Skeletonization by distance

Team members: Carlos Reátegui, Fabrizio Franco, Christian Ledgard and Maor Roizman.

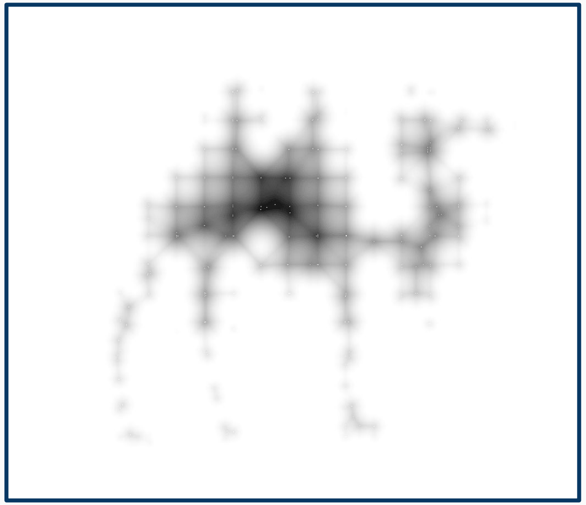
Teacher: Bryan Gonzales.

Subject: Analysis And Design of Algorithms.

Date: November 4, 2020.

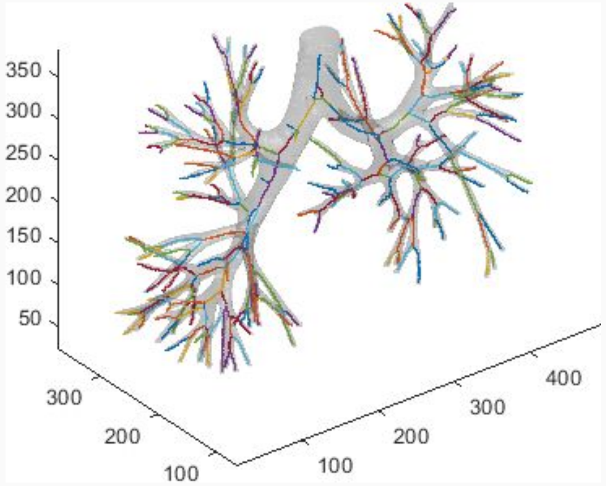
Overview

This article analyzes theoretically and empirically different algorithms for the calculation of the distance transform map, as well as it makes a theoretical comparison between two algorithms for skeletonization.



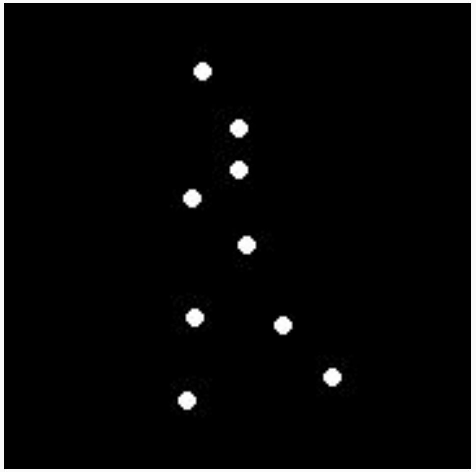
What is a Skeletonization process ?

- Process of “**peeling off” a pattern as many pixels as possible** without affecting the general shape of the pattern.
- Pattern **should still be recognizable** after pixels have been peeled of
- Also known as **Medial Axis**.



Problem description

Find the minimum relevant structure for an object that, despite eliminating fragments of it, manages to maintain its main properties morphologically and topologically.



Applications

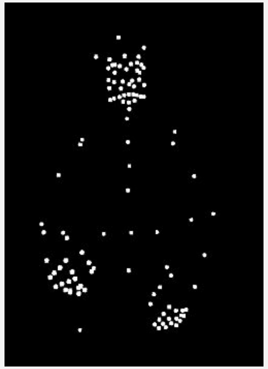
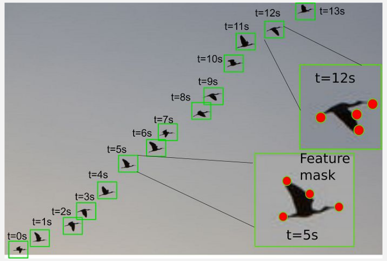
Object tracking

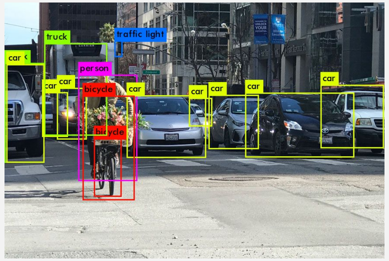
Object recognition

File compression

Object representation

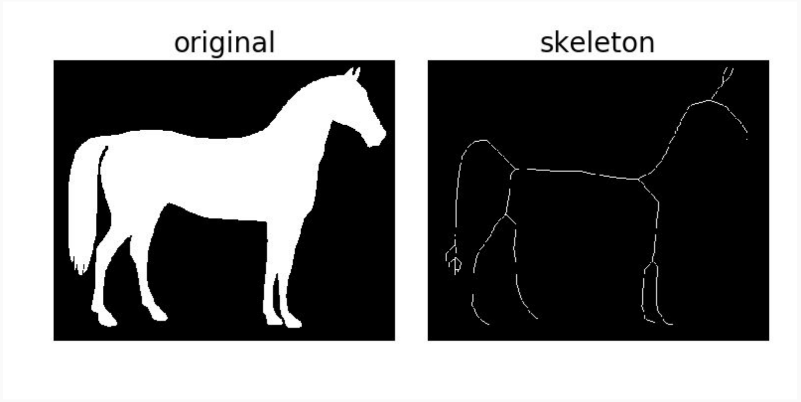
Medical imaging



A good Skeletonization Algorithm

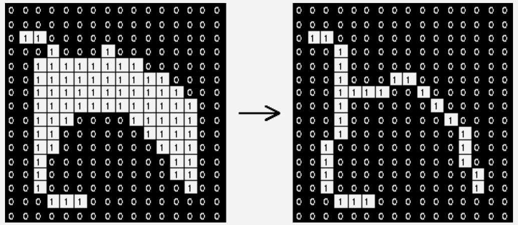
1) Preserving connectivity of the skeleton.
2) Converging to skeleton of unit width.
3) Preserving the original topology.
4) Locating at the geometrical center of the object image.



Related works

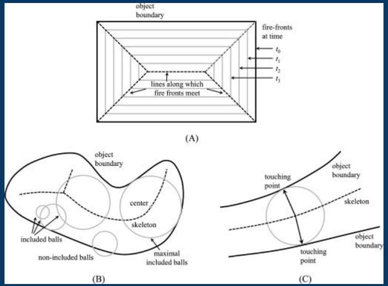
Skeletonization by Thinning

“Reduce the object to an approximate line”



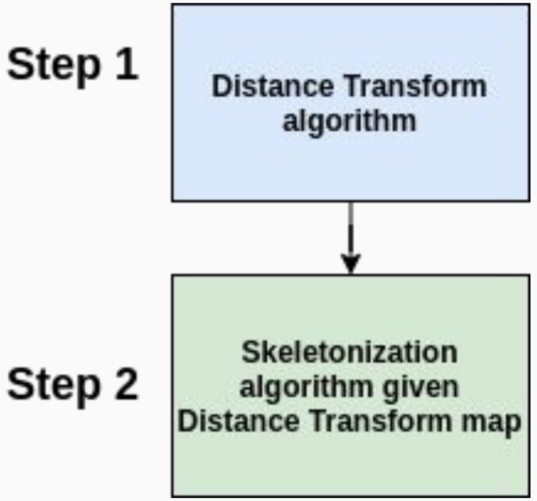
Skeletonization based on mathematical morphology

“Symmetric point distance from a skeleton to the boundary and maximal disks inscribable inside a filled-in image object”



[3] & [4]

Schematic

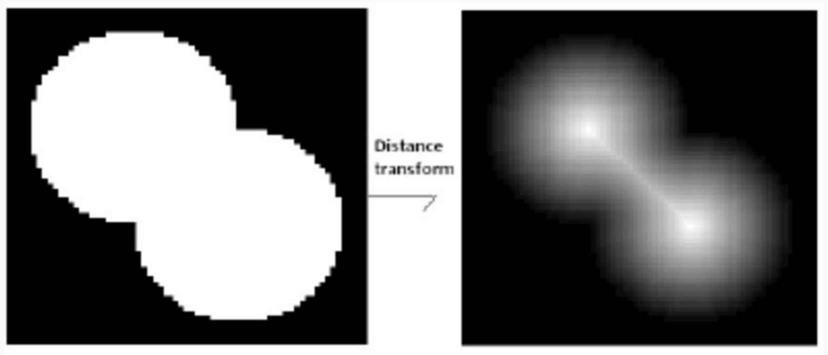

Step 1
Distance Transform algorithm
Step 2
Skeletonization algorithm given Distance Transform map

[3] & [4]

Distance Transform algorithms

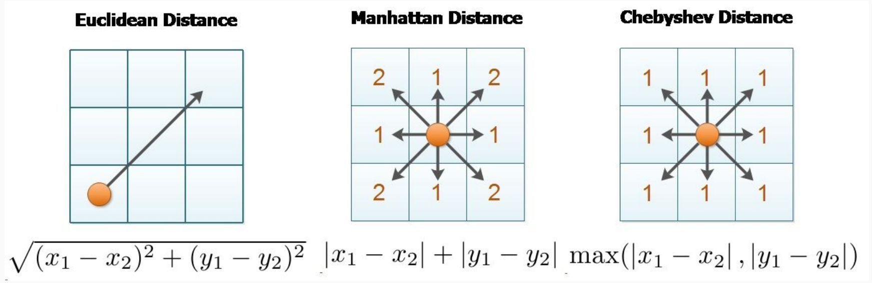
Distance Transform

- Operator applied to binary images
- Operations like skeletonization rely on distance maps (result)



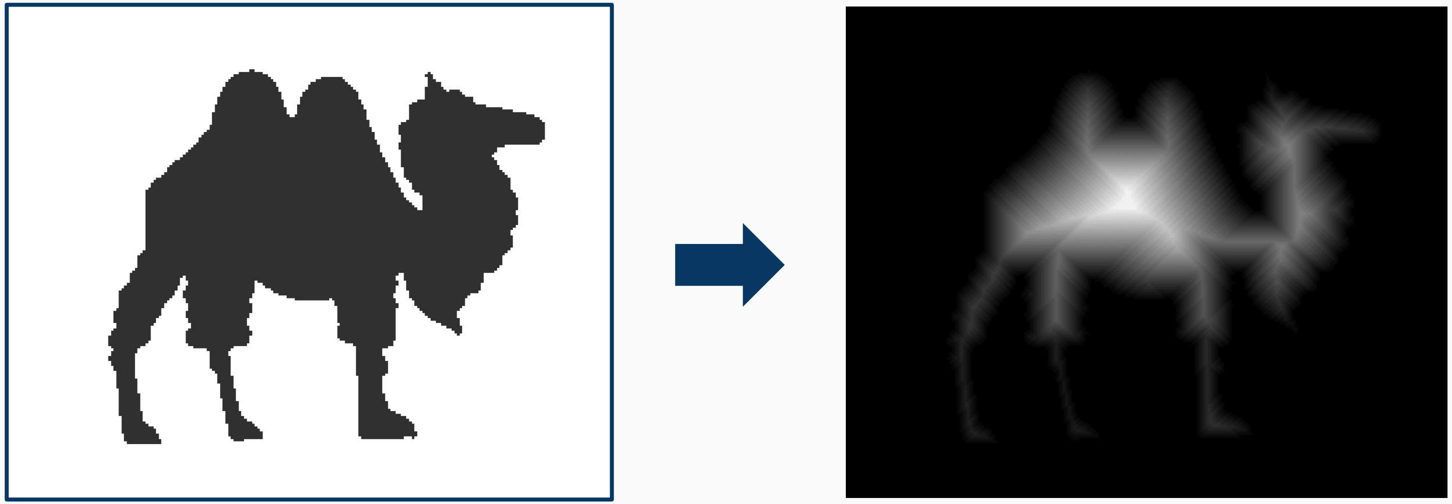
Algorithm Comparison

First Approach	Second Approach
• **Neighbors**: 8	• **Neighbors**: 4
• Can be generalized into various types of distances:	• It is optimized for one type of distance:
○ Manhattan	○ Chebyshev
○ Euler	
○ Chebyshev	

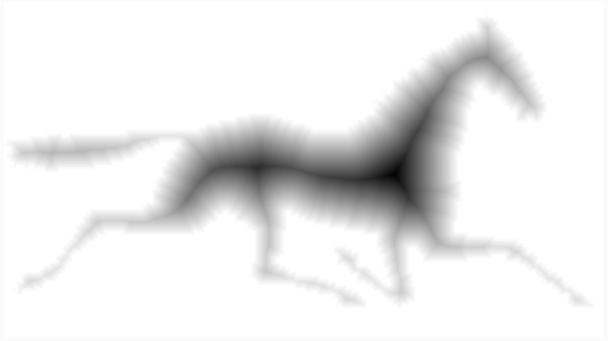


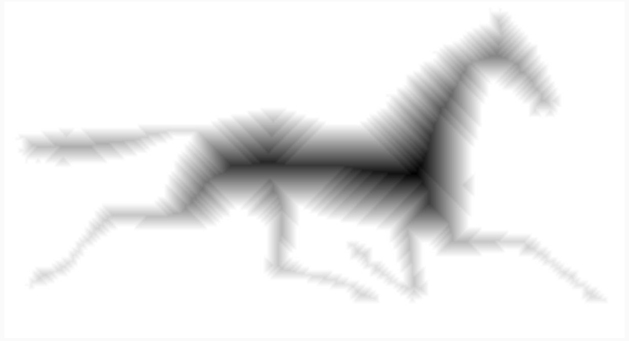
$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ $|x_1 - x_2| + |y_1 - y_2|$ $\max(|x_1 - x_2|, |y_1 - y_2|)$

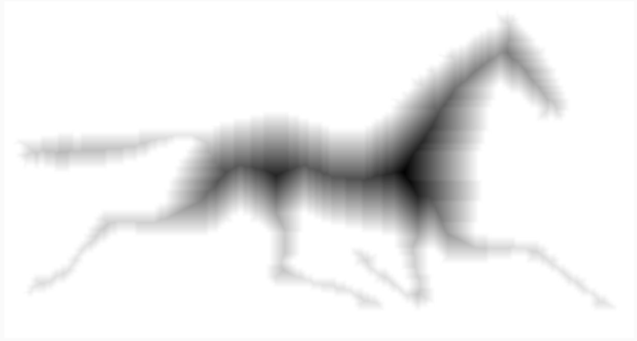
Example



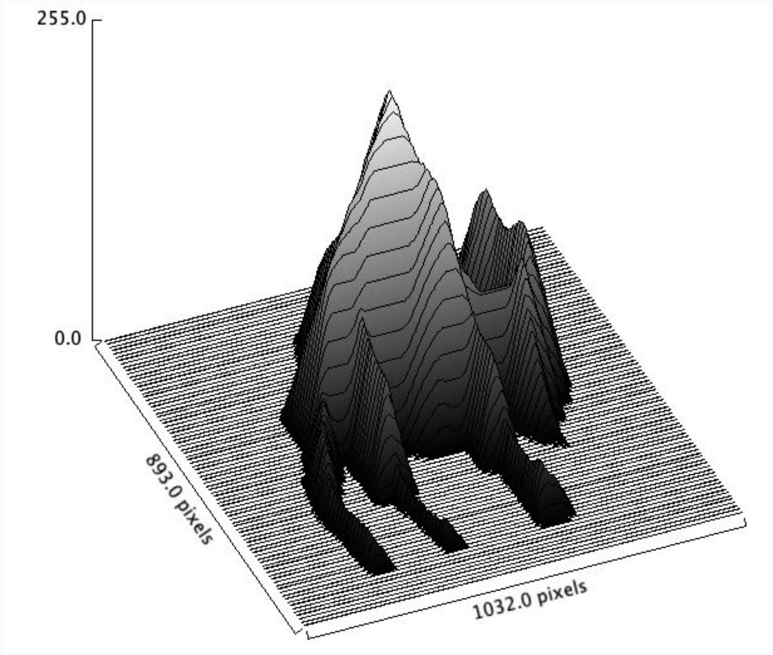
Algorithm Comparison


Euclidean Distance


Chebyshev Distance


Manhattan Distance

Surface map



ImageJ

Algorithm Comparison

Theoretical analysis

First Approach

- **Complexity**: T(n) = O(nlg(n))

```
while pixels_queue has elements:
    pixel = pixels_queue.pop
    for neighbor in eight_neighbors(pixel):
        distance = distance between borders(p) and neighbor
        if distance < distances(neighbor):
            distances(p) = distance
            borders(neighbor) = borders(pixel)
            if neighbor is not in pixels_queue:
                insert neighbor to pixels_queue
```

Second Approach

- **Complexity**: T(n) = O(n)

```
firstPassDistance()
  for i = 0 to rows:
    for j = 0 to columns:
      if image[i][j] > 0:
        loadNeighborTop(i,j) //Load the top three neighbors of the pixel.
        image[i][j] = min() + 1

secondPassDistance()
  for i = rows to 0:
    for j = columns to 0:
      if image[i][j] > 0:
        loadNeighborBottom(i,j) //Load the bottom three neighbors of the pixel.
        image[i][j] = minSecond()
        if image[i][j] < newMinValue:
          newMinValue = image[i][j]
        if image[i][j] > newMaxValue:
          newMaxValue = image[i][j]
```

Algorithm Comparison

Theoretical analysis

First Approach

- **Complexity**: T(n) = O(nlg(n))

➔ Let n be the number of pixels the input image and T(n) be
the execution time of this algorithm.

➔ An insertion is made to the pixels queue in line 9; since the pixel queue is constructed as a priority queue (heap) and can have at most n elements, this operation takes O(lg(n)).

➔ The while loop of line 1, executes while the pixel queue is not empty.

Second Approach

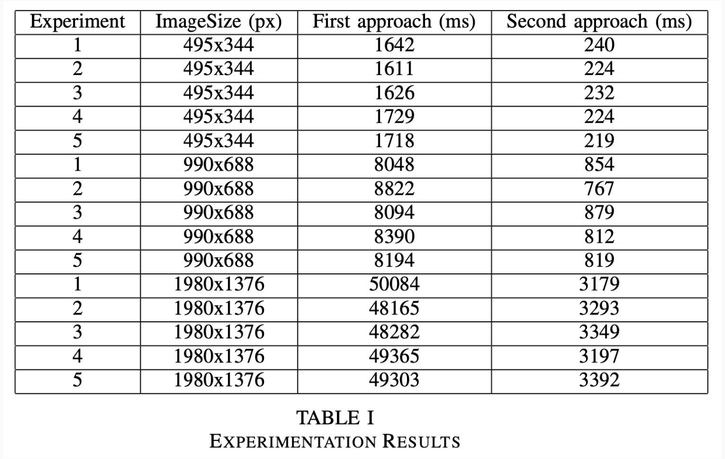
- **Complexity**: T(n) = O(n)

➔ Let n be the total number of pixels of an image, and T(n) be the running time of the algorithm.

➔ Both passes iterate through each pixel of the image. Thus, T(n) = O(n).

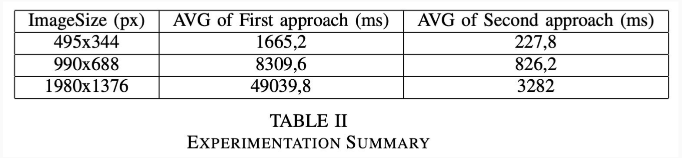
Experiments and Results

Empirical analysis



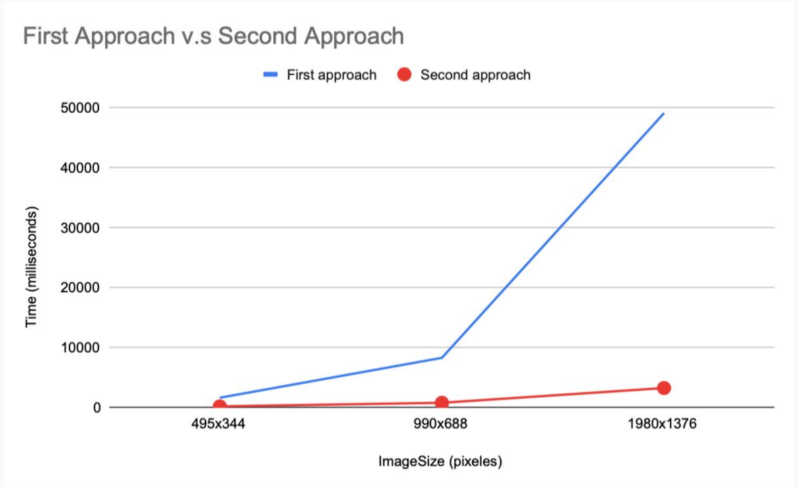
Experiment	ImageSize (px)	First approach (ms)	Second approach (ms)
1	495x344	1642	240
2	495x344	1611	224
3	495x344	1626	232
4	495x344	1729	224
5	495x344	1718	219
1	990x688	8048	854
2	990x688	8822	767
3	990x688	8094	879
4	990x688	8390	812
5	990x688	8194	819
1	1980x1376	50084	3179
2	1980x1376	48165	3293
3	1980x1376	48282	3349
4	1980x1376	49365	3197
5	1980x1376	49303	3392

TABLE I
EXPERIMENTATION RESULTS



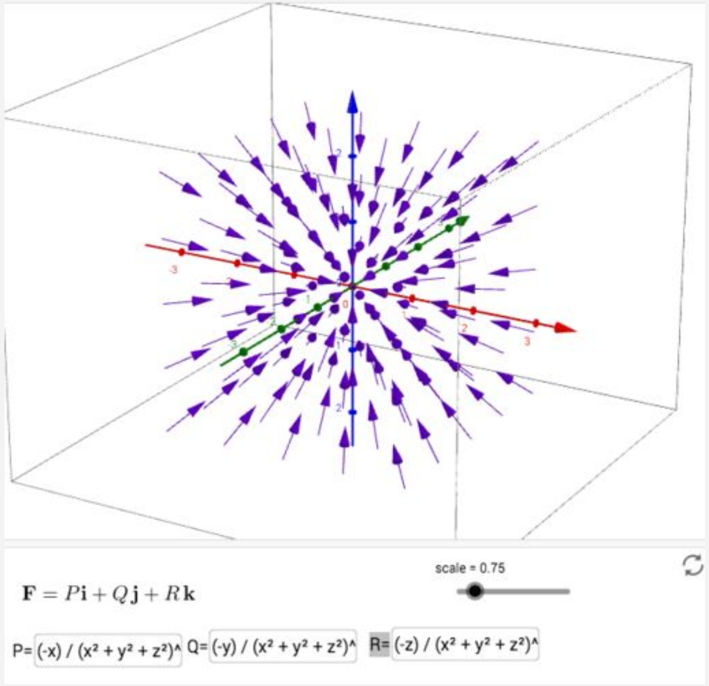
ImageSize (px)	AVG of First approach (ms)	AVG of Second approach (ms)
495x344	1665,2	227,8
990x688	8309,6	826,2
1980x1376	49039,8	3282

TABLE II
EXPERIMENTATION SUMMARY



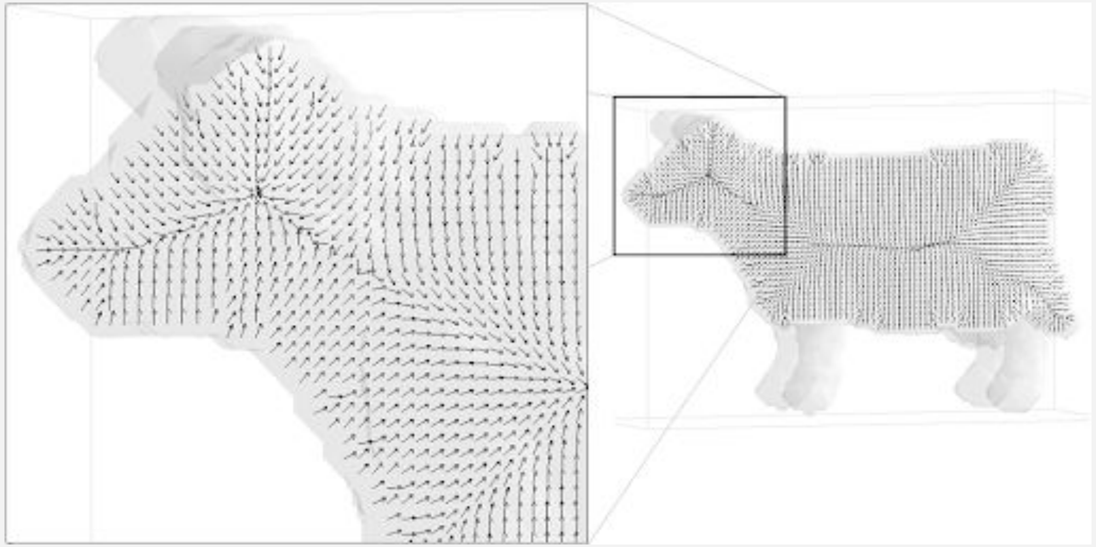
Skeletonization algorithms

Gradient Vector Field



$\mathbf{F} = P\mathbf{i} + Q\mathbf{j} + R\mathbf{k}$

P= (-x) / (x² + y² + z²)^ Q= (-y) / (x² + y² + z²)^ R= (-z) / (x² + y² + z²)^



$$(u_0, v_0) = \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

Source

First Approach: Skeletonization from the SSM Of The Distance Transform

[Thinning algorithms, discrete domain algorithms based on the Voronoi diagram, algorithms based on distance transform, and algorithms based on mathematical morphology.] → Ensure the accurate localization of skeleton points but neither connectivity nor completeness, that is, the branches extracted may be disconnected and may not be able to represent all the significant visual parts.

The skeleton is obtained by connecting the critical points with geodesic paths.

This approach overcomes intrinsic drawbacks of distance transform based skeletons, since it yields stable and connected skeletons without losing significant visual parts.

Source

SSM

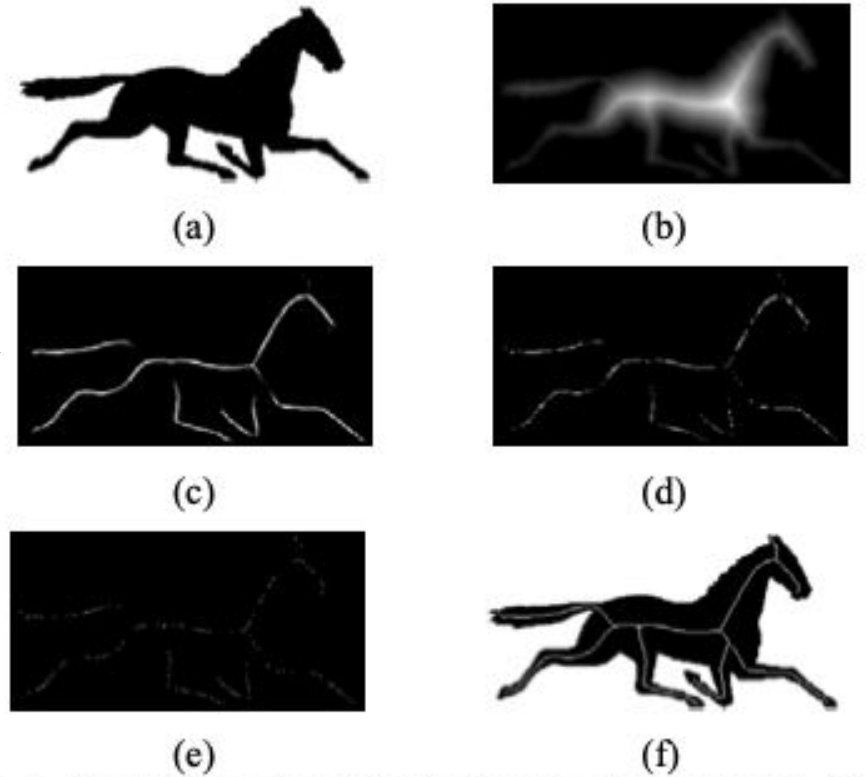


Fig. 1 Illustration of skeletonization by this approach. (a) is the original image, (b) the distance transform of (a), (c) is the SSM, (d) is the local maxima, (e) is the critical point set extracted from (d), (f) is the final skeleton.

Source

Steps of Skeletonization using the SSM

Input: Euclidean distance transform of an image

1) Calculate the SSM from the distance transform of the image.
2) Calculate local maxima to get connected components.
3) Critical points selection.
4) Critical points connection.

Output: Skeleton of the image

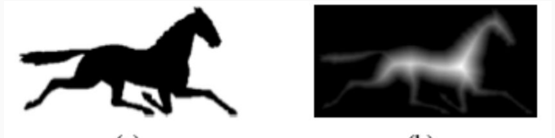
Source

Before applying the SSM algorithm

Calculate:

$$f(\vec{r}) = 1 - \| \nabla G_{\delta}(\vec{r}) * dt(\vec{r}) \|$$

Represents the inverted smoothed version of the distance transform


(a) (b)

Source

Finding the SSM at a given point

$$SSM(\vec{r}) = \max\left(0, \sum_{r \in N(\vec{r})} \frac{gvf(\vec{r}) \bullet (\vec{r} - \vec{r}\,')}{\|\vec{r} - \vec{r}\,'\|}\right)$$

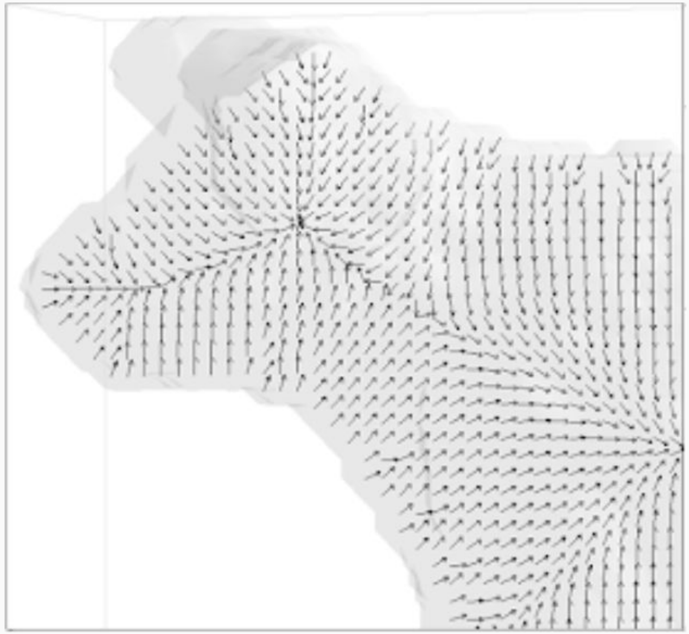
r is the current point

r’ is the neighbor

gvf(r) is the gradient vector field at the point r

(r - r’) is a vector subtraction

||r-r’|| is the euclidean distance between r and r’

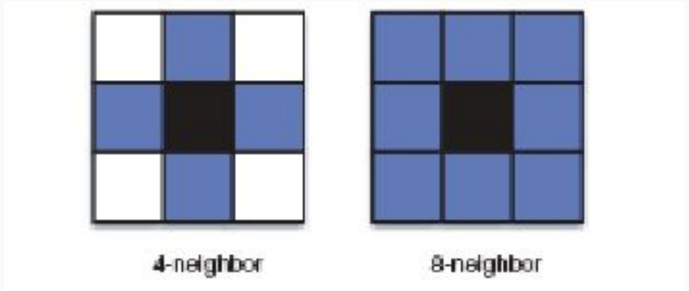


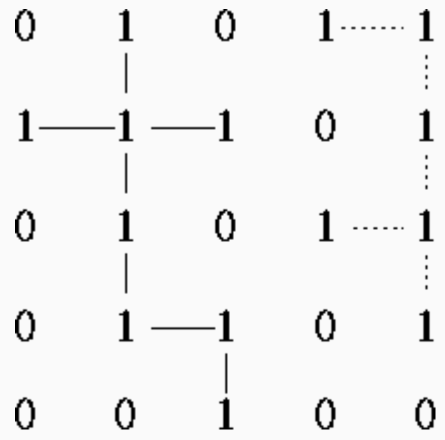
8-connected path

Path built from the 8 neighbors of a pixel.

1s represent the shape

0s represent the background





Example: 4-connected path

Gradient path

Gradient Length

$$|R|_G = \sum_{i=1}^{n} |f(\vec{r}_i)|$$

Gradient Distance

Between two points r and r'

Minimum gradient length of all paths conformed those points

Gradient Path

Smallest gradient distance

SSM Pseudocode

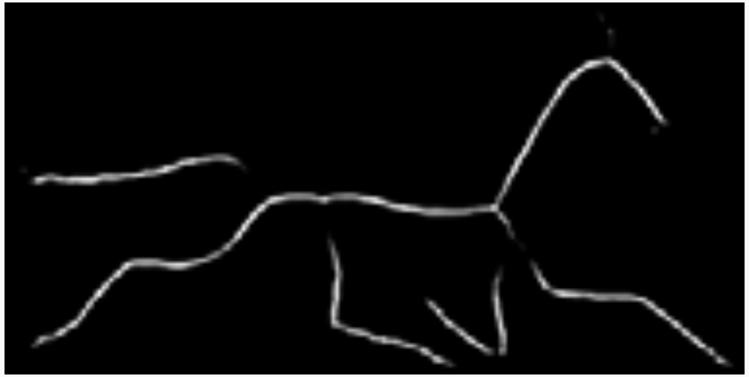
```
GradientVectorField:
    gradient = []
    for pixel in pixels:
        gradient[pixel] = CalculateGradient(pixel)
```

Takes linear time with respect to the number of pixels

SSM Pseudocode

```
FindSSM:
    SSM = {}
    for pixel in pixels:
        add to SSM max(0, sumOf8NeighborsGradients(
    pixel))
    returun SSM
```

$$SSM(\vec{r}) = \max\left(0, \sum_{r \in N(\vec{r})} \frac{gvf(\vec{r}) \bullet (\vec{r} - \vec{r}\,')}{\| \vec{r} - \vec{r}\,' \|}\right)$$



Takes linear time with respect to the number of pixels

SSM Pseudocode

```
ConnectedComponents:
  SSM = FindSSM()
  connectedComponents = {}
  for pixel in pixels:
    maxSSMofNeighbors = maxSSMofNeighbors(pixel)
    if SSM(pixel) >= maxSSMofNeighbors:
      add neighbor to region
    add pixel to connectedComponents
  return connectedComponents
```

$$SSM(\vec{r}) \geq \max_{\vec{r}' \in N(\vec{r})} \{SSM(\vec{r}')\}$$

Takes linear time with respect to the number of pixels

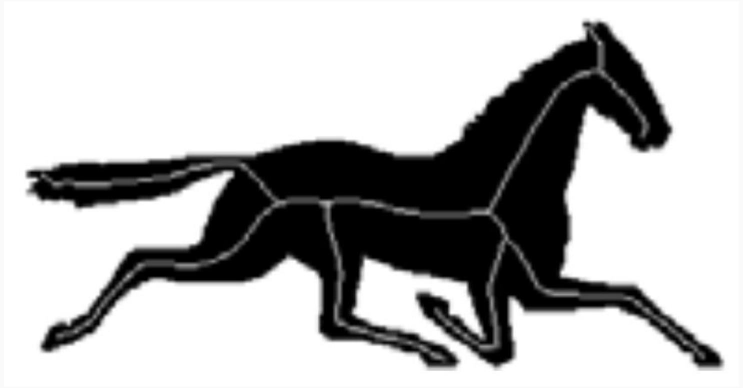
SSM Pseudocode

```
CriticalPointsSelection:
  connectedComponents = ConnectedComponents()
  criticalPoints = {}
  for connectedComponent in connectedComponents:
    criticalPoint = connectedComponent[0].gradient
    for i = 1 to connectedComponent.size()
      if connectedComponent[i].gradient <
    criticalPoint.gradient:
        criticalPoint = connectedComponent[i]
    add criticalPoint to criticalPoints
    add connectedComponent.endPoints to
    criticalPoints
  return criticalPoints
```

Takes O(m * r) time, where m is the number of connected components, and r is size of the biggest connected component

SSM Pseudocode

```
CriticalPointsConnection:
  criticalPoints = CriticalPointsSelection()
  gradientPaths = getGradientPaths()
  gradientPathsEndpoints = gradientPaths.endpoints
  maxPoint = getMaxDistanceTransform()
  skeleton = graph with criticalPoints and
    gradientPathsEndpoints as nodes
  for node in graph
    dijkstra(maxPoint, node)
  return skeleton
```



Takes O(p(p+e)lg(p)), where p is the number of critical points + gradient paths endpoints and e is the total number of edges

Skeletonization by SSM of the distance transform asymptotic analysis

Execution time takes $O(p(p+e)\lg(p))$

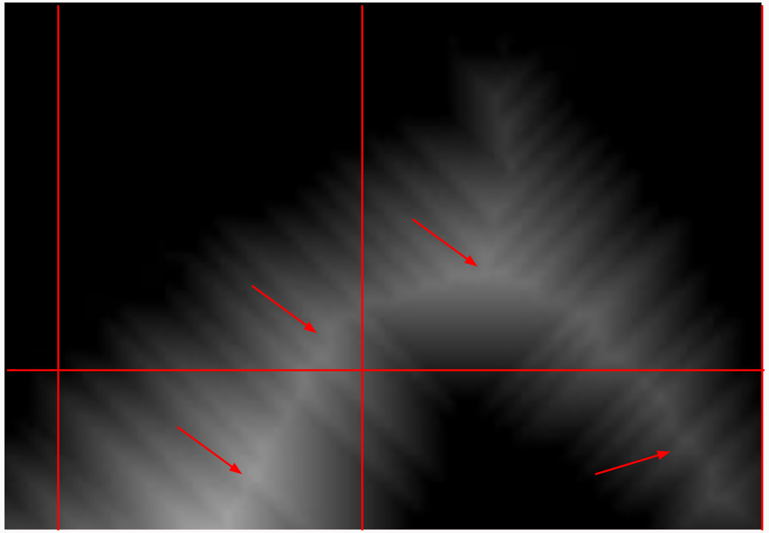
p: number of critical points + gradient paths endpoints

e: number of edges

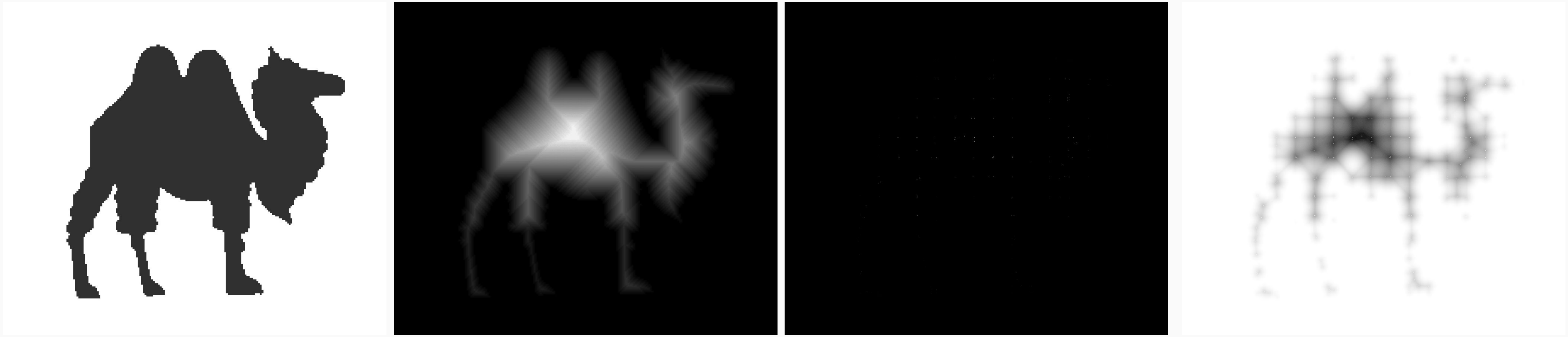
Skeletonization method proposal

To skeletonize the distance transforms we chose our own approach.

- First, we fragment the image into ’S’ segments. Then we implement a function to get the top ’N’ pixels with greater intensity. From this, we were able to find the most relevant points regarding each segment and thus the most relevant for the generation of the skeleton.



Result of Proposal method



Skeletonization method proposal asymptotic analysis

Iterate n pixels → n

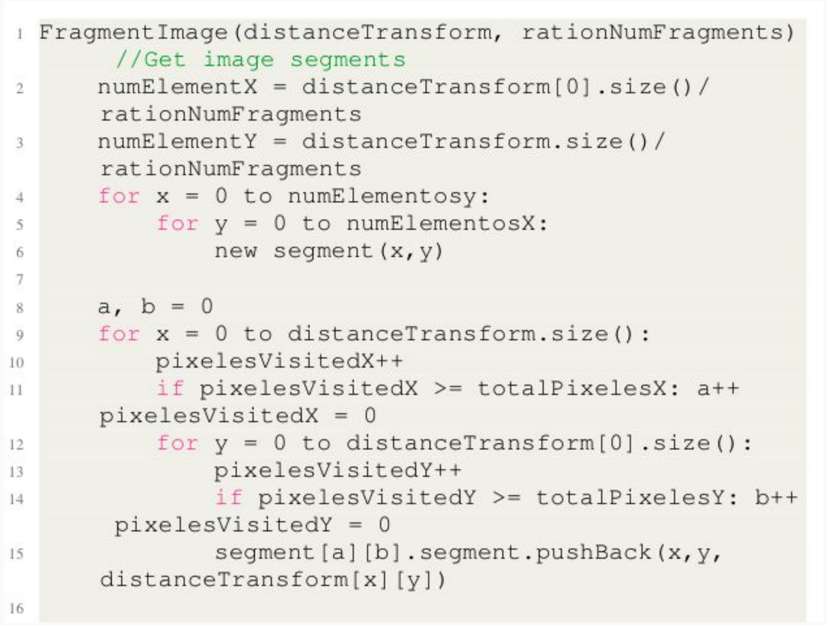
Merge sort → n lg(n)

The complexity of the proposed algorithm is $O(n^2 \log(n))$ where **n**: number of pixels of the image.

Because the number of segments is n/c, where c > 0, and we iterate through every segment.

Pseudocode

Method proposal

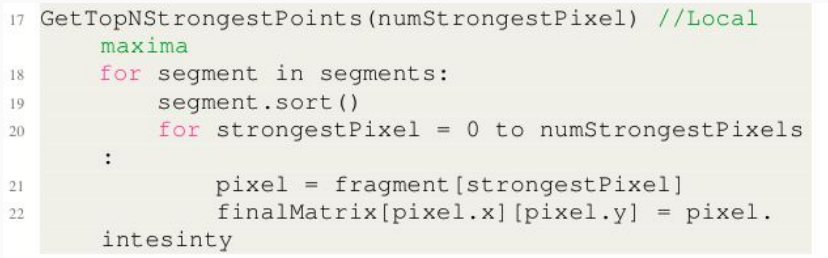


```
FragmentImage(distanceTransform, rationNumFragments) //Get image segments
    numElementX = distanceTransform[0].size()/rationNumFragments
    numElementY = distanceTransform.size()/rationNumFragments
    for x = 0 to numElementosy:
        for y = 0 to numElementosX:
            new segment(x,y)

    a, b = 0
    for x = 0 to distanceTransform.size():
        pixelesVisitedX++
        if pixelesVisitedX >= totalPixelesX: a++ pixelesVisitedX = 0
        for y = 0 to distanceTransform[0].size():
            pixelesVisitedY++
            if pixelesVisitedY >= totalPixelesY: b++ pixelesVisitedY = 0
            segment[a][b].segment.pushBack(x,y,distanceTransform[x][y])

```

numSegments = n / c



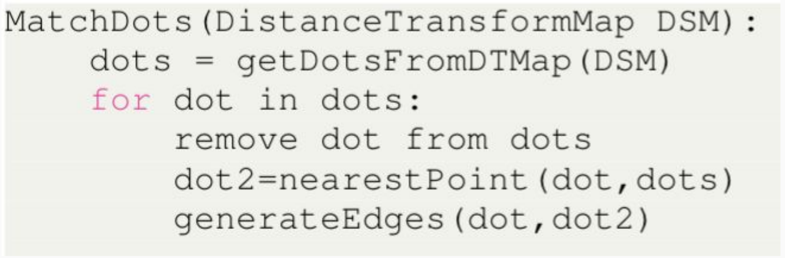
```
GetTopNStrongestPoints(numStrongestPixel) //Local maxima
    for segment in segments:
        segment.sort()
        for strongestPixel = 0 to numStrongestPixels:
            pixel = fragment[strongestPixel]
            finalMatrix[pixel.x][pixel.y] = pixel.intesinty
```

Listing 5. Skeletonization method proposal

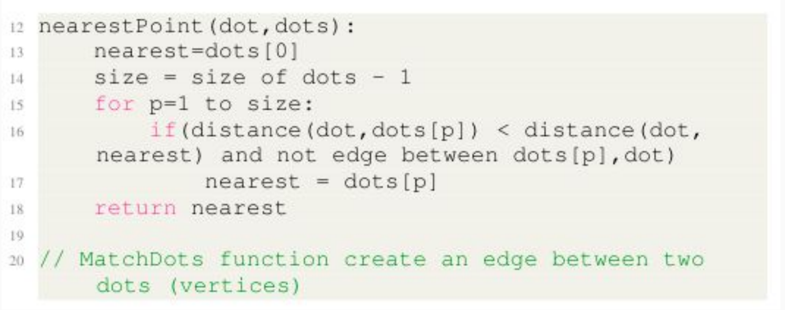
Joining dots - Building skeleton

- This step takes O(n²) time, where n is the number of dots received from the preview sub-step. This because an O(n) algorithm is applied n times where n is the size of the structure.

- Time complexity for this approach is dominated by the cost of track dots, for that reason the total cost is O(n²log(n))



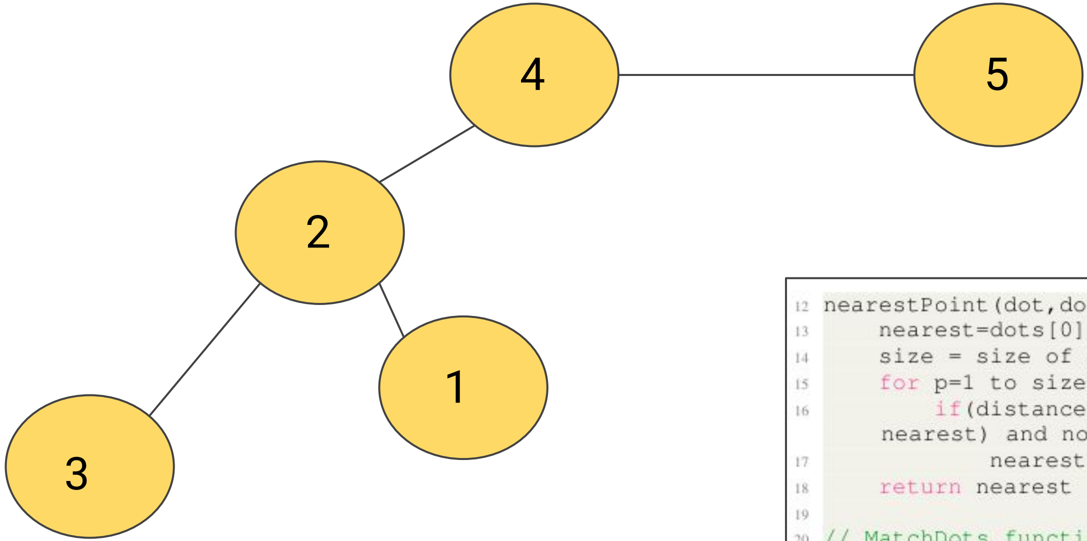
```
MatchDots(DistanceTransformMap DSM):
    dots = getDotsFromDTMap(DSM)
    for dot in dots:
        remove dot from dots
        dot2=nearestPoint(dot,dots)
        generateEdges(dot,dot2)
```

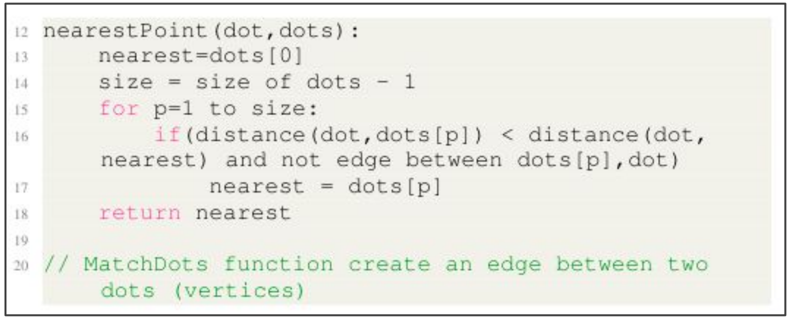


```
nearestPoint(dot,dots):
    nearest=dots[0]
    size = size of dots - 1
    for p=1 to size:
        if(distance(dot,dots[p]) < distance(dot,
    nearest) and not edge between dots[p],dot)
            nearest = dots[p]
    return nearest

// MatchDots function create an edge between two
    dots (vertices)
```

Demo

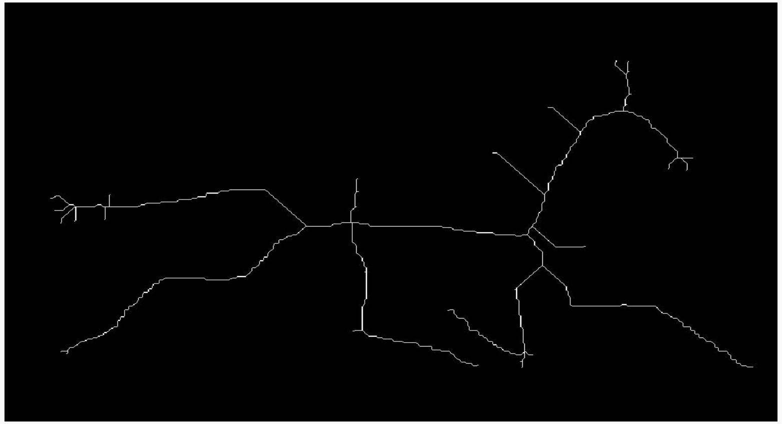


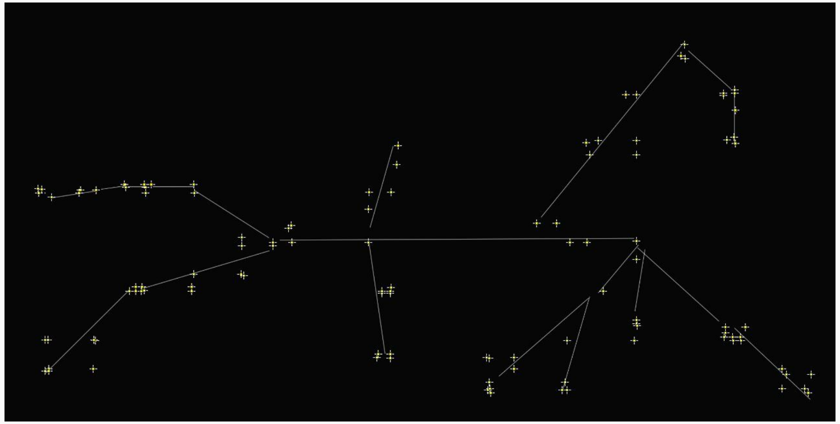


```
nearestPoint(dot,dots):
    nearest=dots[0]
    size = size of dots - 1
    for p=1 to size:
        if(distance(dot,dots[p]) < distance(dot,
    nearest) and not edge between dots[p],dot)
            nearest = dots[p]
    return nearest

// MatchDots function create an edge between two
    dots (vertices)
```

Expected vs Obtained

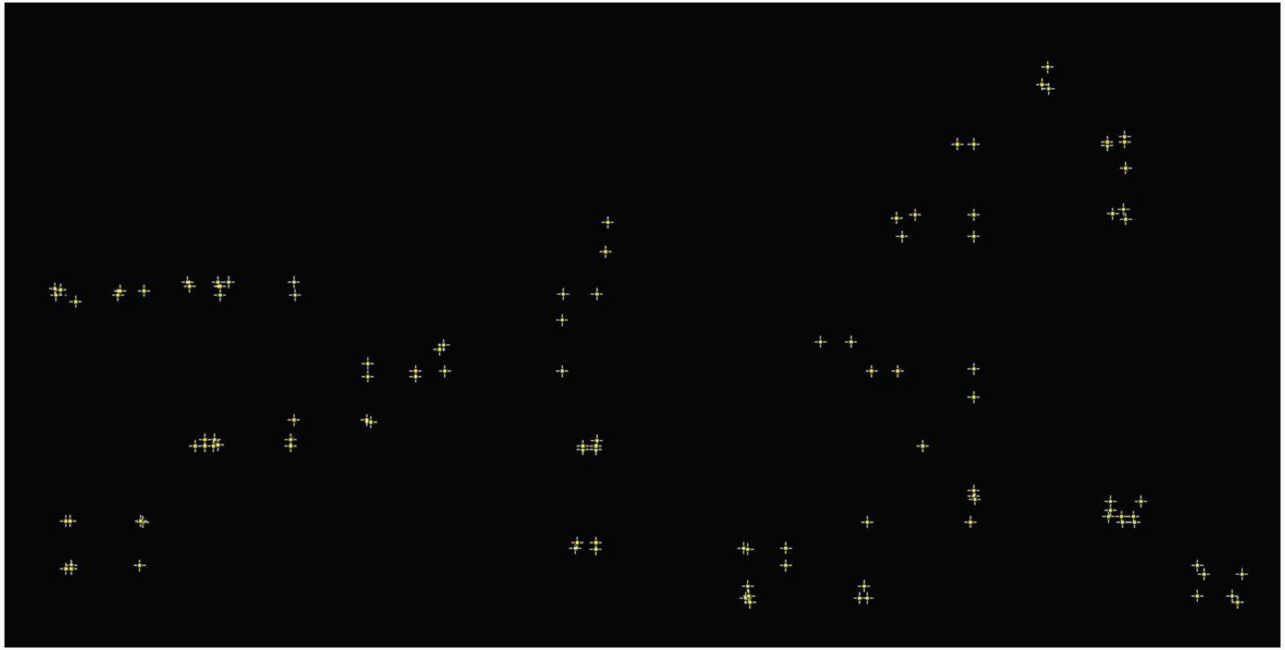




Limitations

- In the last step, software of other authors is used due failures and time limitations, but it sure to say that success can be reached because that software do exactly what we tried to implement.

Limitations



SSM vs. Skeletonization method proposal

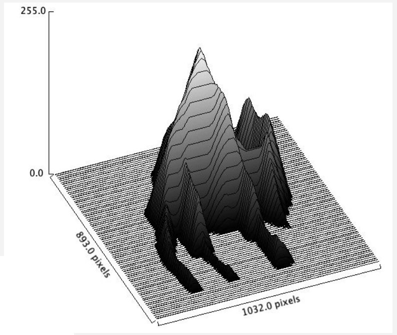
$$\boxed{\lim_{n \to \infty}\left(\frac{f_1(n)}{f_2(n)}\right)} \Rightarrow \boxed{\lim_{n\to\infty} \frac{n\left((n+e)\log_2(n)\right)}{n^2\log_2(n)}} \Rightarrow \boxed{\lim_{n\to\infty} \frac{n+e}{n}} \Rightarrow \boxed{\lim_{n\to\infty} \frac{n\left((n+e)\log_2(n)\right)}{n^2\log_2(n)} = 1}$$

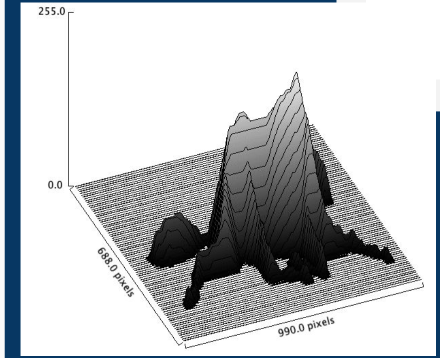
Since the limit of the expression is 1, a constant C, it is possible to say that T(n) of the first approach is θ(f(n)) where f(n) is T(n) for second approach, In other words, first approach is as asymptotically fast as the second approach.

Conclusion

- Within the different distance transform algorithms there are some that allow greater morphological precision such as SSM, which guarantees the connection of the critical points with geodesic paths. On the other hand, there exist other skeletonization algorithms that give a fairly precise result, but do not guarantee connectivity between all the sections of the skeleton.

- Both approaches used for skeletonization can not be used for 3D images without modifications, since the algorithms are not able to recognize patterns in variable dimensions. On the other hand, a new analysis would be necessary in order to calculate the new possible time complexity.



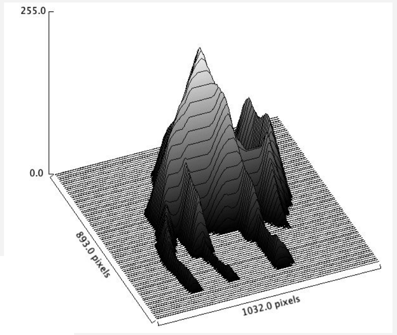


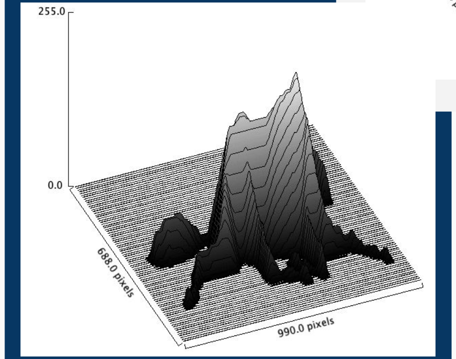
Conclusion

- In the empirical analysis, theoretical complexity times were successfully validated, being consistent with expectations. The first approach grows in a larger rate than the second approach.

- In theoretical analysis, the first approach takes 8 neighbors, and it is design to be generalized into various types of distances (Manhattan, Euler, Chebyshev) taking T(n) = O(nlg(n)) complexity. The second approach only uses 4 neighbors and it is optimized for one typeof distance (Chebyshev) taking T(n)= O(n) time complexity.

- T (n) of the SSM approach is Θ(f (n)) where f (n) is T (n) for proposal method.





Bibliography

[1] X. B. L. J. Latecki, Q. Li and W. Liu, “Skeletonization using ssm of the distance transform,”International Conference on Image Processing, San Antonio, TX, 2007, vol. 349, p. 352, 2007.

[2] B. G. . S. d. B. G. Saha, P. K., “Skeletonization and its applications – a review. skeletonization,” p. 3–42, 2017.

[3] K. Kaur and D. M. Kumar, “A method for binary image thinning using gradient and watershed algorithm,” International Journal of Advanced Reasarch in Computer Science and Software Engineering , vol. 3, 01 2013.

[4] P. Maragos and R. SCHAFER, “Morphological skeleton representation and coding binary images,” Acoustics, Speech and Signal Processing, IEEE Transactions on , vol. 34, pp. 1228 – 1244, 11 1986.

[5] F. A. Z. R. A. Lotufo, A. A. Falcao, “Fast euclidean distance transform using a graph-search algorithm,” SIBGRAPI ’00: Proceedings of the 13th Brazilian Symposium on Computer Graphics and Image Processing , October 2000.

[6] T. Wu, “Distance-transformation,” https://github.com/tyeonn/ Distance-Transformation, 2018.

[7] X. Y. N. Cornea, D. Silver and R. Balasubramanian, “Computing hierarchical curve-skeletons of 3d objects,” The Visual Computer, October 2005.

THANKS!