Segmentation

1

What is segmentation?

- **Segmentation divides an image into groups of pixels**
- **Pixels are grouped because they share some local property (gray level, color, texture, motion, etc.)**

boundaries **labels** **pseudocolors** **mean colors**

(different ways of displaying the output)

2

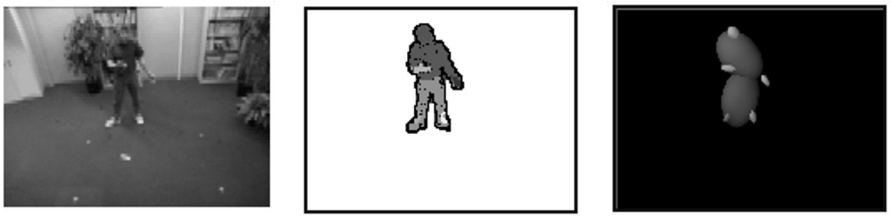
Other variants

- ***Segmentation = partitioning***
  Carve dense data set into (disjoint) regions
  - **Divide image based on pixel similarity**
  - **Divide spatiotemporal volume based on image similarity (shot detection)**
  - **Figure / ground separation (background subtraction)**
  - **Regions can be overlapping (layers)**
- ***Grouping = clustering***
  Gather sets of items according to some model
  - **If items are dense, then essentially the same problem as above (e.g., clustering pixels)**
  - **If items are sparse, then problem has a slightly different flavor:**
    - **Collect tokens that lie on a line (robust line fitting)**
    - **Collect pixels that share the same fundamental matrix (independent 3D rigid motion)**
    - **Group 3D surface elements that belong to the same surface**

The problems are closely related, but we will treat sparse clustering in a separate lecture (model fitting)

3

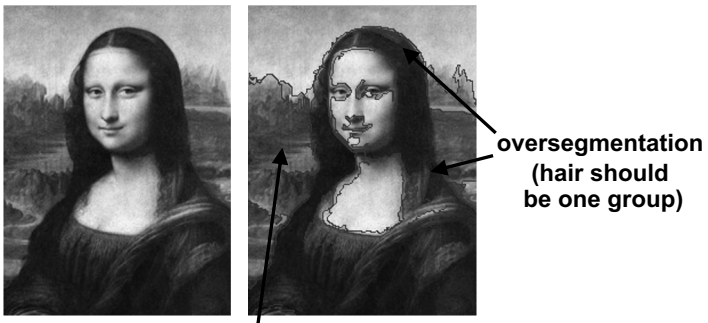
Foreground / background separation



Background subtraction provides figure-ground separation, which is a type of segmentation

4

Two errors



5

Outline

- **Human segmentation**
- **Standard algorithms:**
  - **Split-and-merge**
  - **Region growing**
  - **Minimum spanning tree**
- **Watershed algorithm**
- **Normalized cuts**

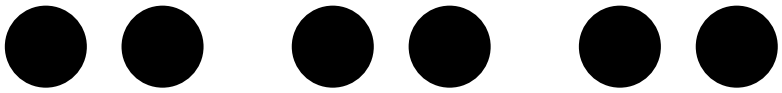
6

An experiment: What do you see?

Just six dots

7

Now what do you see?



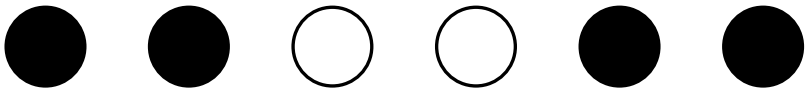
Three groups of dot pairs

Why?

Dots that are close together (“proximity”)
are grouped together by the human visual system

8

And now?



Again, three groups of dot pairs

Why?

Dots are similar in appearance (“similarity”)

9

How about now?

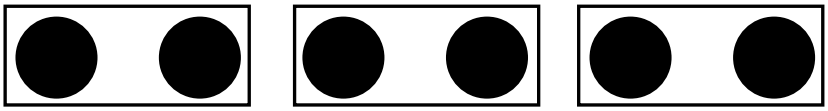
Again, three groups of dot pairs

Why?

Dots move similarly (“common fate”)

10

Last one



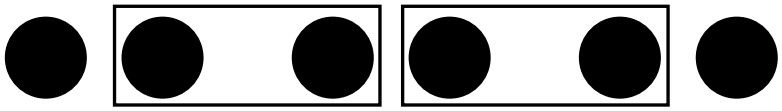
Again, three groups of dots

Why?

Dots are enclosed together (“common region”)

11

But wait!



Note that the “common region” can overwhelm the “proximity” tendency

12

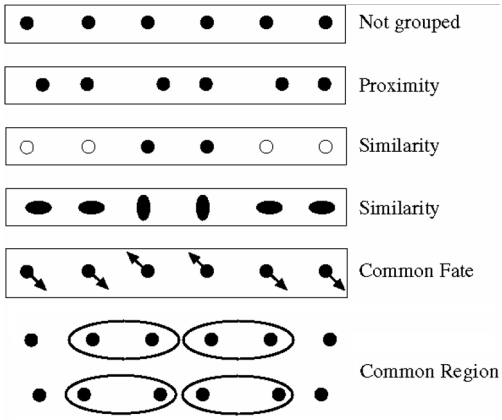
Gestalt psychology

Gestalt school of psychologists emphasized grouping as the key to understanding visual perception.

Recall: Context affects how things are perceived

***gestalt* – whole or group**

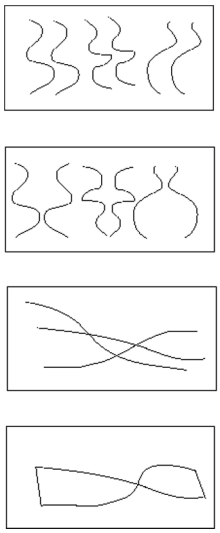
***gestalt qualitat* – set of internal relationships that makes it a whole**



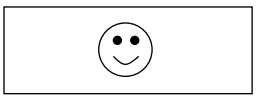
Max Wertheimer, Laws of Organization in Perceptual Forms, 1923
http://psy.ed.asu.edu/~classics/Wertheimer/Forms/forms.htm

13

Gestalt psychology (cont.)

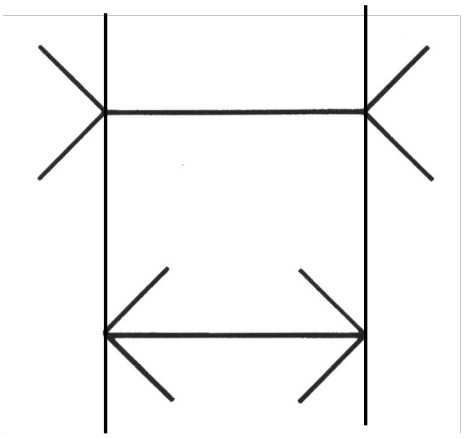


Familiar configuration



14

Muller-Lyer illusion

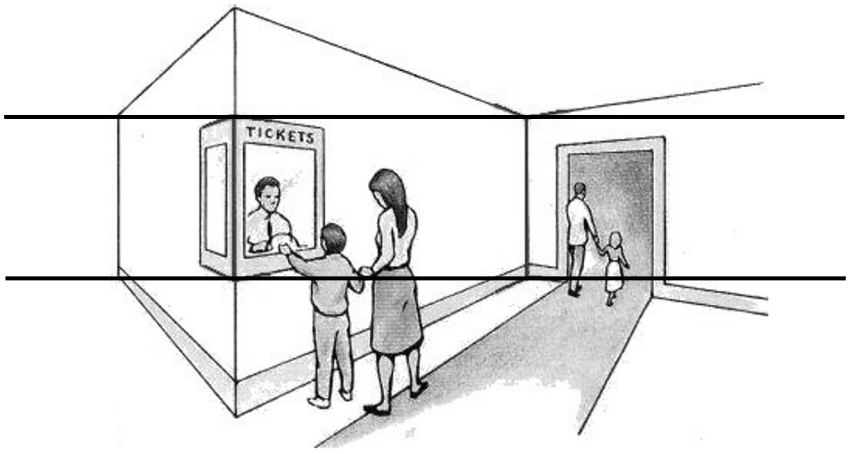


Lines are perceived as components of a whole rather than as individual lines.

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

15

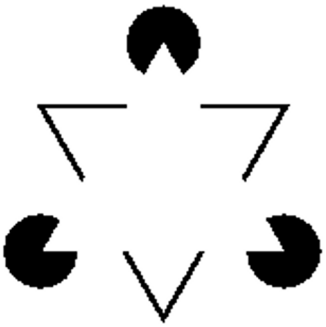
3D interpretation of Muller-Lyer



from http://www.michaelbach.de/ot/sze_muelue/

16

Can you see anything invisible?

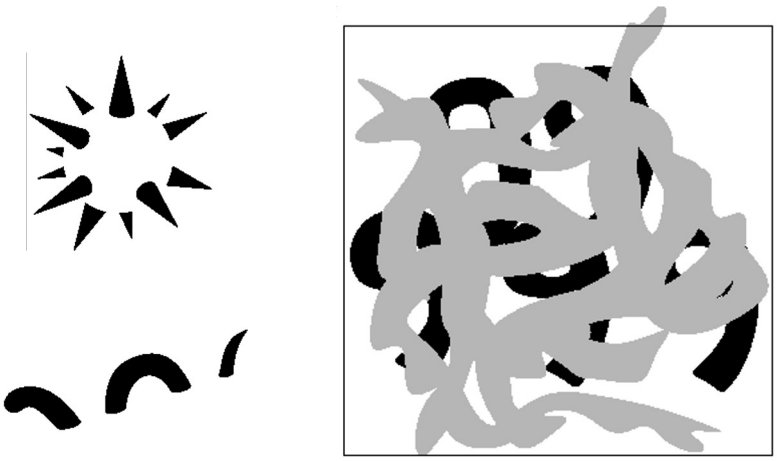


These are illusory contours, formed by grouping the circles

This is the well-known Kanizsa triangle

17

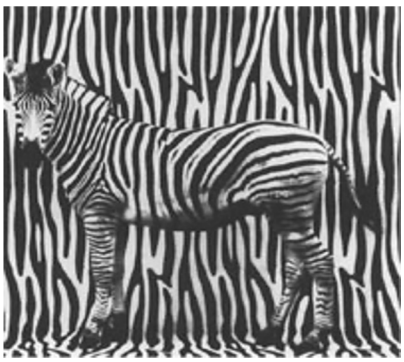
More illusory contours



Grouping by invisible completions

18

Two final examples



What role is top-down playing?

from http://www-static.cc.gatech.edu/classes/AY2007/cs4495_fall/html/materials.html

19

Back to computer vision…

20

Outline

- Human segmentation
- Standard algorithms:
  - Split-and-merge
  - Region growing
  - Minimum spanning tree
- Watershed algorithm
- Normalized cuts

21

Segmentation as partitioning

- A *partition* of image is collection of sets $S_1, .., S_N$ such that

  $I = S_1 \cup S_2 \ldots \cup S_N$ (sets cover entire image)

  $S_i \cap S_j = 0$ for all $i \neq j$ (sets do not overlap)

- A *predicate* $H(S_i)$ measures region *homogeneity*

  $H(R) = \begin{cases} true & \text{if pixels in region } R \text{ are similar} \\ false & \text{otherwise} \end{cases}$

- We want
  1. Regions to be homogeneous

     $H(S_i) = true$ for all i

  2. Adjacent regions to be different from each other

     $H(S_i \cup S_j) = false$ for all adjacent S_i, S_j

22

Two approaches

- **Splitting**
  (Divisive clustering)
  - **start with single region covering entire image**
  - **repeat: split inhomogeneous regions**
  - **even better:**
    repeat: split cluster to yield two distant components (difficult)

Property 2 is always true:
$H(S_i \cup S_j)$ = ***false*** **for adjacent regions**

Goal is to satisfy Property 1:
$H(S_i)$ = ***true*** **for every region**

- **Merging**
  (Agglomerative clustering)
  - **start with each pixel as a separate region**
  - **repeat: merge adjacent regions if union is homogeneous**
  - **even better:**
    repeat: merge two closest clusters

Property 1 is always true:
$H(S_i)$ = ***true*** **for every region**

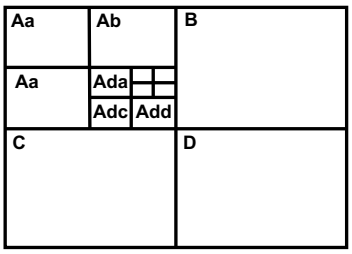
Goal is to satisfy Property 2:
$H(S_i \cup S_j)$ = ***false*** **for adjacent regions**

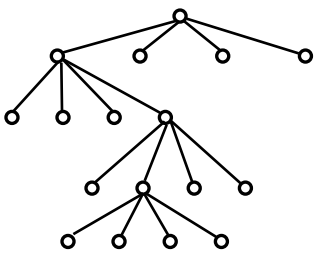
In practice, merging works much better than splitting

23

Region splitting

- **Start with entire image as a single region**
- **Repeat:**
  - **Split any region that does not satisfy homogeneity criterion into subregions**
- **Quad-tree representation is convenient**
- **Then need to merge regions that have been split**





24

Split-and-Merge

- **Split-and-merge algorithm combines these two ideas**
  - **Split image into quadtree, where each region satisfies homogeneity criterion**
  - **Merge neighboring regions if their union satisfies criterion (like connected components)**

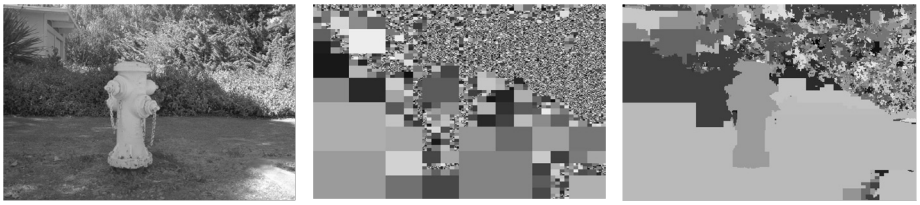


image **after split** **after merge**

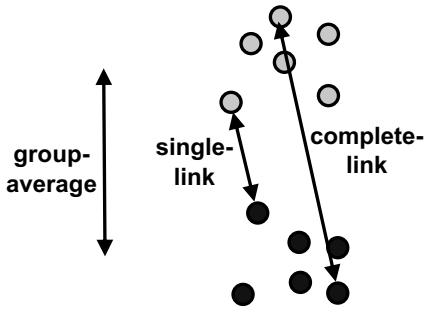
S. L. Horowitz and T. Pavlidis, Picture Segmentation by a Tree Traversal Algorithm, 1976

25

When to merge two clusters

Inter-cluster distance can be computed by

- **single-link clustering (dist. b/w closest elements) allows adaptation**
- **complete-link clustering (dist. b/w farthest elements) avoids drift**
- **group-average clustering (use average distance) good compromise**
- **root clustering (dist. b/w initial points of clusters) variation on complete-link**

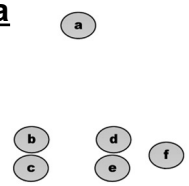


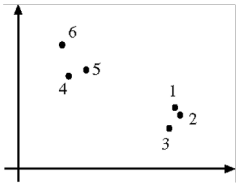
26

Dendrograms

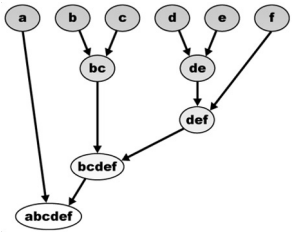
Dendrogram **yields a picture of output as clustering process continues**

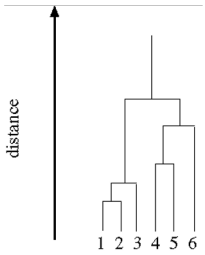
raw data





clusters represented as tree

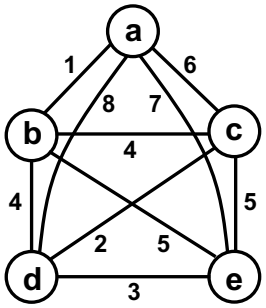




27

An example HCS

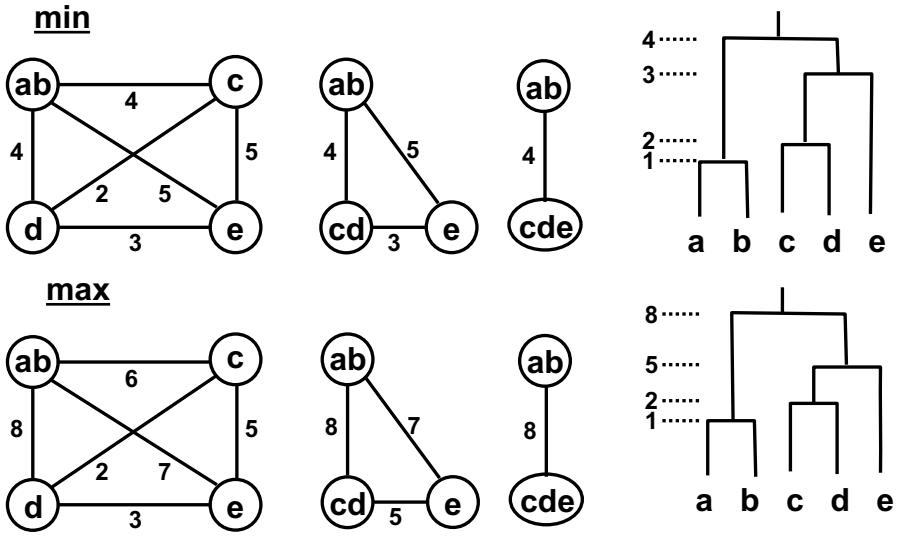
(hierarchical clustering scheme)



28

An example HCS

(hierarchical clustering scheme)



29

Region growing

- Start with (random) seed pixel as cluster
- Repeat:
  - Aggregate neighboring pixels that are similar to cluster model
  - Update cluster model with newly incorporated pixels
- This is a generalized floodfill
- When cluster stops growing, begin with new seed pixel and continue
- An easy cluster model:
  - Store mean and covariance of pixels in cluster
  - Use Mahalanobis distance to cluster
    This leads to a natural threshold, e.g., ± 2.5 σ
  - Update mean and covariance efficiently by keeping track of sum(x) and sum(x^2)
- One danger: Since multiple regions are not grown simultaneously, threshold must be appropriate, or else early regions will dominate

30

Region growing

```
GrowSingleRegion(I, O, p, label)
 1  model.initialize( I(p) )
 2  frontier.push(p)
 3  O(p) ← label
 4  while NOT frontier.isEmpty() do
 5        p ← frontier.pop()
 6        for q ∈ N(p) do
 7              if model.isSimilar( I(q) )
 8              then frontier.push(q)
 9                   O(q) ← label
10                   model.update( I(q) )
```

```
RegionGrow(I)
1  label ← 0
2  for (x, y) ∈ I do
3        L(x, y) ← UNLABELED
4  while L(x, y) = UNLABELED for some (x, y) do
5        p ← GetSeedPixel(I, L)
6        L ← GrowSingleRegion(I, L, p, label)
7        label ← label +1
8  return L
```

31

Region growing results

32

Region growing, balloons

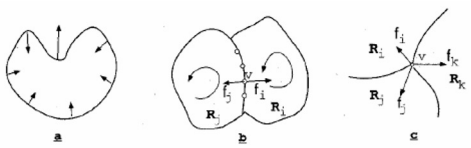
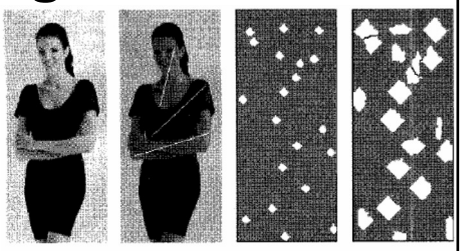
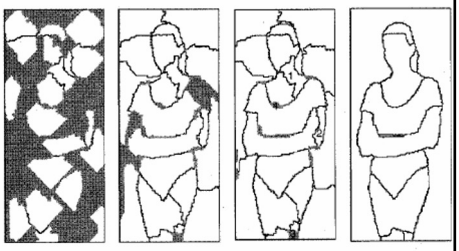


Fig. 2. The forces acting on the contour: (a) the smoothing force, (b the statistics force at a boundary point, (c) the statistics force at a junc tion point.



a (red band) b (grey level) c (t=0) d (t=10)



e (t=20) f (t=60) g (t=100) h (t=130)

S. C. Zhu and A.L Yuille**, Region Competition**: Unifying Snake/balloon, Region Growing and Bayes/MDL/Energy for Multi-band Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.18, no.9, pp.884-900, Sept. 1996.

33

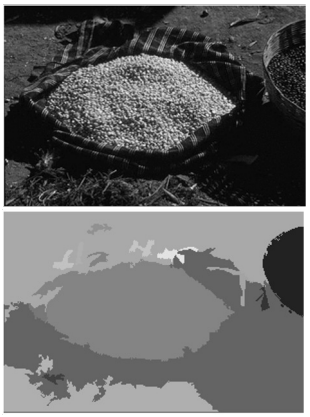
Stochastic relaxation

- **Geman and Geman**
- **Markov Random Field (MRF)**

34

Minimum spanning tree

- Agglomerative clustering can be implemented by building graph using pixels as nodes
- Repeated merging becomes finding a minimum spanning tree in a graph



35

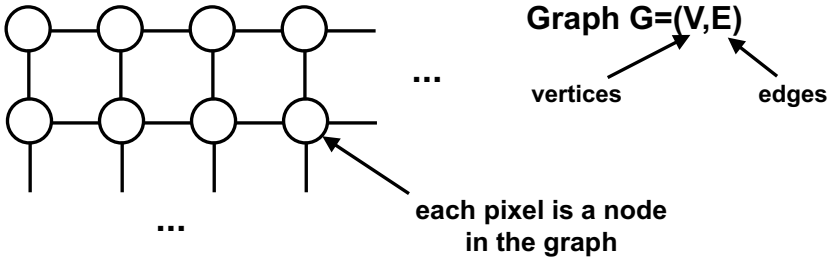
MST advantages

Minimum spanning tree (MST) answers two important questions unaddressed by region growing:

- How to select the starting pixels?
- Among the several pixels adjacent to the region, which should be considered next?

36

Image as a graph



- vertex *v* is just a number *v ≥ 0*
- edge *e=(u,v)* is a pair of vertices
- If undirected graph, then *(u,v)* ⇔ *(v,u)*
- If weighted graph, then *w(e)* is weight of edge

37

An example

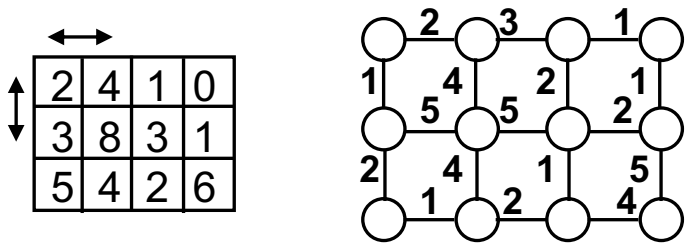


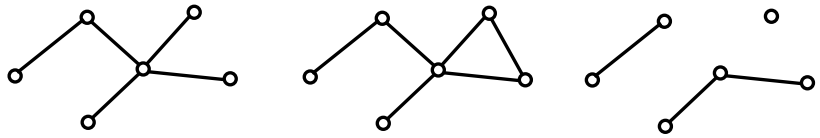
image graph

(using absolute difference in intensities)

38

Minimum spanning tree

path is sequence of vertices: $v_0, v_1, v_2, ..., v_k$
such that (v_i, v_{i+1}) **is edge for all** i
graph is connected if there exists a path b/w each pair of vertices
graph is tree if connected and acyclic (no cycles)
Examples:



tree | **not a tree (contains a cycle)** | **not a tree (not connected)**

Given a graph G=(V,E), the minimum spanning tree is a set of edges such that
- **resulting graph is a tree**
- **the sum of all the edge weights is minimal**

39

Kruskal's MST algorithm

1. **Initialize each vertex as separate set (or component or region)**
2. **Sort edges of E by weight (non-decreasing order)**
3. **T = ϕ (empty set)**
4. **for each edge (u,v)**
   1. **if FindSet(u) ≠ FindSet(v)**
      1. **T = T U {(u,v)}**
      2. **Merge(u, v)**
5. **return T**

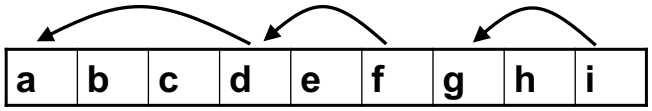
greedy algorithm yet optimal

40

Kruskal implementation

Kruskal's algorithm is implemented similar to connected components that we saw before

disjoint set data structure (equivalence table):



FindSet(u) recursively traces links (GetEquivalentLabel)
Merge(u,v) simply adds link b/w u and v (SetEquivalence)

How does this relate to image segmentation?
This procedure relates to a single image region; it finds the MST of each region in the image.

41

Kruskal's MST algorithm

```
KruskalMST(I)
1  T ← φ
2  disjoint-set.Initialize(width * height)
3  E ← ConstructEdges(I)
4  ⟨e_1, ..., e_n⟩ ← SortAscendingByWeight(E)
5  for (u, v) ← e_1 to e_n do
6        if disjoint-set.FindSet(u) ≠ disjoint-set.FindSet(v) then
7              T ← T ∪ {(u, v)}
8              disjoint-set.Merge(u, v)
9  return T
```

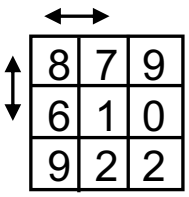
```
DisjointSet:Initialize(n)
1  for i ← 0 to n − 1 do
2        equiv[i] ← i
```

```
DisjointSet:Merge(u, v)
1  a ← min(FindSet(u), FindSet(v))
2  b ← max(FindSet(u), FindSet(v))
3  equiv[b] ← a
```

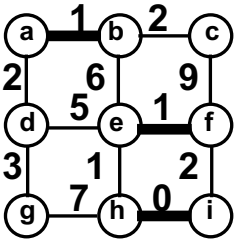
```
DisjointSet:FindSet(u)
1  p ← u
2  while p ≠ equiv[p] then
3        p ← equiv[p]
4  return p
```

42

Kruskal example



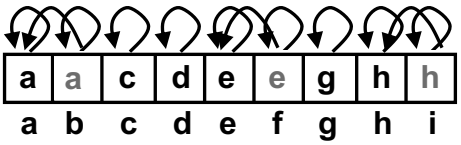
image



graph

Minimum spanning tree

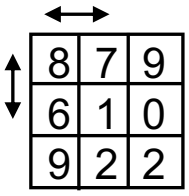
sorted edges: <hi_0,ab_1,ef_1,eh_1,bc_2,fi_2,ad_2,dg_3,de_5,be_6,gh_7,cf_9>



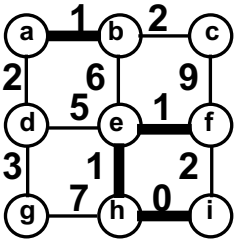
disjoint set data structure

43

Kruskal example



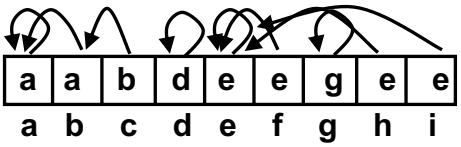
image



graph

Minimum spanning tree (after step 4)

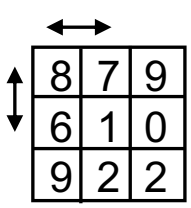
sorted edges: <hi_0,ab_1,ef_1,eh_1,bc_2,fi_2,ad_2,dg_3,de_5,be_6,gh_7,cf_9>



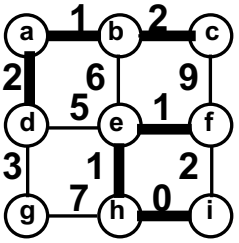
disjoint set data structure

44

Kruskal example



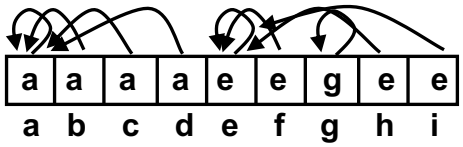
image



graph

Minimum spanning tree (after step 7)

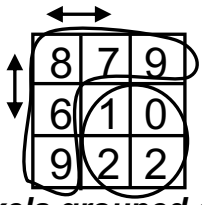
sorted edges: <hi_0,ab_1,ef_1,eh_1,bc_2,fi_2,ad_2,dg_3,de_5,be_6,gh_7,cf_9>



disjoint set data structure

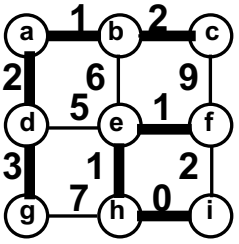
45

Kruskal example



Note: Pixels grouped correctly!

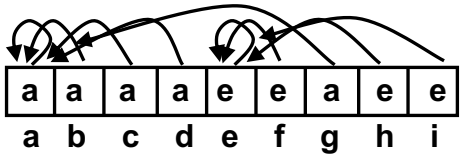
image



graph

Minimum spanning tree (before last step)

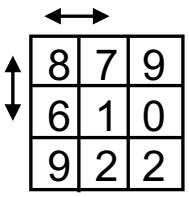
sorted edges: <hi_0,ab_1,ef_1,eh_1,bc_2,fi_2,ad_2,dg_3,de_5,be_6,gh_7,cf_9>



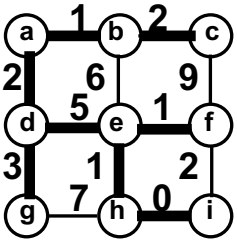
disjoint set data structure

46

Kruskal example



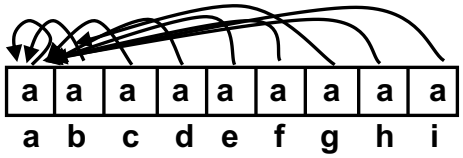
image



graph

Minimum spanning tree (final)

sorted edges: <hi_0,ab_1,ef_1,eh_1,bc_2,fi_2,ad_2,dg_3,de_5,be_6,gh_7,cf_9>



disjoint set data structure

47

MST image segmentation

- First, smooth image by convolving with Gaussian
  - Small variance (e.g., σ^2=0.5) is fine
  - This step is important to produce floating point weights for edges
- Build graph from image:
  - vertices are pixels
  - edges connect adjacent vertices (e.g., 4-adjacency)
  - edge weights are absolute intensity differences: $w(u,v) = | I(u) – I(v) |$
- Run Kruskal's algorithm on the graph, but only merge two regions if certain criteria are met
- While running the algorithm,
  - we have a *forest* (set of trees)
  - properties are maintained for each tree
  - trees are merged based on criteria
  - (but we don't care about the trees themselves, so we only need to store the regions, i.e., the set of pixels not edges)
- When the algorithm finishes,
  - the individual trees are the image regions

48

MST image segmentation

(Felzenszwalb-Huttenlocher IJCV 2004)

- **Initialize each pixel as separate set (or component or region)**
- **Sort edges of *E* by weight (non-decreasing order)**
- **for each edge (*u,v*)**
  - **if FindSet(*u*) ≠ FindSet(*v*) and**

$$w(u,v) < \min(\, m_u + k/N_u,\; m_v + k/N_v)$$

extra conditions

Note: weight must be floating point, b/c w(u,v) ≥ m_u (since edges are considered in non-decreasing order); otherwise, once $k/N_u < 1$, no more merging will occur

- **Merge(*u, v*)**

m_u = maximum weight of all edges in *u* MST
N_u = number of pixels in *u* region
k = parameter (e.g., 150 or 300)

49

MST image segmentation

(oversimplified version – does not work)

- **Initialize each pixel as separate set (or component or region)**
- **Sort edges of *E* by weight (non-decreasing order)**
- **for each edge (*u,v*)**
  - **if FindSet(*u*) ≠ FindSet(*v*) and**

$$(w(u,v) < m_u \text{ and } w(u,v) < m_v) \text{ or } (N_u < k \text{ and } N_v < k)$$

extra conditions

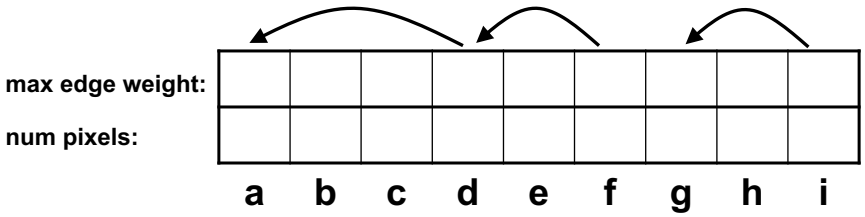
- **Merge(*u, v*)**

m_u = maximum weight of all edges in u MST
N_u = number of pixels in u region
k = parameter (minimum region size)

50

MST image segmentation

for each region, disjoint set data structure contains:
- **max edge weight of all edges in region's MST**
- **number of pixels in region**



Merge(u,v) is easy:
- **set max edge weight to max of two values**
- **set num pixels to sum of two values**

51

MST segmentation

```
MST-SEGMENTATION(I; σ, k)
 1  I_s ← SMOOTH(I; σ)                          ← smoothing necessary for IsSimilar2 to work
 2  disjoint-set.INITIALIZE(width * height)
 3  E ← CONSTRUCTEDGES(I_s)                     ← weights must be floats!
 4  ⟨e_1, ..., e_n⟩ ← SORTASCENDINGBYWEIGHT(E)
 5  for (u, v) ← e_1 to e_n do
 6        u' ← disjoint-set.FINDSET(u)
 7        v' ← disjoint-set.FINDSET(v)
 8        if u' ≠ v' and disjoint-set.ISSIMILAR2(w(u, v), u', v'; k) then
 9              disjoint-set.MERGE(u, v, w(u, v))
10  for (x, y) ∈ I do
11        L(x, y) ← disjoint-set.FINDSET(x, y)
12  return L
```

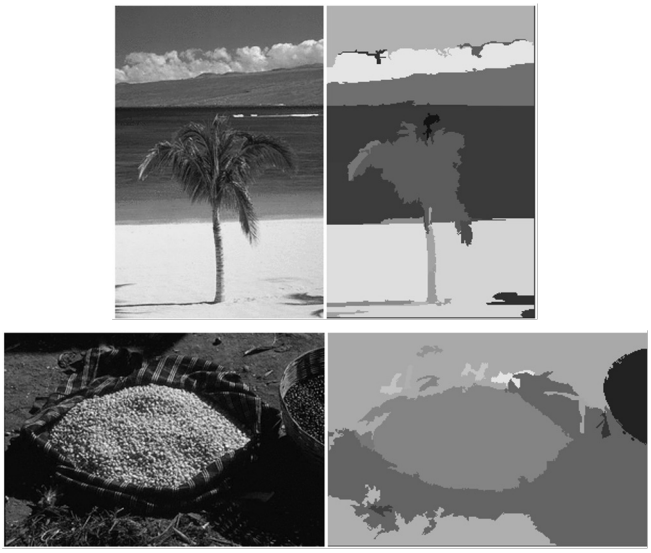
```
DISJOINTSET:ISSIMILAR2(w, u, v)                 ← do not use integer division!
1  return w < min(max-edge-weight[u] + k/num-pixels[u], max-edge-weight[v] + k/num-pixels[v])
```

```
DISJOINTSET:INITIALIZE(n)
1  for i ← 0 to n − 1 do
2        equiv[i] ← i
3        max-edge-weight[i] ← 0
4        num-pixels[i] ← 1
```

```
DISJOINTSET:MERGE(u, v, w)
1  a ← min(FINDSET(u), FINDSET(v))   } (FindSet is redundant if u' v' passed instead)
2  b ← max(FINDSET(u), FINDSET(v))   }
3  equiv[b] ← a
4  max-edge-weight[a] ← max(w, max-edge-weight[a], max-edge-weight[b])   ← w will always be maximum
5  num-pixels[a] ← num-pixels[a] + num-pixels[b]
```

52

Segmentation examples



from Pedro F. Felzenszwalb and Daniel P. Huttenlocher, Efficient Graph-Based Image Segmentation, IJCV, 59(2), 2004
http://people.cs.uchicago.edu/~pff/segment/

53

More examples

54

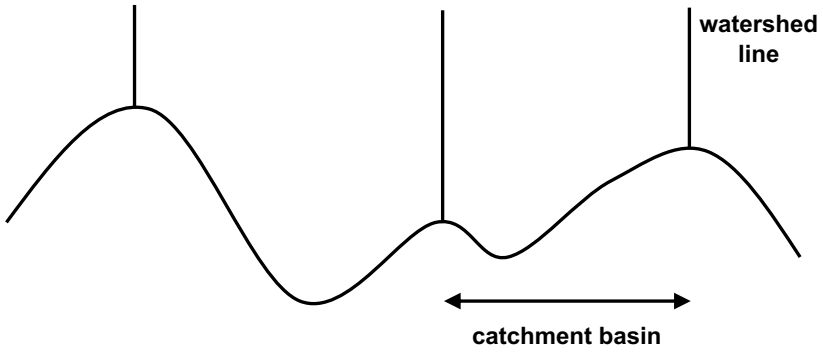
Outline

- Human segmentation
- Standard algorithms:
  - Split-and-merge
  - Region growing
  - Minimum spanning tree
- Watershed algorithm
- Normalized cuts

55

Watershed segmentation

Interpret (gradient magnitude) image as topographical surface:



All pixels in catchment basin are connected to minimum by monotonically decreasing path

56

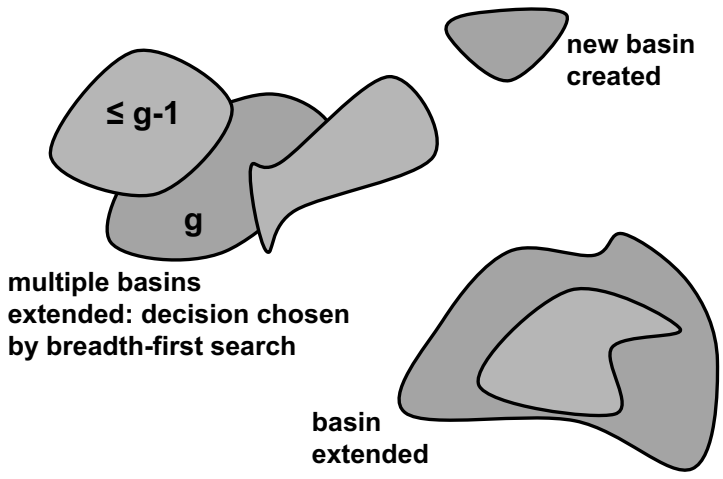
Watershed algorithms

- **Water immersion (Vincent-Soille)**
  - **Puncture hole at each local minimum, immerse in water**
  - **Grow level by level, starting with dark pixels**
  - ***Sorting step:*** **For efficiency, precompute for each graylevel a list of pixels with that graylevel (histogram with pointers)**
  - ***Flooding step:*** **Then, repeat:**
    - **Breadth-first search (floodfill) of level g given flooding up to level g-1**
    - **For each pixel with value g, either assign to closest catchment basin or declare new catchment basin (geodesic influence zone)**
- **Tobogganing**
  - **Find downstream path from each pixel to local minimum**
  - **Difficult to define for discrete (quantized) images because of plateaus**

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

57

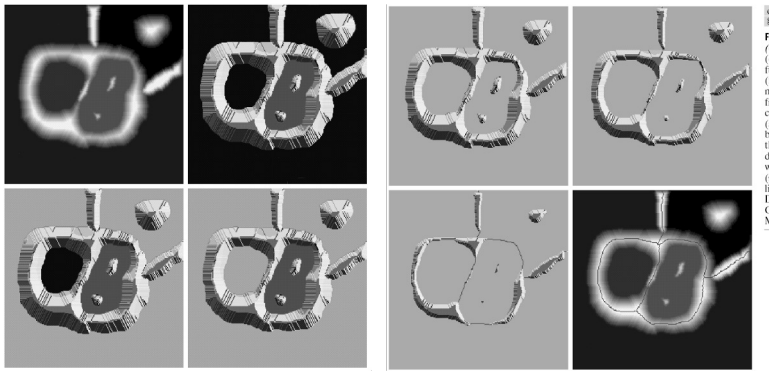
Vincent-Soille algorithm



S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

58

Watershed



a b
c d
FIGURE 10.44
(a) Original image.
(b) Topographic view. (c)–(d) Two stages of flooding.

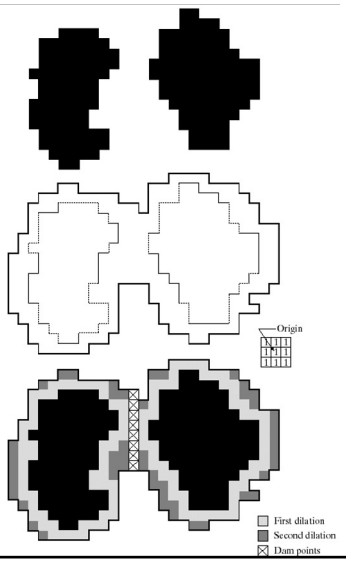
c f
g h
FIGURE 10.44
(Continued)
(e) Result of further flooding. (f) Beginning of merging of water from two catchment basins (a short dam was built between them). (g) Longer dams. (h) Final watershed (segmentation) lines. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

from Gonzalez and Woods, *Digital Image Processing*

59

Traditional watershed uses dams

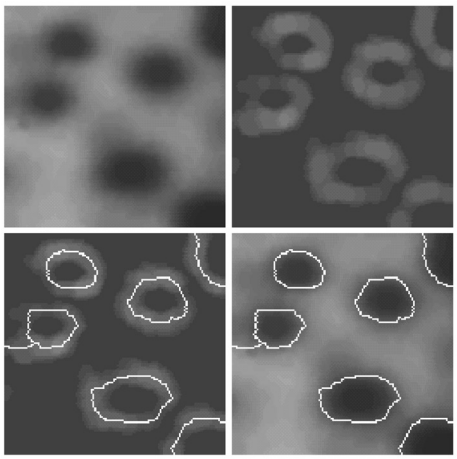
a
b
d c
FIGURE 10.45 (a) Two partially flooded catchment basins at stage $n-1$ of flooding. (b) Flooding at stage n, showing that water has spilled between basins (for clarity, water is shown in white rather than black). (c) Structuring element used for dilation. (d) Result of dilation and dam construction.



(But our implementation does not need dams)

60

Watershed results

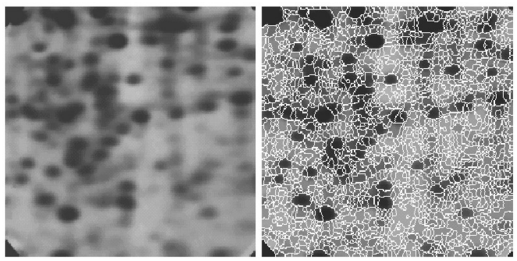


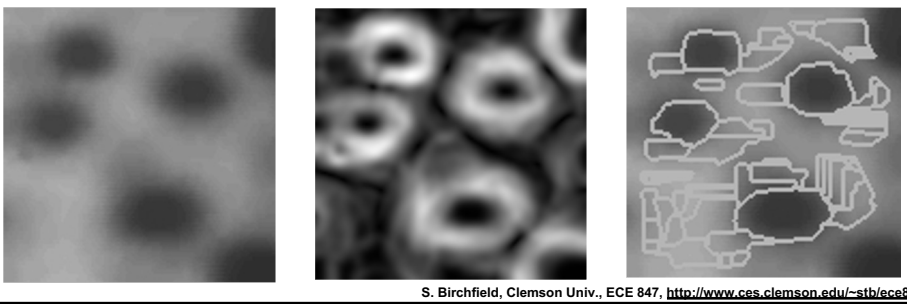
a b
c d

FIGURE 10.46
(a) Image of blobs. (b) Image gradient. (c) Watershed lines. (d) Watershed lines superimposed on original image. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

(But the results from our implementation will be better than this)

Watershed leads to oversegmentation

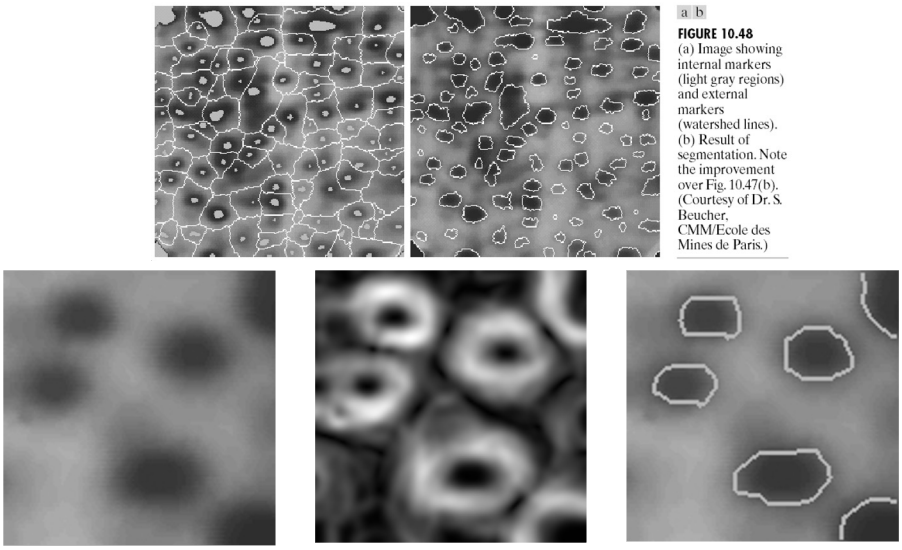




a b

FIGURE 10.47
(a) Electrophoresis image. (b) Result of applying the watershed segmentation algorithm to the gradient image. Oversegmentation is evident. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

Markers solve this problem



a b

FIGURE 10.48
(a) Image showing internal markers (light gray regions) and external markers (watershed lines). (b) Result of segmentation. Note the improvement over Fig. 10.47(b). (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

Marker-based watershed

- **Threshold image**
- **Compute chamfer distance**
- **Run watershed to get lines between objects**
- **Set these lines (skeletons) and blobs to zero in the gradient magnitude image**

```
for each x, y:
   grad(x,y) = max( grad(x,y), 1 )
   if marker(x,y), grad(x,y) = 0
```

(Alternatively, use separate gradient and marker images)

- **Only allow new basins where the value is zero**

Simplified Vincent-Soilles algorithm (Marker-based)

1. // initalization
   (a) Precompute array of pixel lists for each graylevel g
   (b) Set label[p]=-1 for all pixels p
   (c) globallabel = 0
2. // flood topological surface one graylevel at a time
   for each graylevel g=0 to G,
   (a) // grow existing catchment basins by one pixel, creating initial frontier
   for each pixel p such that img[p]=g and there exists a neighbor q of p for which label[q]≥0 (in an existing catchment basin),
      i. label[p] = label[q]
      ii. `frontier.push_back(p)`
   (b) // continue to grow existing basins one pixel thick each iteration by expanding frontier
   while !frontier.empty(),
      i. `p = frontier.pop_front()`
      ii. for each neighbor q of p such that img[q]==g and label[q]=-1 (unlabeled),
         A. label[q] = label[p]
         B. `frontier.push_back(q)`
   (c) // create new catchment basins
   for each p such that img[p]=g and ~~label[p]=-1 (still unlabeled),~~ **marker[p] = true**
      i. floodfill region containing p, assigning label 'globallabel++' **(floodfill using marker image)**

only needs to be done once initially (can use connected components)

(FIFO queue is important – use std::queue or std::deque)

Simplified Vincent-Soilles algorithm (in detail) (Marker-based)

1. **// initalization**
   (a) for each graylevel g=0 to G-1,
   pixellist[g].clear()
   (b) for each pixel p in input image f,
      i. pixellist[f(p)].push_back(p) // precompute pixel lists
      ii. label(p) = -1 // all pixels are initially unlabeled
   (c) next_label = 0
   (d) frontier.clear()
2. **// flood topological surface one graylevel at a time**
   for each graylevel g=0 to G-1,
   (a) **// grow existing catchment basins by one pixel, creating initial frontier**
   for each pixel p in pixellist[g],
   for each neighbor q of p,
   if label(q) ≥ 0 (in an existing catchment basin),
      i. label(p) = label(q)
      ii. `frontier.push_back(p)`
   (b) **// continue to grow existing basins one pixel thick each iteration by expanding frontier**
   while !frontier.empty(),
      i. `p = frontier.pop_front()`
      ii. for each neighbor q of p
   if f(q) == g and label(q) < 0 (unlabeled),
         A. label(q) = label(p)
         B. `frontier.push_back(q)`
   (c) **// create new catchment basins**
   for each pixel p in pixellist[g],
   if ~~label(p) < 0 (still unlabeled),~~ **marker(p) = true**
      i. floodfill region containing p, assigning label 'next_label' **(floodfill using marker image)**
      ii. next_label = next_label + 1

only needs to be done once initially (can use connected components)

FIFO queue is important to ensure that regions grow at equal rates (breadth-first search) (use std::queue or std::deque)

Simplified Vincent-Soilles algorithm (in detail)

```
WATERSHED(I)
    ▷ initialization
1   f ← GRADIENTMAGNITUDE(I)
2   for each value k ← 0 to n_grad − 1 do
3       pixellist[k].CLEAR()
4   for each pixel p ∈ f do
5       pixellist[f(p)].PUSHBACK(p)                    ▷ precompute pixel lists
6       L(p) ← UNLABELED                               ▷ all pixels are initially unlabeled
7   next-label ← 0
8   frontier.clear()
    ▷ flood topological surface one value at a time
9   for each value k ← 0 to n_grad − 1 do
        ▷ grow existing catchment basins by one pixel, creating initial frontier
10      for each pixel p in pixellist[k] do
11          if there exists a neighbor q of p such that L(q) ≠ UNLABELED
12          then L(p) ← L(q)                           ▷ (in an existing catchment basin)
13              frontier.PUSHBACK(p)
        ▷ continue to grow existing basins one pixel thick each iteration by expanding frontier
14      while NOT frontier.EMPTY() do
15          p ← frontier.POPFRONT()
16          if there exists a neighbor q of p such that f(q) ≤ k and L(q) == UNLABELED
17          then L(q) ← L(p)                           ▷ (unlabeled)
18              frontier.PUSHBACK(q)
        ▷ create new catchment basins
19      for each pixel p in pixellist[k] do
20          if L(p) == UNLABELED                       ▷ (still unlabeled)
21          then floodfill region containing p, assigning label next-label
22              next-label ← next-label +1
23  return L
```

67

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

68

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

0: (1,3)
1:
2:
3:
4:
5:
6:
7:
8:
9:

pixel list

Step 1:
Compute pixel list

69

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2:
3:
4:
5:
6:
7:
8:
9:

pixel list

Step 1:
Compute pixel list

70

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3:
4:
5:
6:
7:
8:
9:

pixel list

Step 1:
Compute pixel list

71

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 1:
Compute pixel list

72

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 2:
Set all pixels to "unlabeled"

73

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 3: k=0
Grow catchment basins (none to grow)

74

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 4: k=0
Expand frontier
(no frontier yet)

75

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	0	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 5: k=0
Create new catchment basins

76

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	0	0	u	u
u	0	u	u	u

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 6: k=1
Grow catchment basins

77

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	0	0	u	u
u	0	u	u	u

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 7: k=1
Expand frontier
(nowhere to go)

78

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	1	u
u	u	u	u	u
u	0	0	u	u
u	0	u	u	2

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 8: k=1
Create new basins

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	1	1	1
u	0	u	u	u
u	0	0	u	2
u	0	u	u	2

labels

(shaded pixels are on frontier)
0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 9: k=2
Grow basins

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	1	1	1
u	0	u	u	u
u	0	0	u	2
u	0	u	u	2

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 10: k=2
Expand frontier
(nowhere to go)

81

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	1	1	1
u	0	u	u	u
u	0	0	u	2
u	0	u	u	2

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 11: k=2
Create new basins
(none to create)

82

Fast forward…

83

Watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

0	0	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	2
0	0	0	0	2

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list **Final result (but note that ties can be broken in other ways)**

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

84

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

There are two markers here, indicated by

85

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Steps 1 and 2 (initialization) are the same as before

86

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 3: k=0
Grow catchment basins (none to grow)

87

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 4: k=0
Expand frontier (no frontier yet)

88

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	1
u	0	u	u	u
u	u	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 5: k=0
Create new catchment basins (at markers)

89

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	1
u	0	0	u	u
u	0	u	u	u

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 6: k=1
Grow catchment basins

90

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	8	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	1
u	0	0	u	u
u	0	u	u	u

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 7: k=1
Expand frontier
(nowhere to expand)

91

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	8	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	u
u	u	u	u	1
u	0	0	u	u
u	0	u	u	u

labels

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Skip step to create new catchment basins, because this is only done at markers

92

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	u	u	1
u	0	u	u	1
u	0	0	u	1
u	0	u	u	u

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 8: k=2
Grow catchment basins

93

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	u
u	u	1	1	1
u	0	u	u	1
u	0	0	u	1
u	0	u	u	1

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 9: k=2
Expand frontier

94

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	1
u	u	1	1	1
u	0	u	1	1
0	0	0	u	1
u	0	0	u	1

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 10: k=3
Grow catchment basins

95

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	0	1	8	2
6	1	3	6	1

gradmag image

u	u	u	u	1
u	u	1	1	1
u	0	u	1	1
0	0	0	u	1
u	0	0	u	1

labels

(shaded pixels are on frontier)

0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list

Step 11: k=3
Expand frontier
(nowhere to expand)

96

Fast forward…

97

Marker-based watershed example

8	7	5	4	3
9	6	2	1	2
4	2	7	3	4
3	8	1	8	2
6	1	3	6	1

gradmag image

0	0	1	1	1
0	0	1	1	1
0	0	0	1	1
0	0	0	0	1
0	0	0	0	1

labels

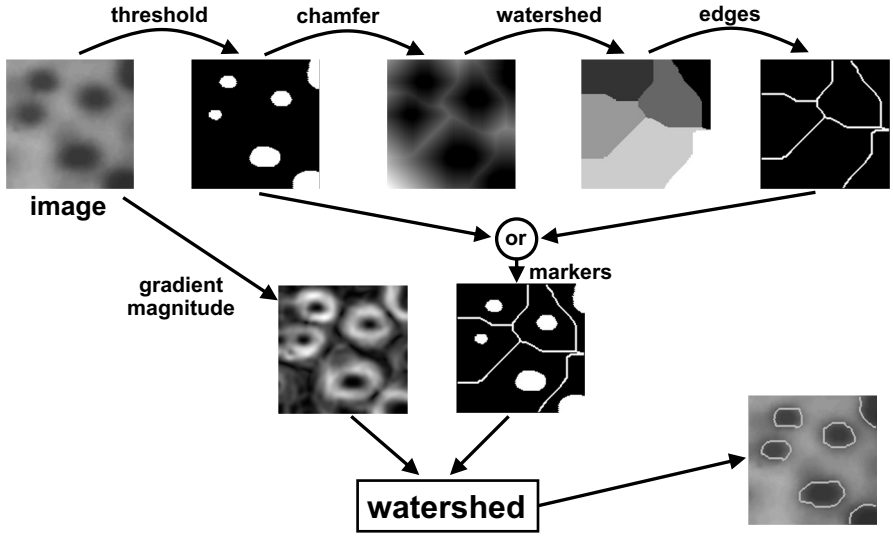
0: (1,3)
1: (3,1), (2,3), (1,4), (4,4)
2: (2,1), (4,1), (1,2), (4,3)
3: (4,0), (3,2), (0,3), (2,4)
4: (3,0), (0,2), (4,2)
5: (2,0)
6: (1,1), (0,4), (3,4)
7: (1,0), (2,2)
8: (0,0), (3,3)
9: (0,1)

pixel list **Final result (but note that ties can be broken in other ways)**

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

98

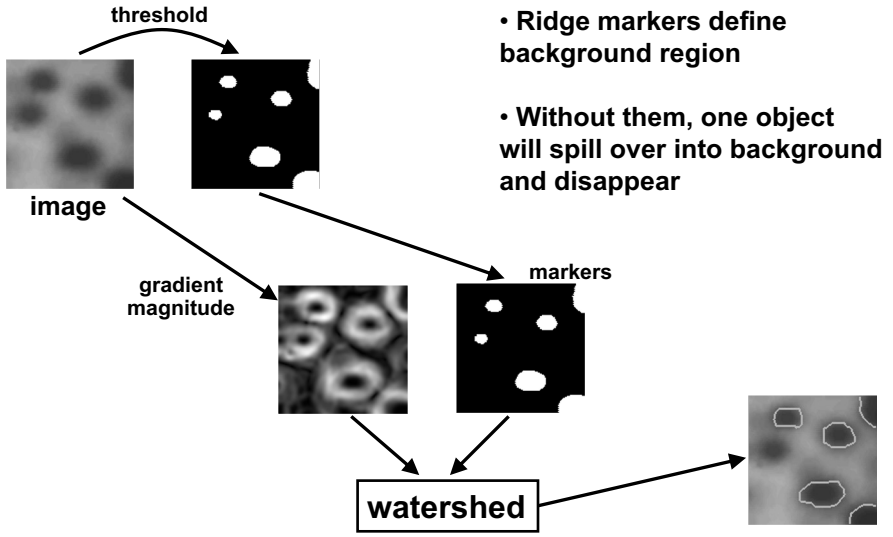
Marker-based watershed



S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

99

Why are ridges needed?

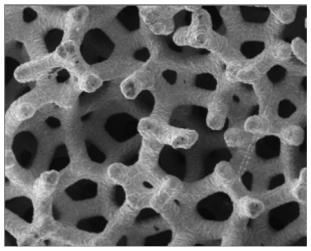


- Ridge markers define background region
- Without them, one object will spill over into background and disappear

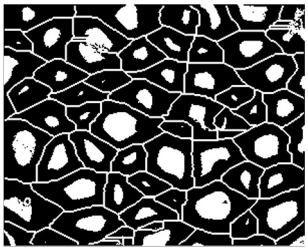
S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

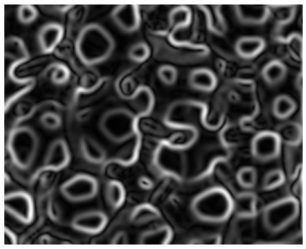
100

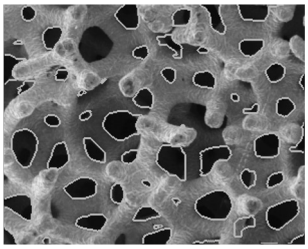
Another example



image


markers


gradient magnitude


watershed

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

101

Outline

- Human segmentation
- Standard algorithms:
  - Split-and-merge
  - Region growing
  - Minimum spanning tree
- Watershed algorithm
- Normalized cuts

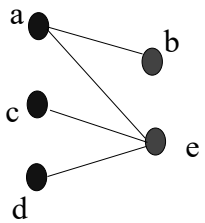
102

Graph theoretic clustering

- **Represent tokens (which are associated with each pixel) using a weighted graph.**
  - **affinity matrix (p_{ij} *has* affinity of 0)**
- **Cut up this graph to get subgraphs with strong interior links and weaker exterior links**

Application to vision originated with Shi and Malik at Berkeley

Graph Representations

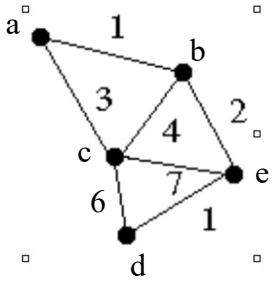


$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Adjacency Matrix: W

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Weighted Graphs



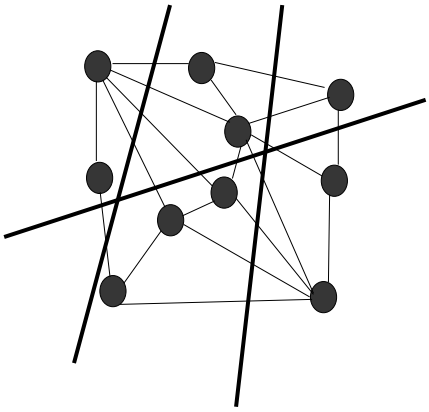
$$\begin{bmatrix} 0 & 1 & 3 & 0 & 0 \\ 1 & 0 & 4 & 0 & 2 \\ 3 & 4 & 0 & 6 & 7 \\ 0 & 0 & 6 & 0 & 1 \\ 0 & 2 & 7 & 1 & 0 \end{bmatrix}$$

Weight Matrix: W

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

105

Minimum Cut



A cut of a graph G is the set of edges S such that removal of S from G disconnects G.

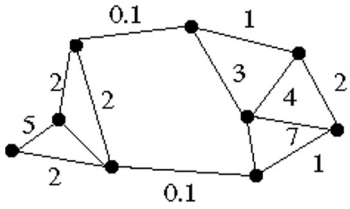
Minimum cut is the cut of minimum weight, where weight of cut <A,B> is given as

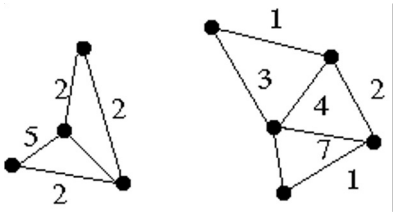
$$w(\langle A,B\rangle) = \sum_{x\in A, y\in B} w(x,y)$$

* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

106

Minimum Cut and Clustering

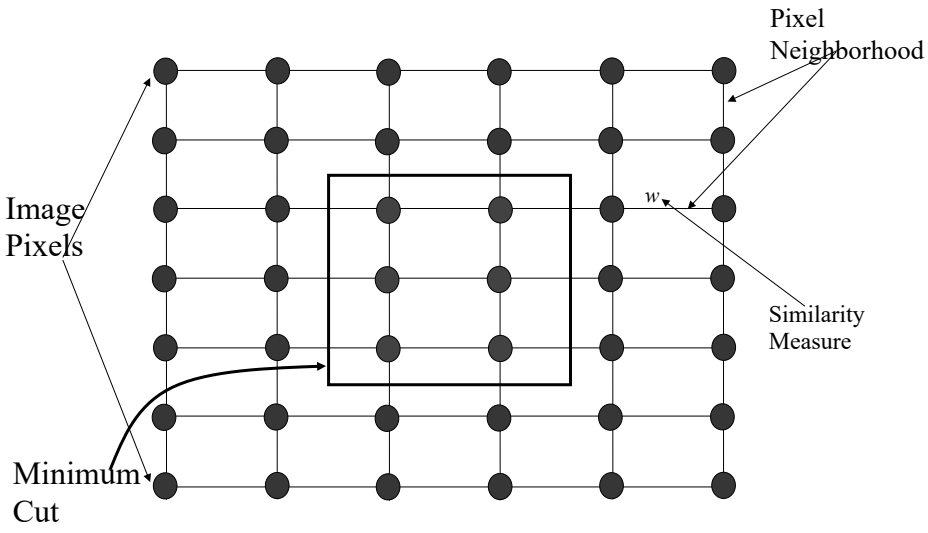




* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

107

Image Segmentation & Minimum Cut

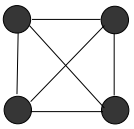


* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

108

Minimum Cut

- **There can be more than one minimum cut in a given graph**



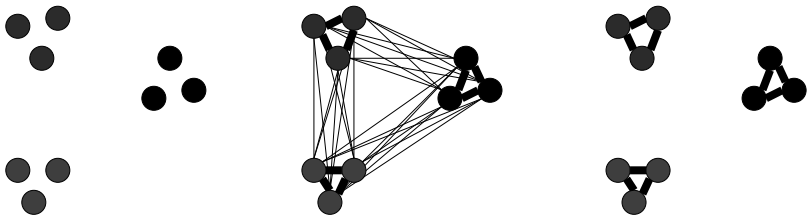
- **All minimum cuts of a graph can be found in polynomial time[1].**

[1]H. Nagamochi, K. Nishimura and T. Ibaraki, "Computing all small cuts in an undirected network. SIAM J. Discrete Math. 10 (1997) 469-481.

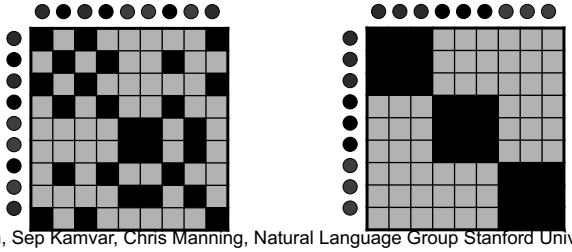
* From Khurram Hassan-Shafique CAP5415 Computer Vision 2003

109

Finding the Minimal Cuts: Spectral Clustering Overview



Data Similarities Block-Detection



* Slides from Dan Klein, Sep Kamvar, Chris Manning, Natural Language Group Stanford University

110

Eigenvectors and Blocks

- **Block matrices have block eigenvectors:**

$\lambda_1 = 2$ $\lambda_2 = 2$ $\lambda_3 = 0$ $\lambda_4 = 0$

1	1	0	0
1	1	0	0
0	0	1	1
0	0	1	1

eigensolver

.71	0
.71	0
0	.71
0	.71

- **Near-block matrices have near-block eigenvectors: [Ng *et al*., NIPS 02]**

$\lambda_1 = 2.02$ $\lambda_2 = 2.02$ $\lambda_3 = -0.02$ $\lambda_4 = -0.02$

1	1	.2	0
1	1	0	-.2
.2	0	1	1
0	-.2	1	1

eigensolver

.71	0
.69	-.14
.14	.69
0	.71

* Slides from Dan Klein, Sep Kamvar, Chris Manning, Natural Language Group Stanford University

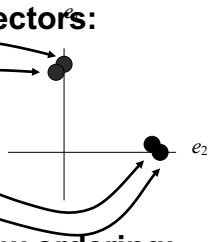
111

Spectral Space

- **Can put items into blocks by eigenvectors:**

1	1	.2	0
1	1	0	-.2
.2	0	1	1
0	-.2	1	1

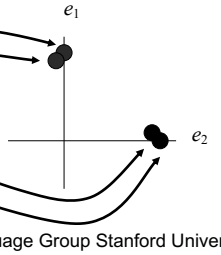
e_1	e_2
.71	0
.69	-.14
.14	.69
0	.71



- **Resulting clusters independent of row ordering:**

1	.2	1	0
.2	1	0	1
1	0	1	-.2
0	1	-.2	1

e_1	e_2
.71	0
.14	.69
.69	-.14
0	.71

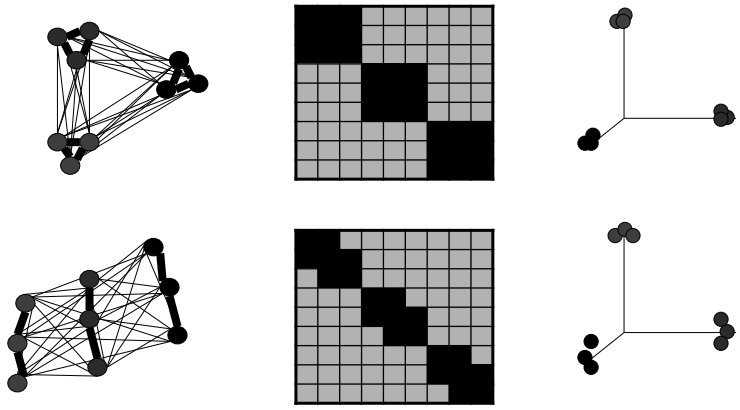


* Slides from Dan Klein, Sep Kamvar, Chris Manning, Natural Language Group Stanford University

112

The Spectral Advantage

- **The key advantage of spectral clustering is the spectral space representation:**

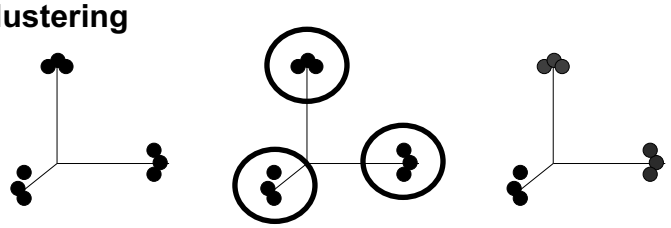


* Slides from Dan Klein, Sep Kamvar, Chris Manning, Natural Language Group Stanford University

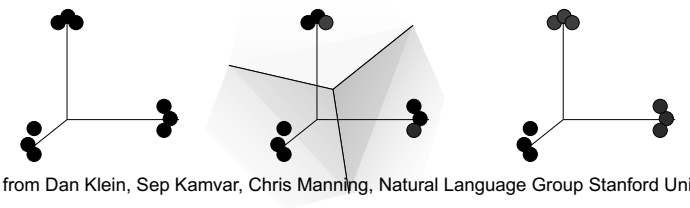
113

Clustering and Classification

- **Once our data is in spectral space:**
  - **Clustering**



  - **Classification**



* Slides from Dan Klein, Sep Kamvar, Chris Manning, Natural Language Group Stanford University

114

Measuring Affinity

Intensity

$$aff(x,y) = \exp\left\{-\left(1/2\sigma_i^2\right)\left(\|I(x)-I(y)\|^2\right)\right\}$$

Distance

$$aff(x,y) = \exp\left\{-\left(1/2\sigma_d^2\right)\left(\|x-y\|^2\right)\right\}$$

Texture

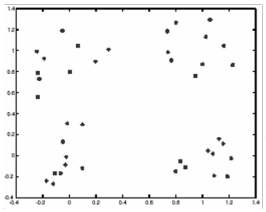
$$aff(x,y) = \exp\left\{-\left(1/2\sigma_t^2\right)\left(\|c(x)-c(y)\|^2\right)\right\}$$

* From Marc Pollefeys COMP 256 2003

115

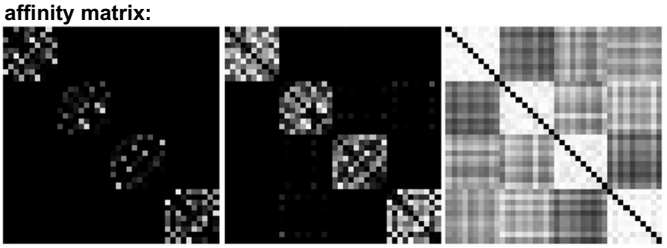
Scale affects affinity

Four isotropic Gaussians are sampled:

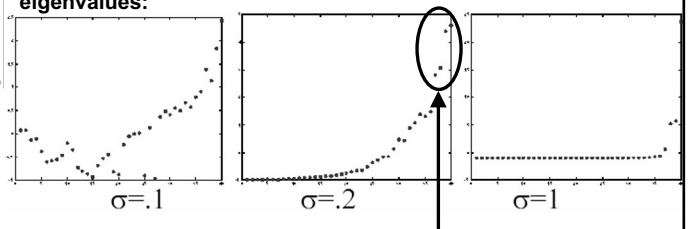


σ_{data}=0.2

affinity matrix:



eigenvalues:

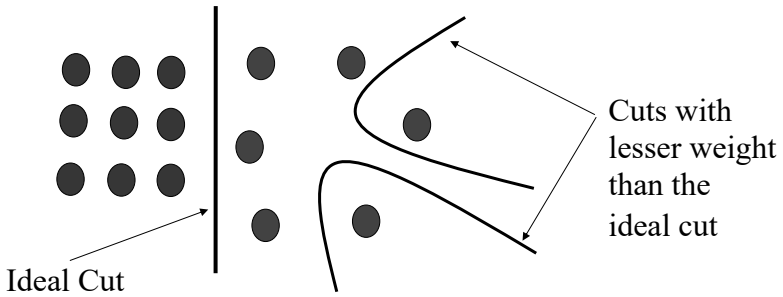


Four large eigenvalues

116

Drawbacks of Minimum Cut

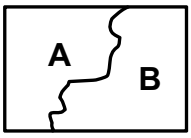
- **Weight of cut is directly proportional to the number of edges in the cut.**



* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

117

Graph cut



- **Goal is to split an image into two regions A and B**
- **The cost of the cut is the sum of the edge weights between the two regions:**

$$\xi(A,B) = \sum_{u \in A, v \in B} w(u,v)$$

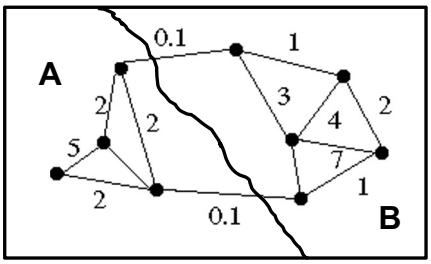
Note: We will assume undirected graph:

$$\xi(A,B) = \xi(B,A)$$

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

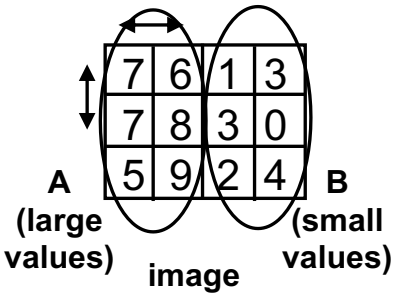
118

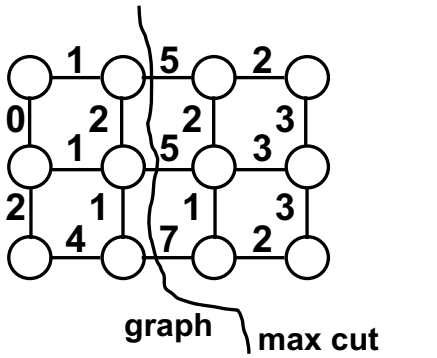
Example



$$\xi(A, B) = 0.2$$

An example





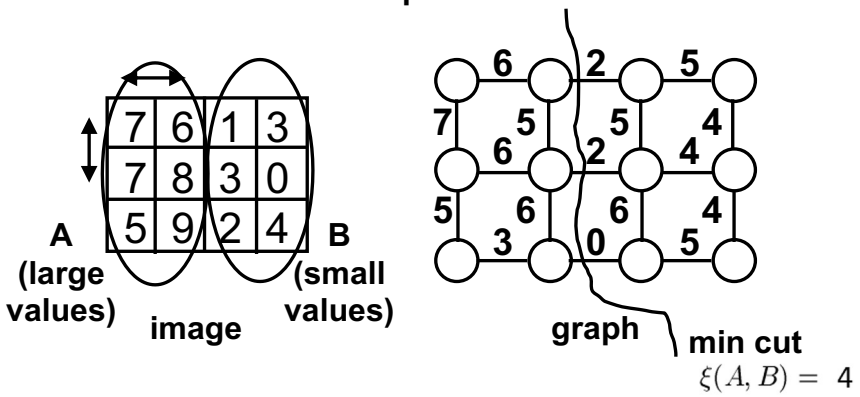
$$\xi(A, B) = 17$$

(using absolute difference in intensities)

But no one knows how to compute max cut…

An example (cont.)

So we flip values around…

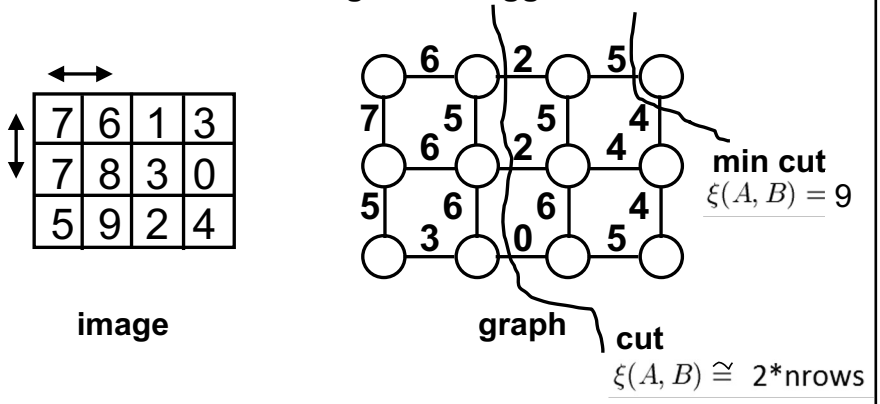


Here we flip using v' = 7-v,
but v'=exp(-v²/2σ²) is more common

121

An example (cont.)

But if image were bigger…

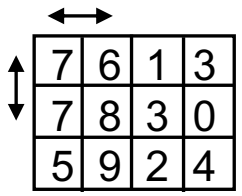


The min cut would favor the corners

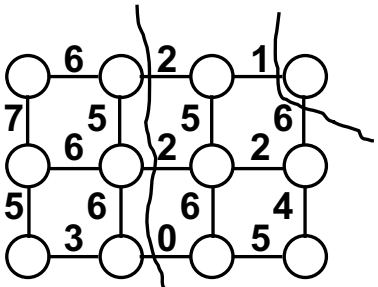
122

An example (cont.)

So we normalize…



image



graph

cut

min cut

$$\frac{\xi(A,B)}{|A|} + \frac{\xi(B,A)}{|B|} \cong \frac{2*\text{nrows}}{2*\text{nrows}} + \frac{2*\text{nrows}}{2*\text{nrows}} = 2$$

$$\frac{\xi(A,B)}{|A|} + \frac{\xi(B,A)}{|B|} \cong \frac{7}{4*\text{nrows}} + \frac{7}{1} \cong 7$$

Association

We approximate the region size with the *association*…

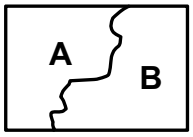
$$\xi(A,V)$$

which is the sum of the edge weights between pixels in A and all the pixels V

$$A \cup B = V$$

The larger the region, the larger the association

Normalized cuts



$\Omega = A \cup B$

- **To solve this problem, define the normalized cut:**

cut

$$\mathrm{Ncut}(A,B) = \frac{\xi(A,B)}{\xi(A,V)} + \frac{\xi(B,A)}{\xi(B,V)}$$

association

- **where**

$$\xi(A,B) = \sum_{u \in A, v \in B} w(u,v)$$

- **Here the cost of the cut is normalized by the total connection from the nodes in A and B to all the nodes in the graph**

J. Shi and J. Malik, Normalized Cuts and Image Segmentation, PAMI 2000

Normalized cuts

- **It is easy to see that**

$$\xi(A, B \cup C) = \xi(A,B) + \xi(A,C)$$

for any subgraphs A, B, and C

- **Therefore, assuming** $A \cup B = V$ **:**

$$\begin{aligned} \xi(A,V) &= \xi(A, A \cup B) \\ &= \xi(A,A) + \xi(A,B) \end{aligned}$$

$$\begin{aligned} \mathrm{Ncut}(A,B) &= \frac{\xi(A,B)}{\xi(A,V)} + \frac{\xi(A,B)}{\xi(B,V)} \\ &= \frac{\xi(A,V) - \xi(A,A)}{\xi(A,V)} + \frac{\xi(B,V) - \xi(B,B)}{\xi(B,V)} \\ &= 2 - \left[\frac{\xi(A,A)}{\xi(A,V)} + \frac{\xi(B,B)}{\xi(B,V)} \right] \end{aligned}$$

- → ***minimizing*** **the normalized** ***disassociation*** **b/w groups =** ***maximizing*** **the normalized** ***association*** **w/i groups**

J. Shi and J. Malik, Normalized Cuts and Image Segmentation, PAMI 2000

Laplacian matrix

- **What happens when you multiply a matrix by a binary vector on both sides?**

$$\mathbf{a}^T W \mathbf{a} = \begin{bmatrix} 1 & 0 & 1 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \boxed{w_{11}} & w_{12} & \boxed{w_{13}} & \cdots & w_{1N} \\ w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ \boxed{w_{31}} & w_{32} & \boxed{w_{33}} & \cdots & w_{3N} \\ & & \vdots & & \\ w_{N1} & w_{N2} & w_{N3} & \cdots & w_{NN} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{A}} w_{ij}$$

where $i \in \mathcal{A}$ iff $a_i = 1$

127

Laplacian matrix

- **Now define diagonal matrix *D* so that**
  $d_i = \sum_j w_{ij} = \sum_j w_{ji}$
  is the sum of row (or column) *i* of *W*,
  i.e., the total connection from node *i* to all other nodes (assuming W symmetric)
- **What happens now?**

$$\mathbf{a}^T D \mathbf{a} = \begin{bmatrix} 1 & 0 & 1 & \cdots & 0 \end{bmatrix} \begin{bmatrix} \boxed{d_1} & & & & \\ & d_2 & & & \\ & & \boxed{d_3} & & \\ & & & \ddots & \\ & & & & d_N \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \sum_{i \in \mathcal{A}} \sum_{j} w_{ij}$$

128

Laplacian matrix

- **Putting these together, we see**

$$\mathbf{a}^T(D-W)\mathbf{a} = \sum_{i\in\mathcal{A}}\sum_{j\notin\mathcal{A}} w_{ij}$$

$$= \sum_{i\notin\mathcal{A}}\sum_{j\in\mathcal{A}} w_{ij} = (1-\mathbf{a})^T(D-W)(1-\mathbf{a})$$

if *W* symmetric

$$\begin{array}{c|cccccc} & 1 & 0 & 1 & \cdots & 0 \\ 1 & w_{11} & w_{12} & w_{13} & \cdots & w_{1N} \\ 0 & w_{21} & w_{22} & w_{23} & \cdots & w_{2N} \\ 1 & w_{31} & w_{32} & w_{33} & \cdots & w_{3N} \\ \vdots & & & \vdots & & \\ 0 & w_{N1} & w_{N2} & w_{N3} & \cdots & w_{NN} \end{array}$$

- ***D-W* is called the *Laplacian matrix***
- **It is always positive semidefinite (→ non-negative eigenvalues)**

Normalized cuts

- **Straightforward substitution:**

$$\begin{aligned} \mathrm{Ncut}(A,B) &= \frac{\xi(A,B)}{\xi(A,V)} + \frac{\xi(B,A)}{\xi(B,V)} \\ &= \frac{\sum_{i\in\mathcal{A},j\notin\mathcal{A}} w_{ij}}{\sum_{i\in\mathcal{A}}\sum_j w_{ij}} + \frac{\sum_{i\notin\mathcal{A},j\in\mathcal{A}} w_{ij}}{\sum_i\sum_{j\notin\mathcal{A}} w_{ij}} \\ &= \frac{\mathbf{a}^T(D-W)\mathbf{a}}{\mathbf{a}^T D\mathbf{a}} + \frac{(\mathbf{1}-\mathbf{a})^T(D-W)(\mathbf{1}-\mathbf{a})}{(\mathbf{1}-\mathbf{a})^T D(\mathbf{1}-\mathbf{a})} \end{aligned}$$

- **If we let**

$$\mathbf{y} = \mathbf{a} - b(\mathbf{1}-\mathbf{a}) \qquad b = \frac{\sum_{i\in\mathcal{A}} d_i}{\sum_{i\notin\mathcal{A}} d_i} = \frac{\mathbf{a}^T D\mathbf{a}}{(\mathbf{1}-\mathbf{a})^T D(\mathbf{1}-\mathbf{a})}$$

$$y_i \in \{1, -b\}$$

- **… more derivation yields:**

$$\min_{\mathbf{a}} \mathrm{Ncut}(A,B) = \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}$$

Normalized cuts

- Solving the discrete problem is NP-hard
- Solution: Allow *y* to take on real values
- Then, to minimize Rayleigh quotient,

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}$$

solve generalized eigenvalue system:

$$(D-W)\mathbf{y} = \lambda D\mathbf{y}$$

for eigenvector with minimum eigenvalue

$$\lambda_n \geq \cdots \geq \lambda_2 \geq \lambda_1 \geq 0$$

Normalized cuts

- But note that smallest eigenvalue will always be zero ($\lambda_{\square}$=0 and y_1=1), in which case all pixels are labeled A, and no pixels are labeled B
- Why?

$$(D-W)\mathbf{1} = D\mathbf{1} - W\mathbf{1} = \mathbf{0}$$

since $W\mathbf{1}$ sums each row of W, which is the definition of D

- To avoid this trivial solution, we use second smallest eigenvalue by introducing constraint:

$$\min_{\mathbf{y}\,:\,\mathbf{y}^T D\mathbf{1}=0} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}$$

- This constraint is automatically satisfied if $\mathbf{y} \neq \mathbf{1}$:

$$\mathbf{y}^T D\mathbf{1} = (\mathbf{a} - b(\mathbf{1}-\mathbf{a}))^T D\mathbf{1} = \mathbf{a}^T D\mathbf{1} - b(\mathbf{1}-\mathbf{a})^T D\mathbf{1}$$
$$= \sum_{i\in\mathcal{A}} d_i - b\sum_{i\notin\mathcal{A}} d_i = \sum_{i\in\mathcal{A}} d_i - \sum_{i\in\mathcal{A}} d_i = 0$$

Why does constraint work?

- Convert generalized eigenvalue system into standard eigenvalue system:

$$D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}\mathbf{z} = \lambda\mathbf{z} \qquad \text{where} \quad \mathbf{z} = D^{\frac{1}{2}}\mathbf{y}$$

- Note that eigenvector with zero eigenvalue is always given by

$$\mathbf{z}_1 = D^{\frac{1}{2}}\mathbf{1}$$

- Therefore, eigenvector with zero eigenvalue of generalized system is

$$\mathbf{y}_1 = D^{-\frac{1}{2}}\mathbf{z}_1 = \mathbf{1}$$

- Eigenvectors of symmetric matrix are mutually orthogonal:

$$\mathbf{z}_i^T\mathbf{z}_1 = 0 \text{ for any } i > 1$$

- Therefore (any other, smallest)

$$\mathbf{y}_i^T D\mathbf{1} = 0$$

is true for any non-trivial eigenvector. By minimizing with this constraint, we get the minimum non-trivial eigenvector (i.e., the one with the second smallest eigenvalue)

Normalized cuts

Algorithm:

- Set up weighted graph G=(V,E) with vertices as pixels and edges between neighboring pixels
- Set weights to be a measure of similarity between nodes (color, texture, motion, proximity, …)
- Solve (D-W)y = λ D y for smallest eigenvectors (e.g., Lanczos method)
- Use eigenvector with second smallest eigenvalue to bipartition graph (by thresholding eigenvector) *Note: second smallest eigenvalue of (D-W) is known as Fiedler value*
- Recursively partition the segmented regions if necessary until all components are found

Example Results

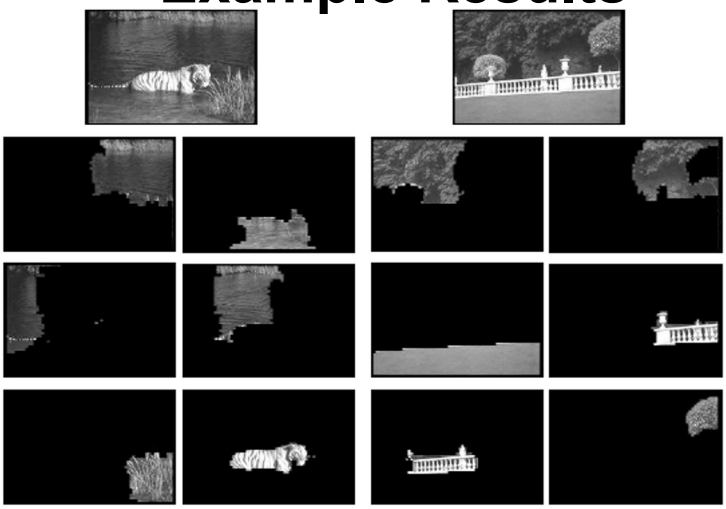


Figure from “Image and video segmentation: the normalised cut framework”, by Shi and Malik, 1998

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

135

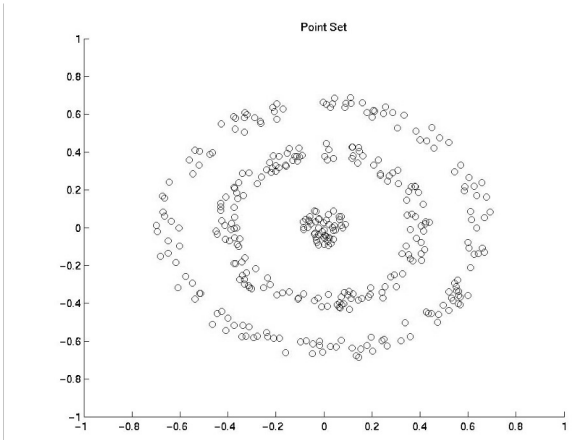
More Results

F igure from “Normalized cuts and image segmentation,” Shi and Malik, 2000

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

136

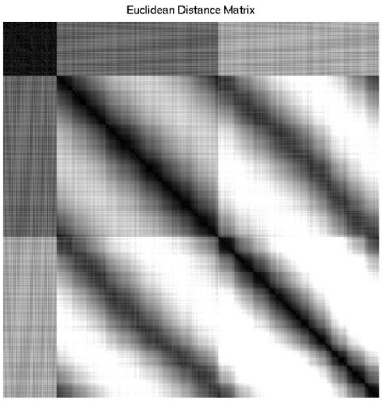
Example I



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

137

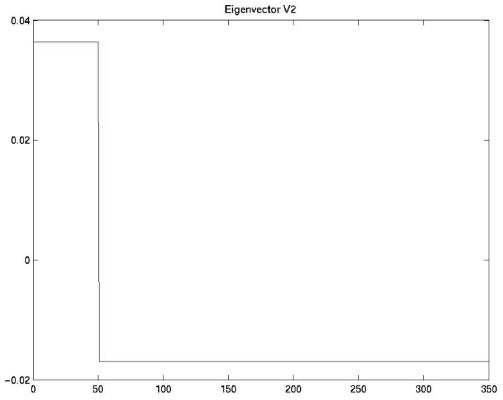
Distance Matrix



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

138

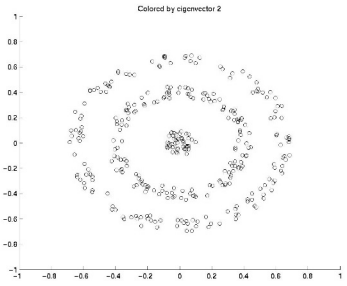
The second generalized eigenvector



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

139

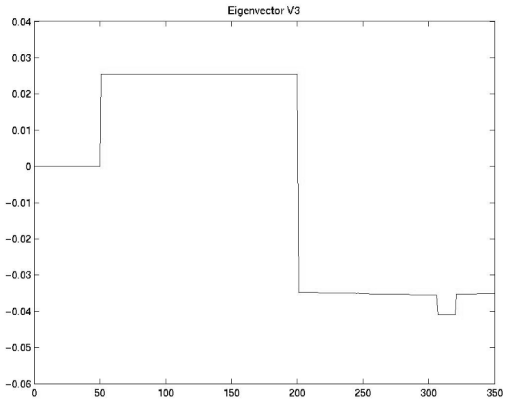
The first partition



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

140

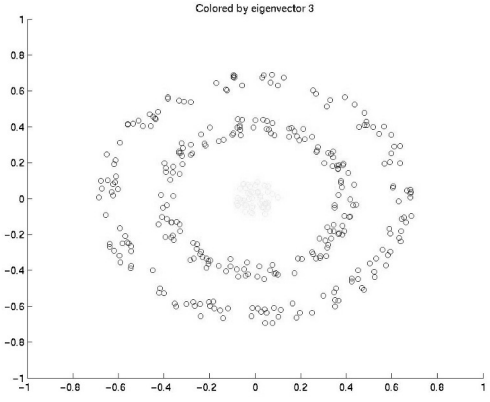
The second generalized eigenvector



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

141

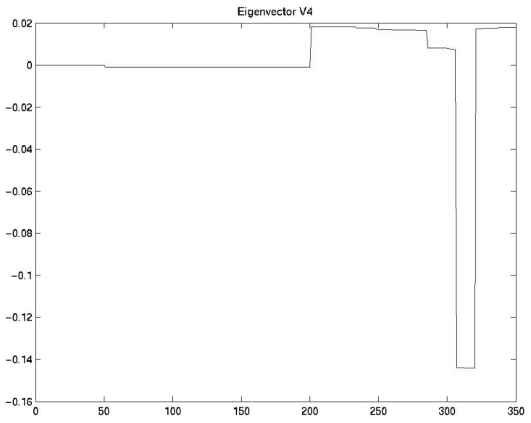
The second partition



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

142

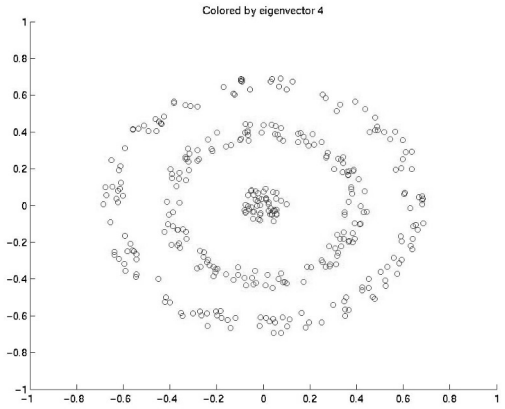
The fourth generalized eigenvector



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

143

The third partition



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

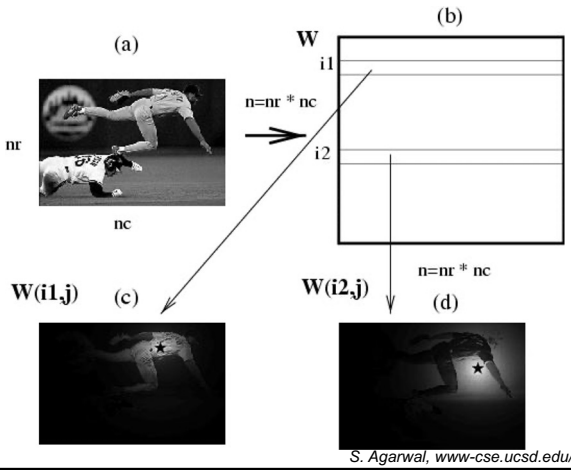
144

Example II

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

145

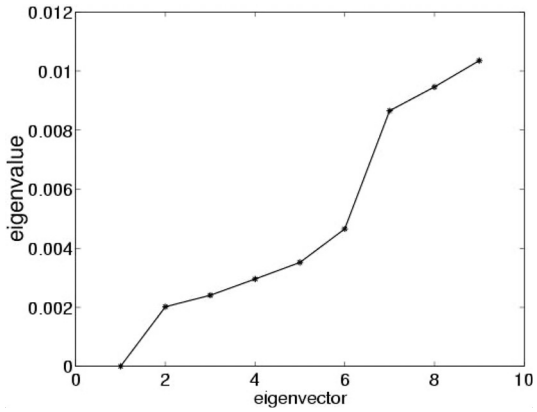
The structure of the affinity matrix



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

146

Generalized eigenvalues



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

147

The first partition

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

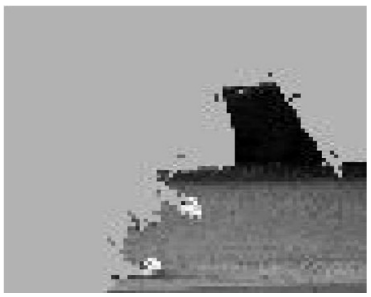
148

The second partition

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

149

The third partition



S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

150

The fourth partition

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

151

The fifth partition

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

152

Relationship between Average, Ratio, and Normalized Cuts

Finding Clumps ⟷ Finding Splits

	Average Association	Normalized Cut	Average Cut
Discrete Formulation	**Assoc(A,A)/|A| + Assoc(B,B)/|B|**	**Cut(A,B)/assoc(A,V)+ Cut(A,B)/assoc(B,V) = 2 – (assoc(A,A)/assoc(A,V) + assoc(B,B)/assoc(B,V))**	**Cut(A,B)/|A| + Cut(A,B)/|B|**
Continuous Formulation	**$Wx=\lambda x$**	**$(D-W)x=\lambda Dx$**	**$(D-W)x=\lambda x$**

S. Agarwal, www-cse.ucsd.edu/classes/fa01/cse291/eigen.ppt

153

Other topics in normalized cuts

- **Nystrom method (Fowlkes et al. 2004)**
  - **In practice, the affinity matrix has low rank**
  - **The eigenvectors of the matrix can be approximated by linearly interpolating the eigenvectors of a randomly sampled submatrix**
  - **Greatly improves performance**
- **Swendsen-Wang cuts (Barbu and Zhu 2005)**
  - **Probabilistic way of turning off edges**
  - **Allows more general energy functionals**

S. Birchfield, Clemson Univ., ECE 847, http://www.ces.clemson.edu/~stb/ece847

154