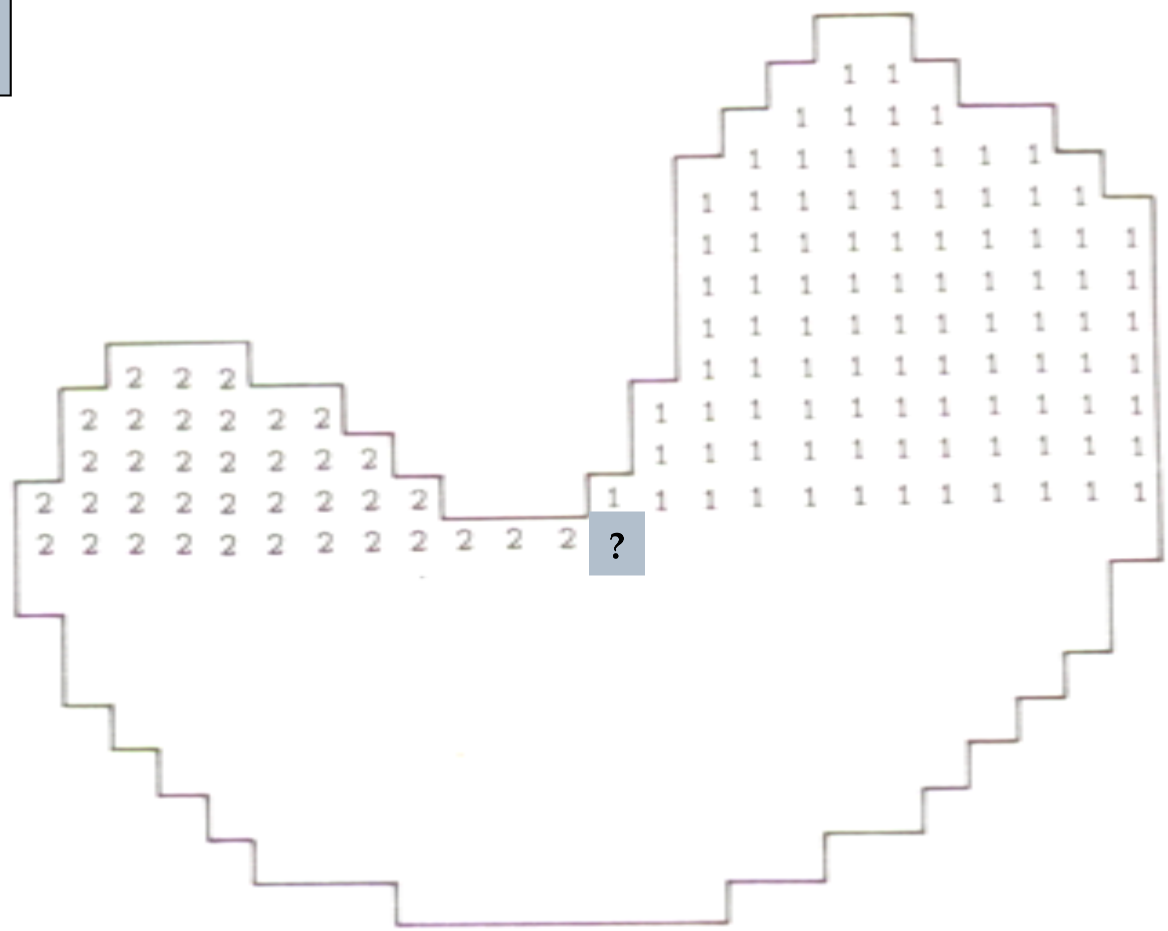
X is background

Sequential Labeling Algorithm

0	B
C	1

->

What if label(B) not equal to label(C)?



Sequential Labeling Algorithm

0	B
C	1

-> What if label(B) not equal to label(C)?

Solution:

Let: label(A) = label(B) = 2

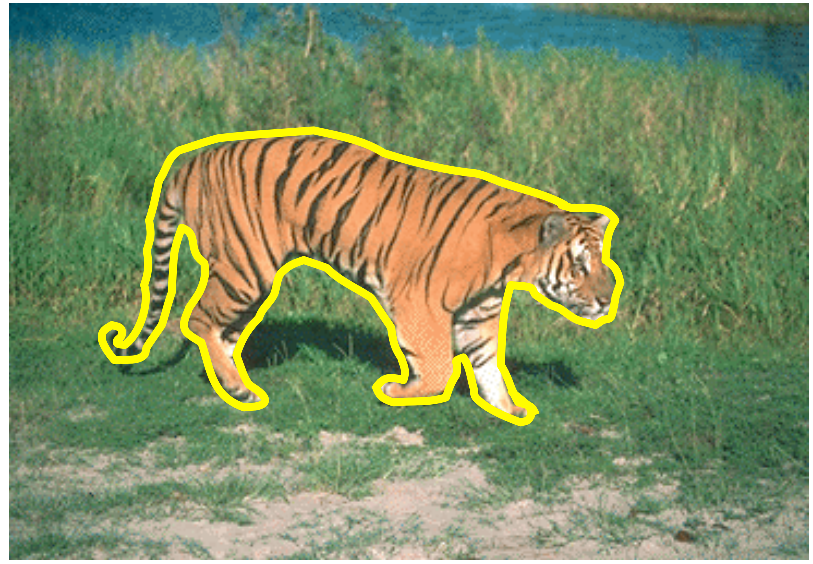
Create EQUIVALENCE TABLE

$2 \equiv 1$
$7 \equiv 3,6,4$
$\vdots$

Resolve Equivalence later
(merge them together)

Intelligent Scissors

Extracting Objects



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

Intelligent Scissors (demo)

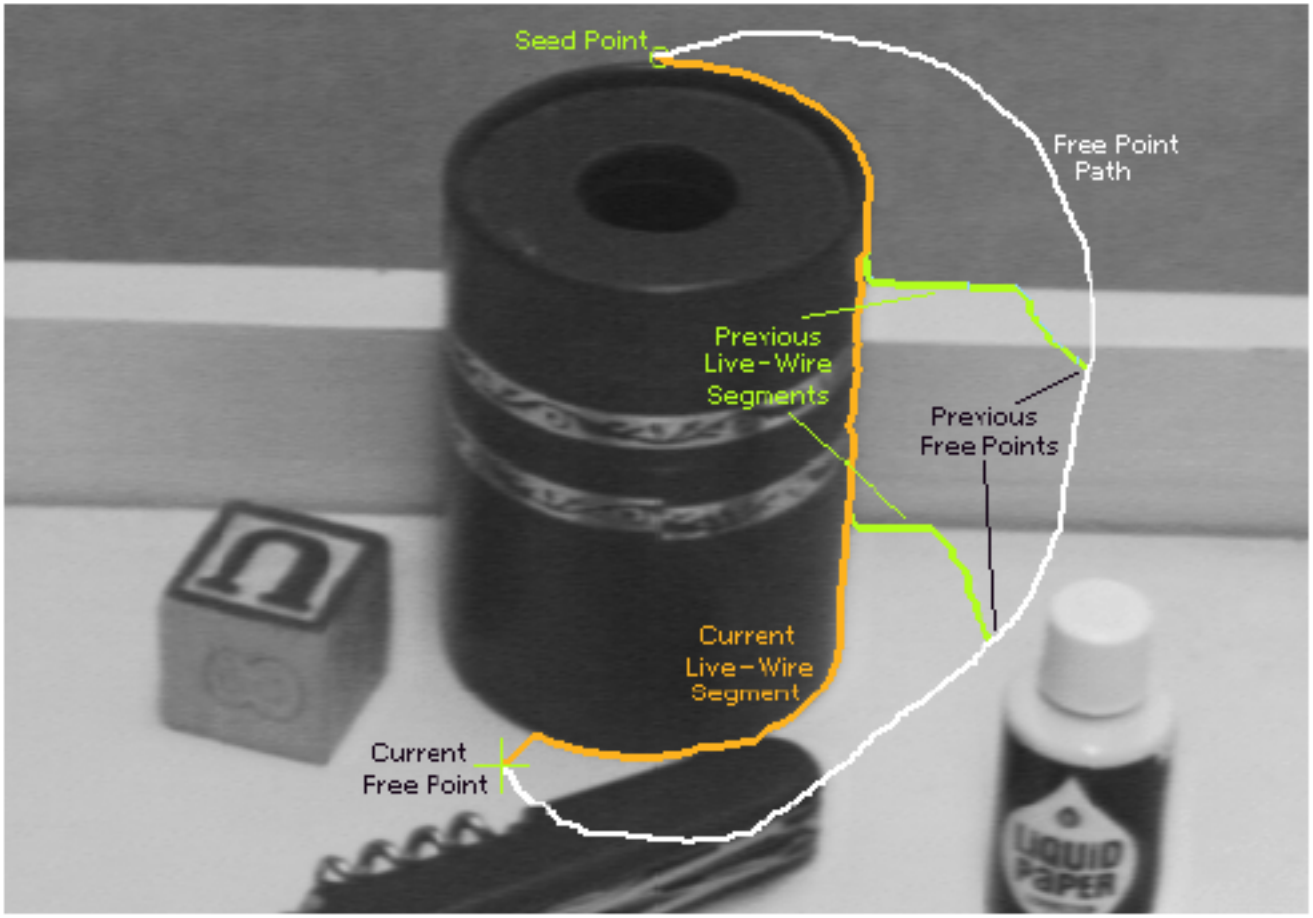


Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Intelligent Scissors

- Approach answers a 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

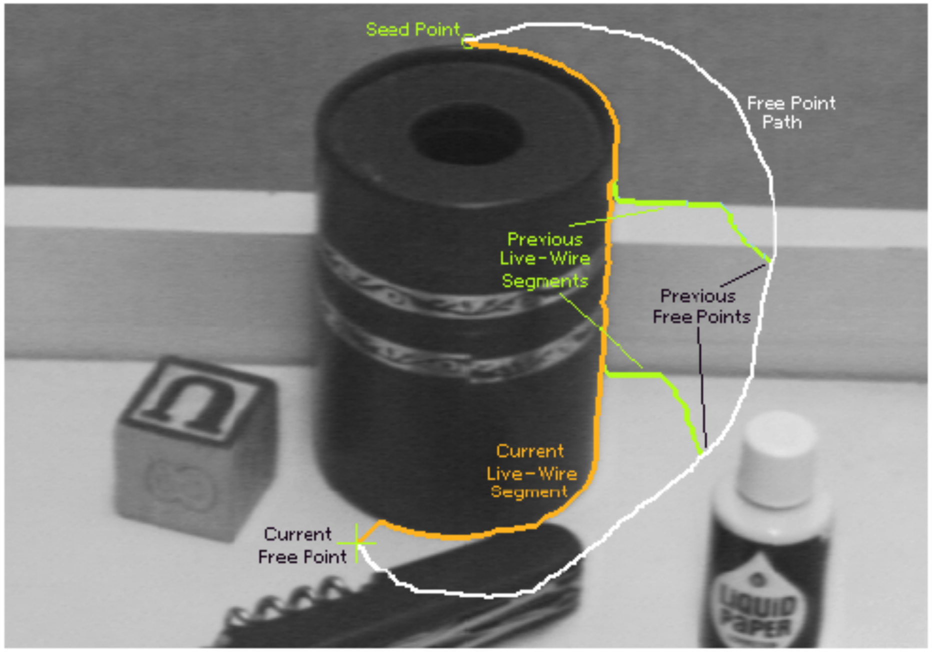
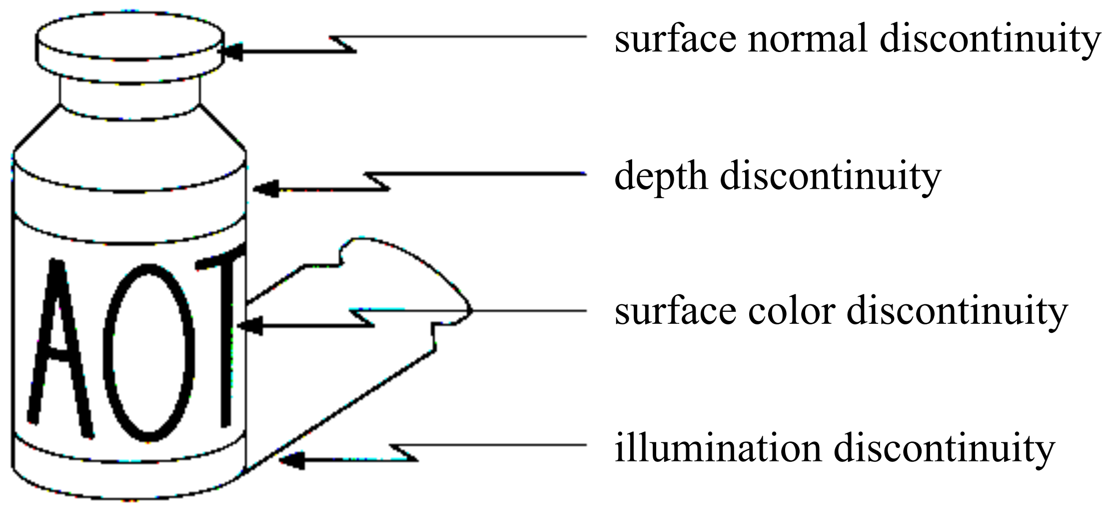


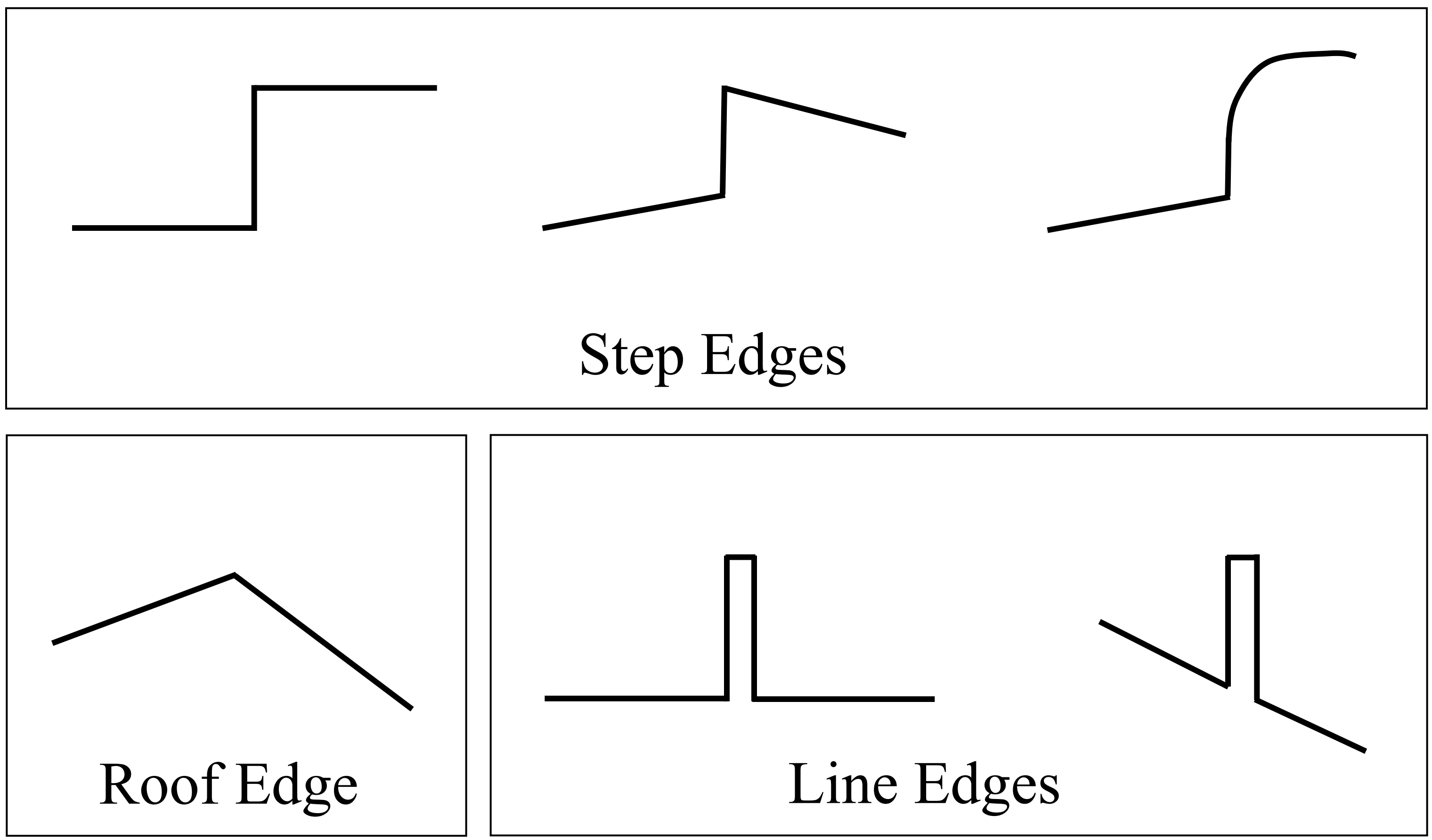
Figure 2: *Image demonstrating how the live-wire segment adapts and snaps to an object boundary as the free point moves (via cursor movement). The path of the free point is shown in white. Live-wire segments from previous free point positions (t_0, t_1, and t_2) are shown in green.*

Origin of Edges



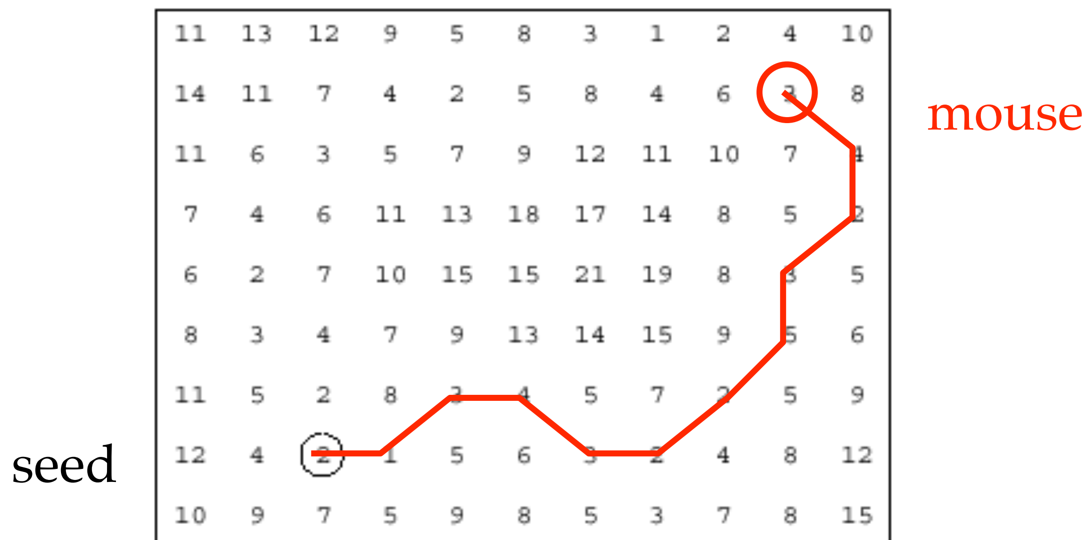
- Edges are caused by a variety of factors

Edge Types



Intelligent Scissors

- Basic Idea
  - Define edge score for each pixel
    - edge pixels have low cost
  - Find lowest cost path from seed to mouse (or the other point you clicked)



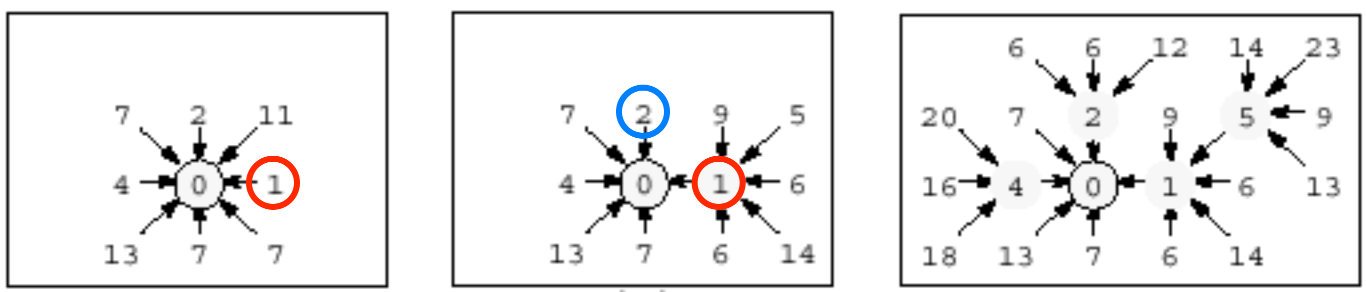
Questions

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

Path Search (basic idea)

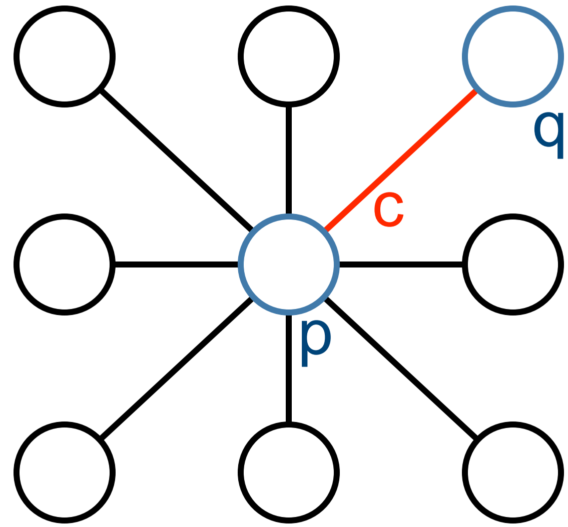
- Graph Search Algorithm
  - Computes minimum cost path from seed to *all other pixels*

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



How does this really work?

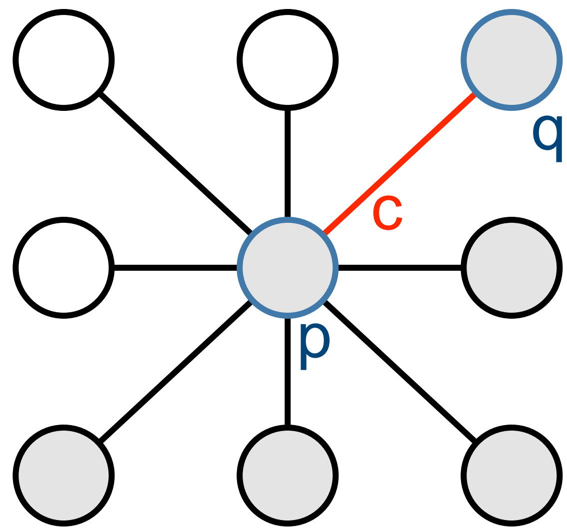
- Treat the image as a graph



- 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

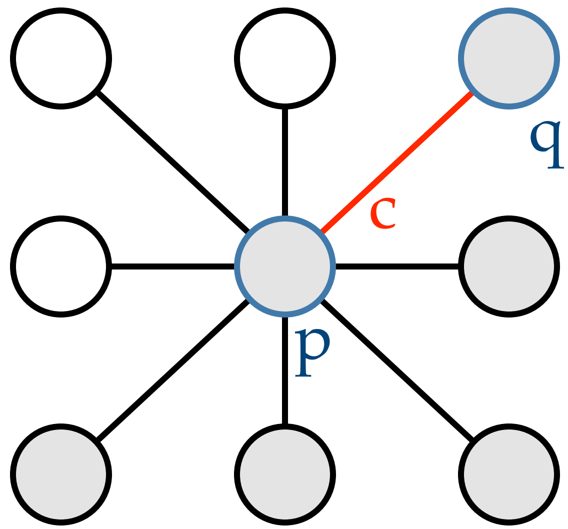
Defining the Costs

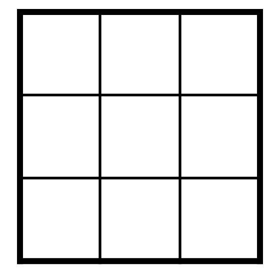
- Treat the image as a graph



- Want to hug image edges: how to define cost of a link?
  - the link should follow the intensity edge
    - want intensity to change rapidly ⊥ to the link
  - c ≈ - |difference of intensity ⊥ to link|

Defining the Costs

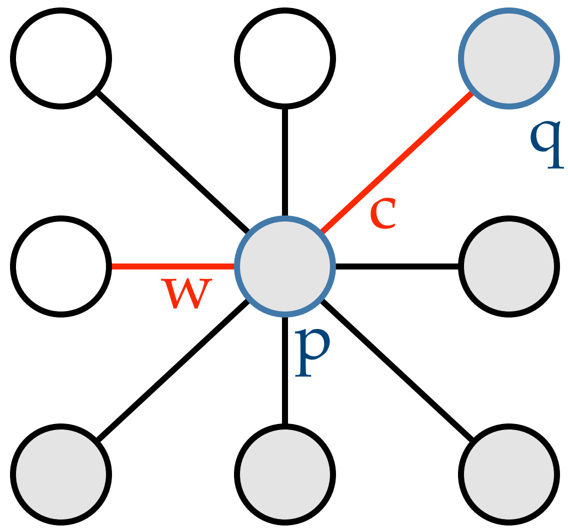




- c can be computed using a cross-correlation filter
  - assume it is centered at p

Defining the Costs

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

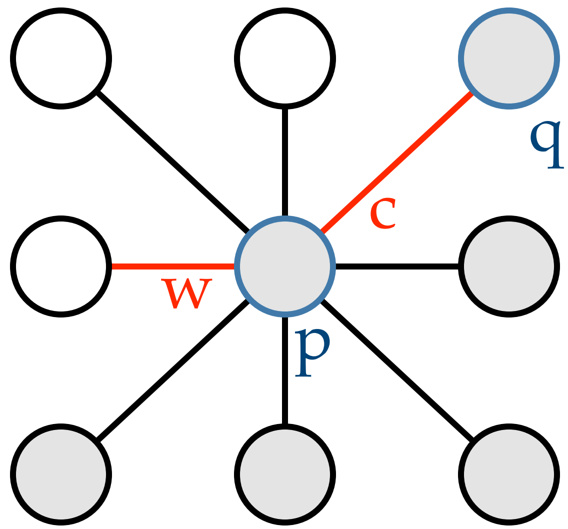


$$\frac{1}{2}\begin{bmatrix} & 1 & \\ & & -1 \\ & & \end{bmatrix}$$

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

Defining the Costs

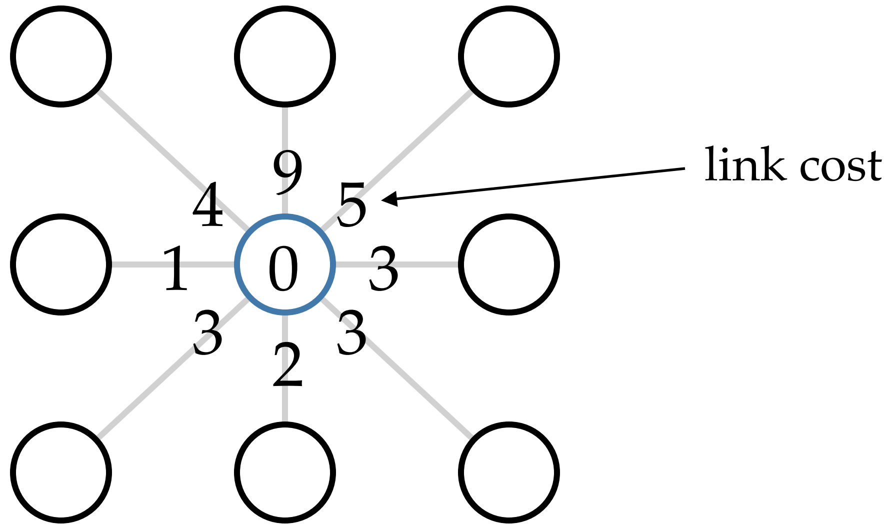
$$\frac{1}{4}\begin{bmatrix} 1 & 1 & \\ & & \\ -1 & -1 & \end{bmatrix}$$



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

- c can be computed using a cross-correlation filter
  - assume it is centered at p
- Also typically scale c by its length
  - set c = (max-|filter response|)×length
    - where max = maximum |filter response| over all pixels in the image

Dijkstra's Shortest Path Algorithm

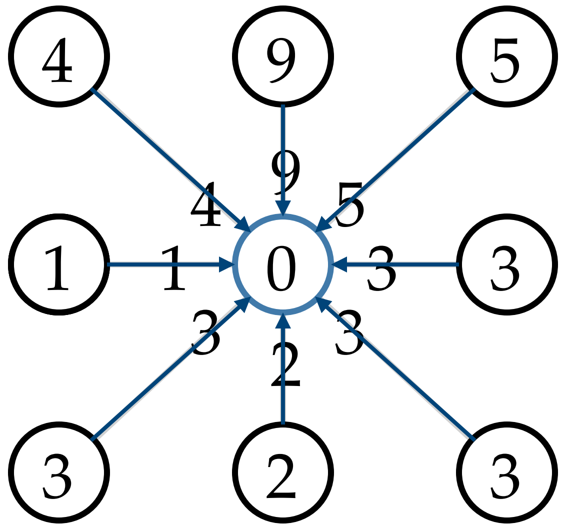


- Algorithm
  1. init node costs to ∞, set p = seed point, $cost(p) = 0$
  2. expand p as follows:

     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

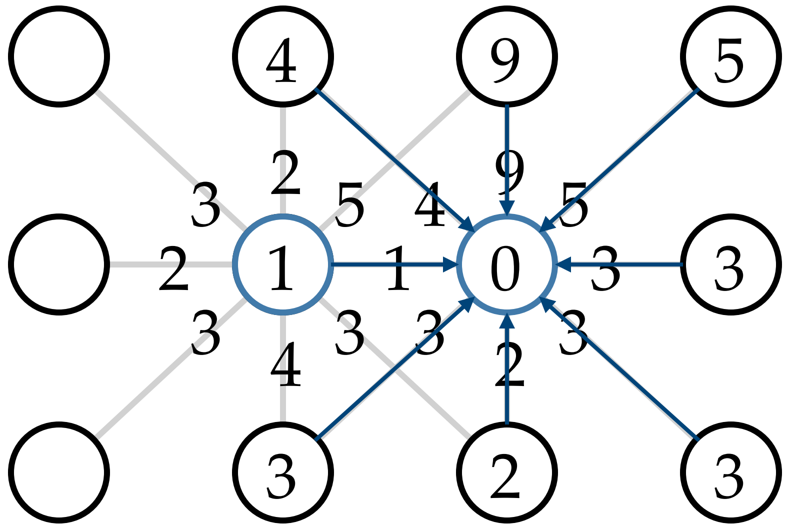


- Algorithm
  1. init node costs to ∞, set p = seed point, cost(p) = 0
  2. expand p as follows:

     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)

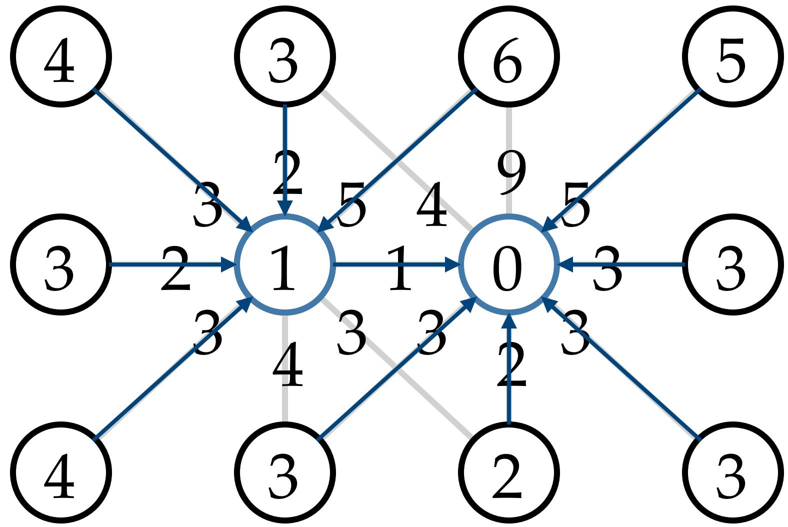
Dijkstra's Shortest Path Algorithm



- 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 (put r on EXPAND 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) + 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 (put r on EXPAND list)

Algorithm (of assignment)

Begin

initialize the priority queue *pq* to be empty;

initialize each node to the INITIAL state;

set the total cost of *seed* to be zero;

insert *seed* into *pq*;

extract the node *q* with the minimum total cost in *pq*;

% Now find the shortest path.

while *q* is **not** (0,0) & q is not goal

% If *q* is (0,0), it means the queue is empty.

mark *q* as EXPANDED;

for each neighbor node *r* of *q*

if *r* has **not** been EXPANDED

mark *r* as ACTIVE;

insert *r* in *pq* with the sum of the total cost of q and link cost from *q* to *r* as its total cost;

if inserting r changed it

make an entry for *r* in the Pointers array indicating that currently the best way to reach *r* is from *q*.

extract the node *q* with the minimum total cost in *pq*;

End

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)
  - What happens when the user specifies a new seed?

Compositing

Results