PA170 Digital Geometry

Lecture 4: Distance Measurement

Martin Maška (xmaska@fi.muni.cz)

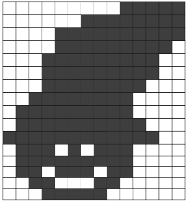
Centre for Biomedical Image Analysis
Faculty of Informatics, Masaryk University, Brno

Autumn 2023

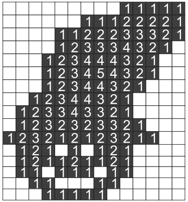
INTRODUCTION TO DISTANCE TRANSFORMS

Distance Transforms

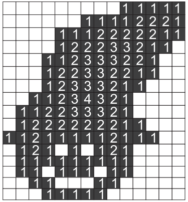
- Let I be a binary image, defined on a grid $\mathbb{G}$, with nonempty foreground $\langle I \rangle$ as well as background $\overline{\langle I \rangle}$. For any grid metric d, the *d* distance transform of I associates with every image element $p \in \mathbb{G}$ the (shortest) d distance from p to $\overline{\langle I \rangle}$
- Remark: The distance from $p \in S$ to $T \subseteq S$ is defined as $d(p, T) = \min_{q \in T} d(p, q)$



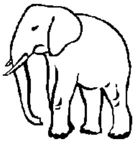
Binary image



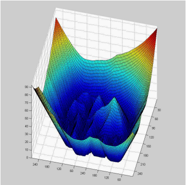
d_4 distance transform



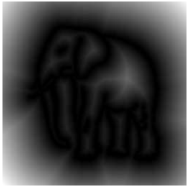
d_8 distance transform



Binary image



Distance graph



Distance map

Iso-distance curves

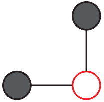
Distance Transforms: Commentary

- If one is interested in distances between background image elements and the foreground, the role of background and foreground is exchanged
- Foreground and background distance transforms are sometimes represented by the signed distance function
- Distance transforms are exploited in a broad range of applications:
  - Separation of touching objects
  - Computation of morphological operators (dilation, erosion)
  - Computation of geometrical representations (skeletonization, Voronoi tesselation, Delaunay triangulation, medial axes, etc.)
  - Robot navigation
  - Distance-based shape measurements (object centers, maximal width, etc.)
  - Pattern (shape) matching
  - Image registration
  - k-NN computation
  - . . .

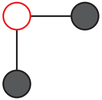
DISTANCE TRANSFORM FOR REGULAR METRICS

Regular Distance Transform: Preliminaries

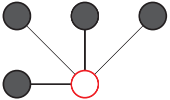
- Let $p \in \mathbb{G}$ be a grid point of a grid $\mathbb{G}$. Its α-adjacency set $A_\alpha(p)$ can be split into two disjoint sets ($A_\alpha(p) = A_\alpha^{\leftarrow}(p) \cup A_\alpha^{\rightarrow}(p)$), depending on whether an adjacent grid point $q \in \mathbb{G}$ precedes ($q \in A_\alpha^{\leftarrow}(p)$) or follows ($q \in A_\alpha^{\rightarrow}(p)$) the grid point p when scanning $\mathbb{G}$ row by row from top to bottom and each row is scanned from left to right



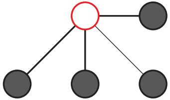
$A_4^{\leftarrow}(p)$



$A_4^{\rightarrow}(p)$



$A_8^{\leftarrow}(p)$



$A_8^{\rightarrow}(p)$

Regular Distance Transform: Two-Pass Algorithm

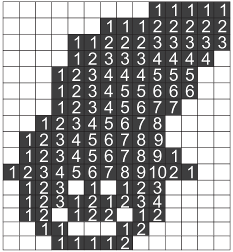
1. In a single left-to-right, top-to-bottom scan of $\mathbb{G}$ calculate for each image element p

$$f_1(p) = \begin{cases} 0, & \text{if } p \in \overline{\langle I \rangle} \\ \min\{f_1(q) + w(p,q) : q \in A_\alpha^{\leftarrow}(p)\}, & \text{if } p \in \langle I \rangle \end{cases}$$

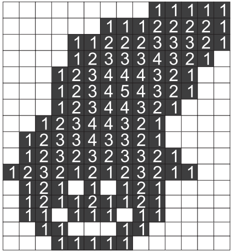
2. In a single right-to-left, bottom-to-top scan of $\mathbb{G}$ calculate for each image element p

$$f_2(p) = \min\{f_1(p), f_2(q) + w(p,q) : q \in A_\alpha^{\rightarrow}(p)\} = d(p, \overline{\langle I \rangle})$$

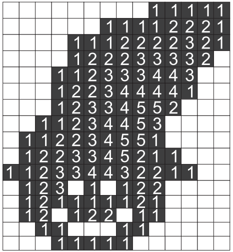
where $w(p,q) = 1$ in case of d_4 and d_8; and $w(p,q) = a$ for 4-adjacent image elements and $w(p,q) = b$ for 8-adjacent image elements in case of $d_{a,b}$ $(0 < a \leq b \leq 2a)$



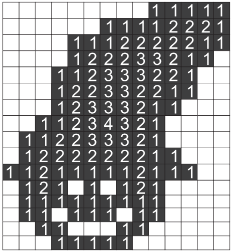
First pass (d_4)



Second pass (d_4)



First pass (d_8)



Second pass (d_8)

Two-Pass Algorithm: Properties

- The calculated distances are exact for all regular metrics on digital grids
- Easy implementation without the need for an auxiliary image buffer (i.e., only the input binary image and the ouput image with the calculated distances are needed)
- Straightforward extension into higher dimensions, especially for higher-dimensional chamfer distances
- Its time complexity is $\mathcal{O}(n)$ (n is the number of image elements)

EUCLIDEAN DISTANCE TRANSFORM (EDT)

Classification of Algorithms for EDT

Remark: The Euclidean metric d_e is not regular on digital grids, and thus the incremental propagation of distances is not so straightforward

Brute-force approaches

- Exact but inefficient calculation of Euclidean distances

Ordered-propagation approaches

- Efficient but non-exact calculation of Euclidean distances
- The Fast Marching algorithm is relatively difficult to implement (see PA166)

Raster-scanning approaches

- Efficient but non-exact calculation of Euclidean distances
- Danielsson's algorithm is easy to implement

Danielsson's Algorithm: Main Idea

- It is a two-pass, raster-scanning algorithm for calculating non-exact Euclidean distance transforms
- It propagates pairs of integers (not distances themselves), which encode the absolute values of the relative coordinates of the nearest background image element
- The pairs of integers are propagated from top to bottom and from bottom to top, being compared as follows: $\mathtt{min}\{(x_1, y_1), (x_2, y_2)\}$ is equal to (x_i, y_i) for which $x_i^2 + y_i^2$ is smaller. If they are equal the pair with the smaller x-coordinate is taken.

Danielsson's Algorithm: Pseudocode

1. Set integer pairs in `tmp` to $(0, 0)$ (or (∞, ∞)) for background (or foreground) pixels
2. For each row of `tmp` (from top to bottom), replace each $(f(x), f(y))$
   1. from left to right with $\min\{(f(x), f(y)), (f(x), f(y-1)) + (0, 1)\}$
   2. from left to right with $\min\{(f(x), f(y)), (f(x-1), f(y)) + (1, 0)\}$
   3. from right to left with $\min\{(f(x), f(y)), (f(x+1), f(y)) + (1, 0)\}$
3. For each row of `tmp` (from bottom to top), replace each $(f(x), f(y))$
   1. from right to left with $\min\{(f(x), f(y)), (f(x), f(y+1)) + (0, 1)\}$
   2. from right to left with $\min\{(f(x), f(y)), (f(x+1), f(y)) + (1, 0)\}$
   3. from left to right with $\min\{(f(x), f(y)), (f(x-1), f(y)) + (1, 0)\}$
4. Calculate the Euclidean distances from the integer pairs stored in `tmp`

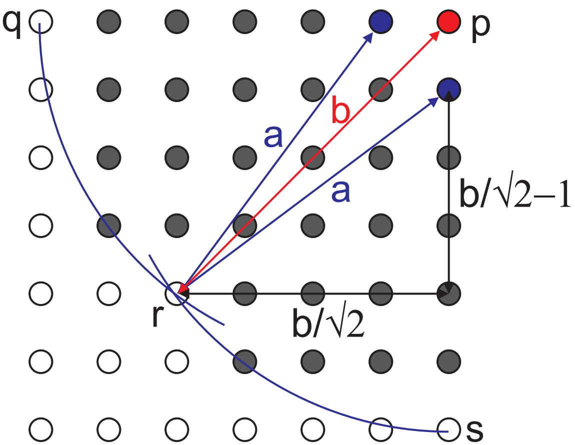
									0,1	0,1	0,1	0,1	0,1
						0,1	0,1	0,1	1,1	0,2	0,2	0,2	1,0
				0,1	0,1	1,1	0,2	0,2	1,2	2,2	1,2	1,1	0,1
				1,0	0,2	1,2	2,2	0,3	1,3	2,1	1,1	0,1	
			0,1	1,1	2,1	1,3	2,3	3,2	2,2	2,0	1,0		
			1,0	2,0	2,2	3,2	3,2	2,2	1,2	1,1	0,1		
			1,0	2,0	3,0	4,0	3,1	2,1	1,1	0,1			
		0,1	1,1	2,1	3,1	0,4	3,0	2,0	1,0				
	0,1	1,1	1,2	2,2	1,3	0,3	3,0	2,0	1,0				
	1,0	2,0	1,2	0,2	1,2	0,2	1,2	2,1	1,1	0,1			
0,1	1,1	2,1	1,1	0,1	1,1	0,1	1,1	2,1	1,1	0,1	0,1		
	1,0	2,0	1,0		1,0		1,0	2,0	1,0				
	1,0	1,1	0,1	0,1	1,1	0,1	0,1	1,1	1,0				
	0,1	1,0		0,1	0,1	0,1		1,0	0,1				
		0,1	0,1				0,1	0,1					
			0,1	0,1	0,1	0,1	0,1						

Final integer pairs (`tmp`)

									1	1	1	1	1
						1	1	1	1.4	2	2	2	1
				1	1	1.4	2	2	2.2	2.8	2.2	1.4	1
				1	2	2.2	2.8	3	3.2	2.2	1.4	1	
			1	1.4	2.2	3.2	3.6	3.6	2.8	2	1		
			1	2	2.8	3.6	3.6	2.8	2.2	1.4	1		
			1	2	3	4	3.6	2.2	1.4	1			
		1	1.4	2.2	3.2	4	3	2	1				
	1	1.4	2.2	2.8	3.2	3	3	2	1				
	1	2	2.2	2	2.2	2	2.2	2.2	1.4	1			
1	1.4	2.2	1.4	1	1.4	1	1.4	2.2	1.4	1	1		
	1	2	1		1		1	2	1				
	1	1.4	1	1	1.4	1	1	1.4	1				
	1	1		1	1	1		1	1				
		1	1				1	1					
			1	1	1	1	1						

Euclidean distances

Danielsson's Algorithm: Erroneous Distances



$$b = \frac{1}{\sqrt{2}} + \sqrt{a^2 - \frac{1}{2}} \approx a + \frac{1}{\sqrt{2}} < a + 1$$

Danielsson's Algorithm: Discussion

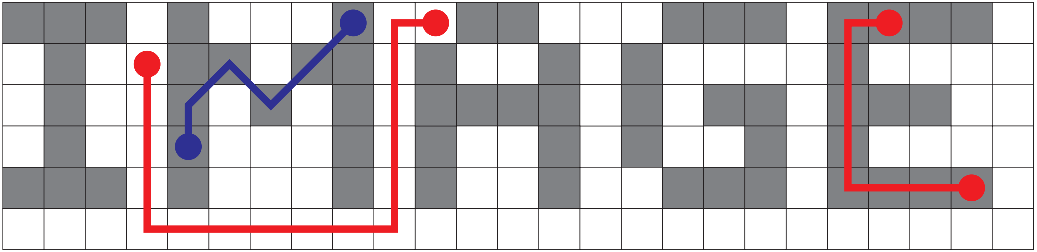
- The calculated distances are not always exact, differing from the Euclidean distances by a fraction of the grid constant at most
- Easy implementation with the need of an auxiliary image buffer
- Possible extension into higher dimensions
- Its time complexity is $\mathcal{O}(n)$ (n is the number of image elements)
- Different masks and propagation strategies can be taken to improve the accuracy of the calculated distances

GEODESIC DISTANCE

- A geodesic distance between two points $p \in S$ and $q \in S$ is defined as the length of the shortest path $\rho = (p = p_1, p_2, \ldots, p_n = q), p_i \in S, 0 \leq i \leq n$, from p and q within a set S:

$$d^S(p, q) = \mathcal{L}(\rho) = \sum_{i=1}^{n-1} d_N(p_i, p_{i+1})$$

where $d_N(\cdot, \cdot)$ is the distance between two adjacent points along the path ρ



The shortest 8-path (of length 5) and 4-paths (of length 16 and 8) within different sets of pixels

Geodesic Distance: Algorithm

- Raster-scanning approaches are not suitable because they must run until stability

- Ordered-propagation approaches are thus preferred:
  - Their main idea is to process image elements from closest to farthest
  - A positive distance between adjacent image elements (i.e., $d_N > 0$) guarantees that every image element is propagated only once

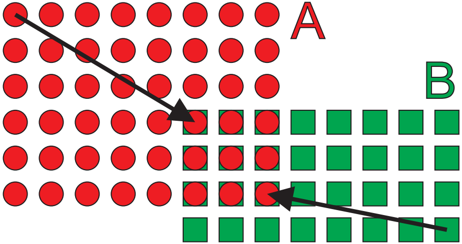
  - Dijkstra algorithm (a graph-based approach): $\mathcal{O}(m \cdot \log n)$ (m is the number of edges, and n is the number of vertices)
  - Fast Marching algorithm (a PDE-based approach): $\mathcal{O}(n \cdot \log n)$ (n is the number of image elements)

DISTANCES BETWEEN SETS

Hausdorff Metric (Hausdorff Distance)

- Any metric d on a compact set S can be extended to a Hausdorff metric on the family of all nonempty compact subsets A, B of S by defining

$$\mathrm{HD}(A,B) = \max\left\{\max_{p\in A}\min_{q\in B} d(p,q), \max_{p\in B}\min_{q\in A} d(p,q)\right\}$$



- The Hausdorff distance is very sensitive to outliers
- Percentile Hausdorff distance (the inner maxima replaced by a percentile – often the 95th percentile), average distance (the inner maxima replaced by averaging), and symmetric-difference-based metrics ($\mathrm{card}(A\Delta B)$ and $\frac{\mathrm{card}(A\Delta B)}{1+\mathrm{card}(A\cup B)}$) are used in practice

Hausdorff Distance: Algorithm

- Let $A, B \subset \mathbb{G}_{m,n}$ be two finite sets of grid points
- Let R be the smallest isothetic rectangle of size $k \times l$, which contains $A \cup B$

- If $\mathrm{card}(A)$ and $\mathrm{card}(B)$ are $\mathcal{O}(k \cdot l)$, a brute-force algorithm takes $\mathcal{O}(k^2 \cdot l^2)$ steps

- By calculating and scanning distance transforms of $\overline{A}$ and $\overline{B}$, the Hausdorff distance $\mathrm{HD}(A, B)$ can efficiently be calculated in $\mathcal{O}(k \cdot l)$ steps:
  1. Calculate a distance transform $\mathrm{DT}(\overline{A})$ in R
  2. Calculate a distance transform $\mathrm{DT}(\overline{B})$ in R
  3. Let a be the maximum value in $\mathrm{DT}(\overline{A})$ across all the grid points in B
  4. Let b be the maximum value in $\mathrm{DT}(\overline{B})$ across all the grid points in A
  5. Return $\max\{a, b\}$

Take-Home Messages

- Distance transforms for regular metrics can be calculated exactly on digital grids using a two-pass algorithm
- Non-exact Euclidean distance transforms can efficiently be computed using the Danielsson algorithm
- Geodesic (intrinsic) distances can efficiently be computed using ordered-propagation approaches
- Modified Hausdorff distances are often used in practice when calculating distances between sets