Binary Shape Analysis

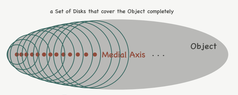
Medial Axis & Distance Transform

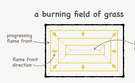
My Lecture Notes - Spring '19 edited for Spring '20 Class

before we start talking about Distance Transform we need to cover Medial Axis

Medial Axis of Objects

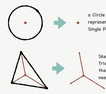
Def. The **Medial Axis (Skeleton, Grass-Fire)** of an object is the collection of the centers of a set of disks that cover the object completely s/t no single disk in the set can be entirely covered by any other disk from the same set.

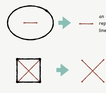


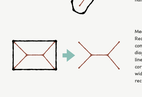


Medial Axis

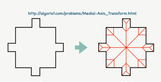
- Medial Axis is used for Image Compression and Object Recognition
- We can compress/represent the entire object using just a set of points (medial axis)

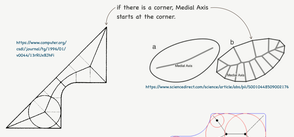


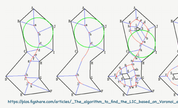


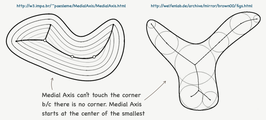


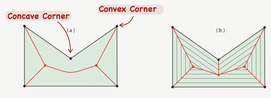
my drawings above are based on: http://cns-alumni.bu.edu/~slehar/ConstructiveAspect/ConstructiveAspect.html

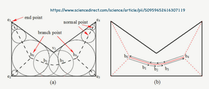


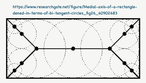








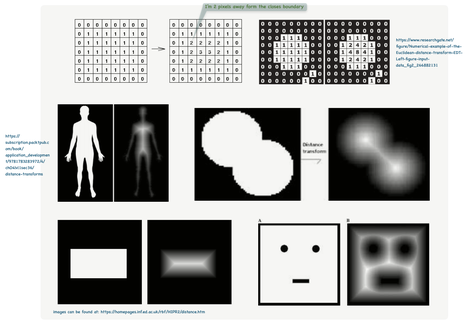




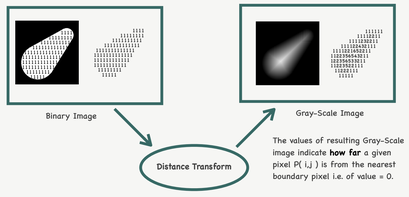
Distance Transform

Def. The **Distance Transform** is an operator normally only applied to Binary images. The result of the transform is a Graylevel image that looks similar to the input image, except that the Graylevel intensities of points inside foreground regions are changed to show the distance to the closest boundary from each point. https://homepages.inf.ed.ac.uk/rbf/HIPR2/distance.htm

Distance Transform (image processing) is used for Image Compression and Object Recognition. When Compressing an image we extract the Medial Axis and use it to compress the data in the same time - lossless compression.



to **Transform** means to change set-1 into set-2



The values of resulting Gray-Scale image indicate **how far** a given pixel P(i,j) is from the nearest boundary pixel i.e. of value = 0.

We only need 2 passes

Distance Transform Algorithm

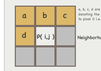
Algorithm — Distance Transform Algorithm

Pass-1 1

Step 0: image ← given Binary Image

Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel

Step 2: if P(i,j) > 0 //is an object pixel
look at neighbors: a, b, c, d



Calculate distance to the nearest boundary (pixel=0) by evaluating the Neighborhood and applying the chosen Distance Template

since we do NOT need the old values we write directly to the image — Gray-Scale values

8-Connected Distance → P(i,j) ← min(a, b, c, d) + 1
(or min(a+1, b+1, c+1, d+1))

City-Block Distance → P(i,j) ← min(a+2, b+1, c+2, d+1)

Euclidean Distance → P(i,j) ← min(a+√2, b+1, c+√2, d+1)

Step 3: repeat steps 1 to 2 until all pixels are processed

Given a Binary Image - first we need to compute the distance to the nearest boundary pixel (i.e. pixel of value 0)

Pass-2 2

Step 0: image ← given the result of Pass-1

Step 1: Scan image Right-Left & Bottom-Top
P(i,j) ← next pixel

Step 2: if P(i,j) > 0 //is an object pixel
look at neighbors: e, f, g, h and P(i,j)

Calculate distance to the nearest boundary (pixel=0) by evaluating the Neighborhood and applying the chosen Distance Template

since we do NOT need the old values we write directly to the image — min(e+1, f+1, g+1, h+1 , P(i,j))

8-Connected Distance → P(i,j) ← min(e+1, f+1, g+1, h+1 , P(i,j))

City-Block Distance → P(i,j) ← min(e+1, f+2, g+1, h+2, P(i,j))

Euclidean Distance → P(i,j) ← min(e+1, f+√2, g+1, h+√2, P(i,j))

Step 3: repeat steps 1 to 2 until all pixels are processed

Local Maxima Operation 3

Step 0: image ← given the result of Pass-2
skeleton ← array of same size at image

Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel

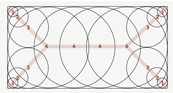
Step 2: check if P(i,j) is a Local Maxima:

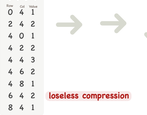
Since the Distance is already computed, look at all the neighbors using either 8-Connectedness (or 4-Connectedness) to find the Local Maxima:

P(i,j) is a Local Maxima:
iff P(i,j) >= a, b, c, d, e, f, g, h
else skeleton(i,j) ← P(i,j) //retain the distance
skeleton(i,j) ← 0

Step 3: repeat steps 1 to 2 until all pixels are processed

So far we only got the Distances, but how do we get the Skeleton (i.e. Medial Axis) ?





loseless compression

Skeleton Image Compression 4

Step 0: image ← given the result of Local Maxima Operation
output ← file for result of compression

Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel

Step 2: if P(i,j) > 0 //is a Local Maxima
output ← i j P(i,j)

Step 3: repeat steps 1 to 2 until all pixels are processed

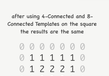
Neighboring Distance Templates

City-Block Distance Template (4-Connected Distance)

Euclidean Distance Template (the most accurate)

8-Connected Distance Template

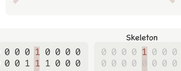
after using 4-Connected and 8-Connected Templates on the square the results are the same

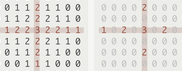


after using 4-Connected Template on a diamond shape

after using 8-Connected Template on a diamond shape

results of applying Distance Transform Alg.





Expansion Pass 1 5

Step 0: image ← load the File rendered by Skeleton Compression

Step 1: Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel

Step 2: if P(i,j) == 0
look at neighbors of P(i,j): a, b, c, d, e, f, g, h

Subtract chosen Distance Template:
max = max(a,b,c,d,e,f,g,h) - 1

8-Connected Distance → max ← max(a-1, b-1, c-1, d-1, e-1, f-1, g-1, h-1)

City-Block Distance → max ← max(a-2, b-1, c-2, d-1, e-1, f-2, g-1, h-2)

Euclidean Distance → max ← max(a-√2, b-1, c-√2, d-1, e-1, f-√2, g-1, h-√2)

if P(i,j) < max
P(i,j) ← max

Step 3: repeat steps 1 to 2 until all pixels are processed

decompression

Threshold Decompressed File 7

Step 0: output ← open file for the decompressed image
image ← result of Expansion Pass-2

Step 1: write the original image header to the de-compressed file
output ← numRows , numCols , minVal , maxVal

Step 2: Threshold the image
Scan image Left-Right & Top-Bottom
P(i,j) ← next pixel

Step 3: if P(i,j) > 0
output ← 1 and a blank space
else
output ← 0 and a blank space

Step 4: repeat steps 2 to 3 until all pixels are processed

Expansion Pass 2 6

Step 0: image ← rendered by Expansion Pass 1

Step 1: Scan image Right-Left & Bottom-Top
P(i,j) ← next pixel

Step 2: look at neighbors of P(i,j): a, b, c, d, e, f, g, h

8-Connected Distance → max ← max(a, b, c, d, e, f, g, h)

City-Block Distance → max ← max(a-2, b-1, c-2, d-1, e-1, f-2, g-1, h-2)

Euclidean Distance → max ← max(a-√2, b-1, c-√2, d-1, e-1, f-√2, g-1, h-√2, P(i,j))

if P(i,j) < max
P(i,j) ← max-1

Step 3: repeat steps 1 to 2 until all pixels are processed

note: Pass-2 will relabel some of the previously labeled in Pass-1 pixels (that are greater than zero) and label those 0 pixels that Pass-1 did NOT label

Thinning

Thinning - thick Edges can be made 1 pixel thin.

Algorithm — Thinning Algorithm Steps

Step 0 img ← given Binary Image

Step 1 thin - North (1 layer)

Step 2 thin - South (1 layer)

Step 3 thin - West (1 layer)

Step 4 thin - East (1 layer)

Step 5 repeat steps 1 to 4 until no more pixels change from 1 to 0

Algorithm — Thin North Algorithm Steps (replace North with South, West or East as desired)

Step 1 Scan img Left-Right & Top-Bottom

Step 2 P(i,j) - get next pixel

Step 3 if P(i,j) > 0

change P(i,j) to 0 under the following conditions:

(1) North Neighbor of P(i,j) is 0
(2) P(i,j) has at least **3** object neighbors (i.e. > 0)
(3) by flipping P(i,j) to 0 will not create more than one Connected Component in P's 3x3 neighborhood

Use 3 for West and East
Use 4 for North and South

Step 4 repeat steps 2 to 3 untill all pixels are processed

Def. **Thinning** is a morphological operation that is used to remove selected foreground pixels from binary images, somewhat like Erosion or Opening. It can be used for several applications, but is particularly useful for **Skeletonization**. Thinning is commonly used to tidy up the output of edge detectors by reducing all lines to single pixel thickness. Thinning is normally only applied to binary images, and produces another binary image as output.

Object-Process diagram - Thinning

Distance Transform Object-Process diagram

Notes are based on the lectures given by Dr. Tsaiyun Phillips in Spring 2019

Did you find a spelling error, grammar mistake, or any other discrepancy? Send an email to mdz.zone@iCloud.com