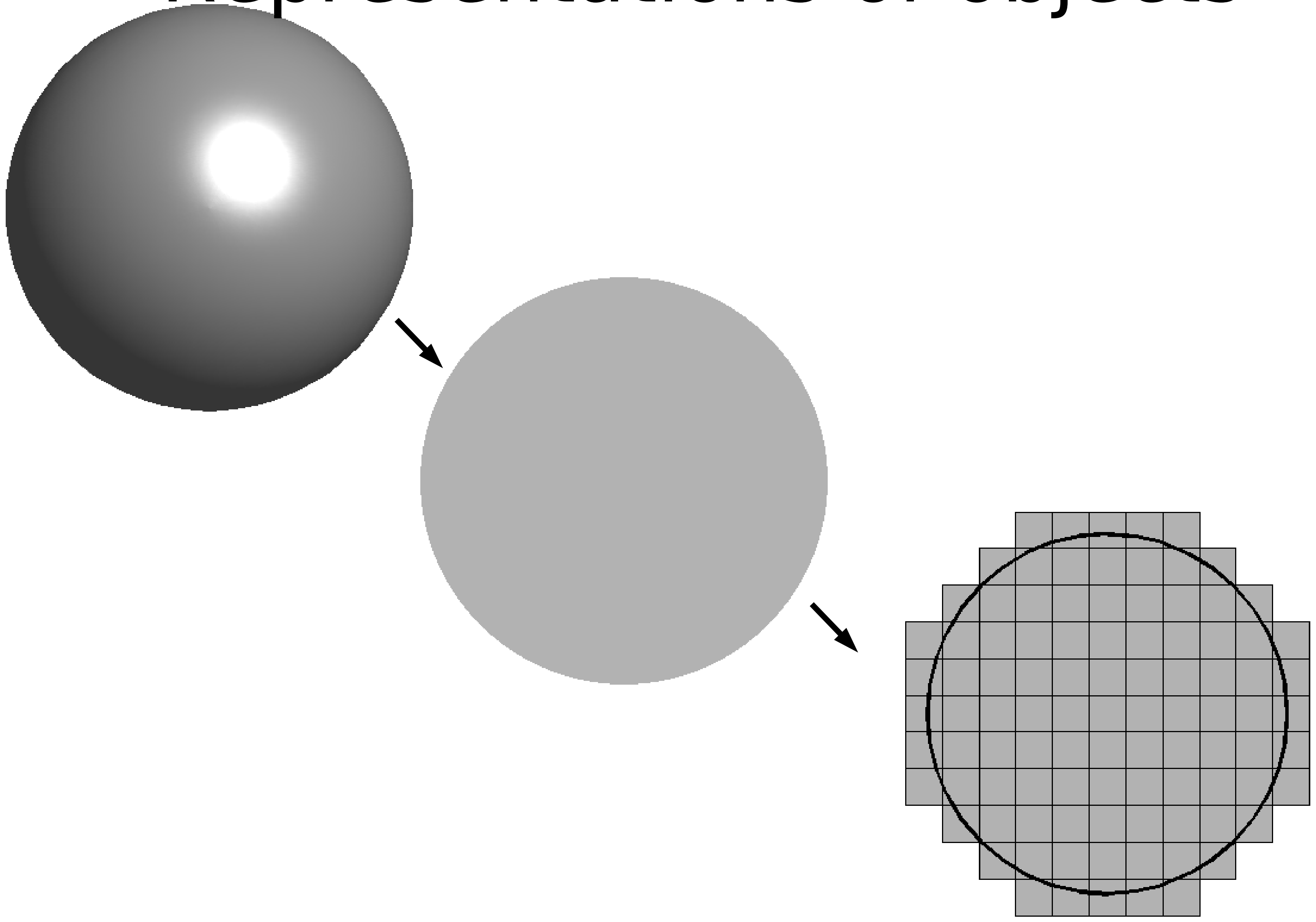
Digital geometry

2D Digital Geometry

Representations of objects



Digital Geometry

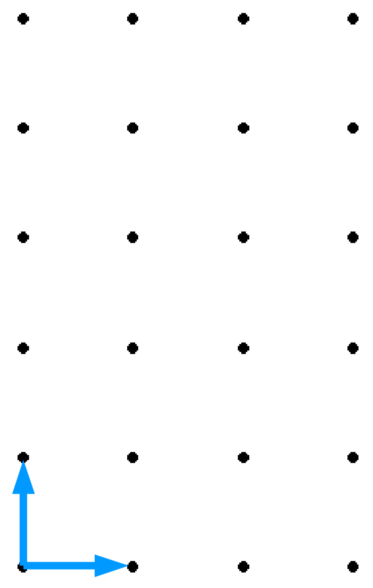
“The geometry of the computer screen”

The elements are points with integer coordinates.

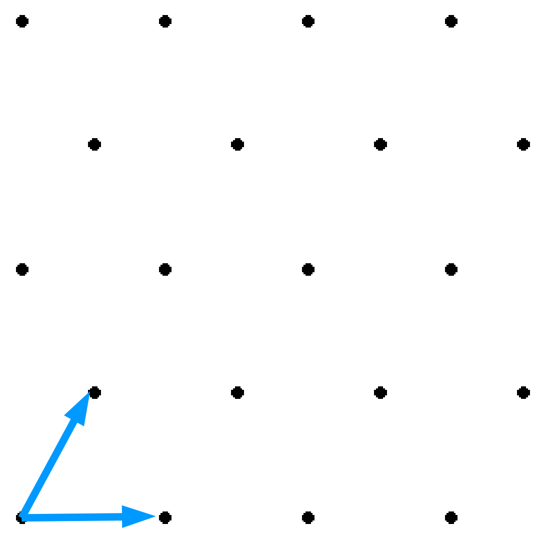
Which primitives do we use?

Grids (2D)

Square grid



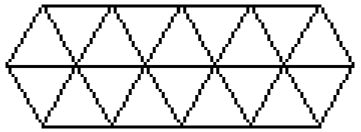
Hexagonal grid



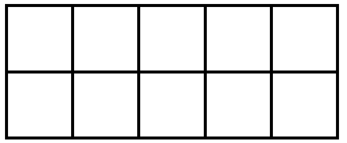
Tessellation

A *tessellation* of the plane is a collection of plane figures that fills the plane with no overlaps and no gaps.

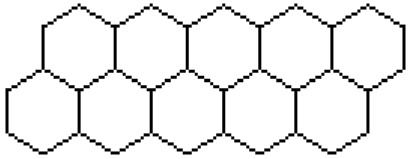
a tessellation of triangles



a tessellation of squares

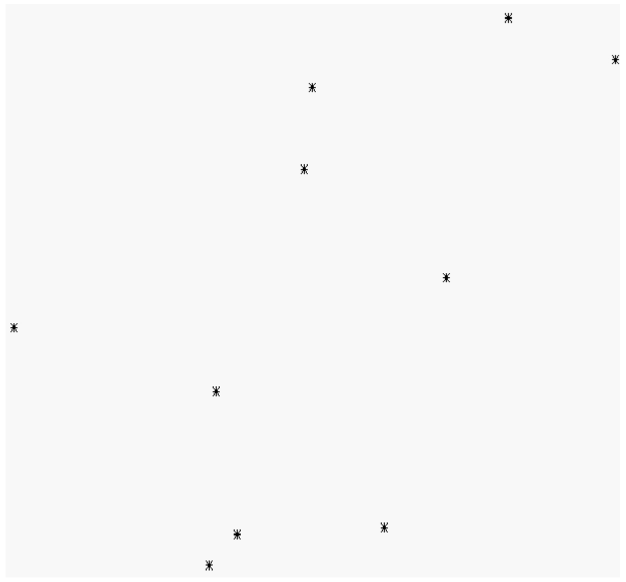


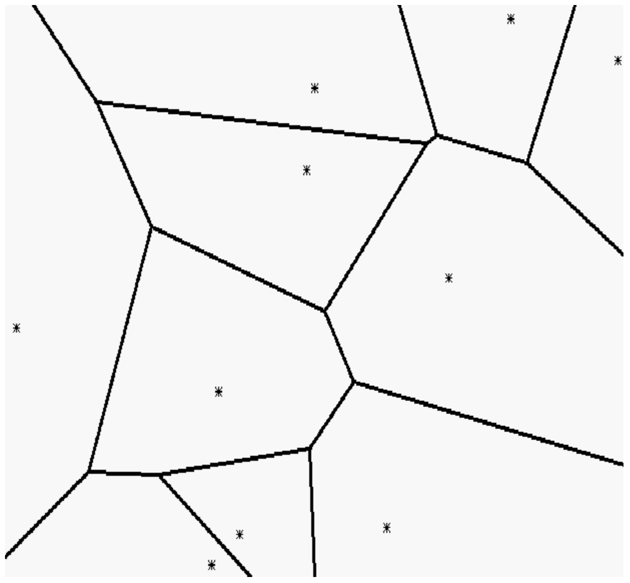
a tessellation of hexagons



Voronoi diagram (giving dirichlet tessellation)

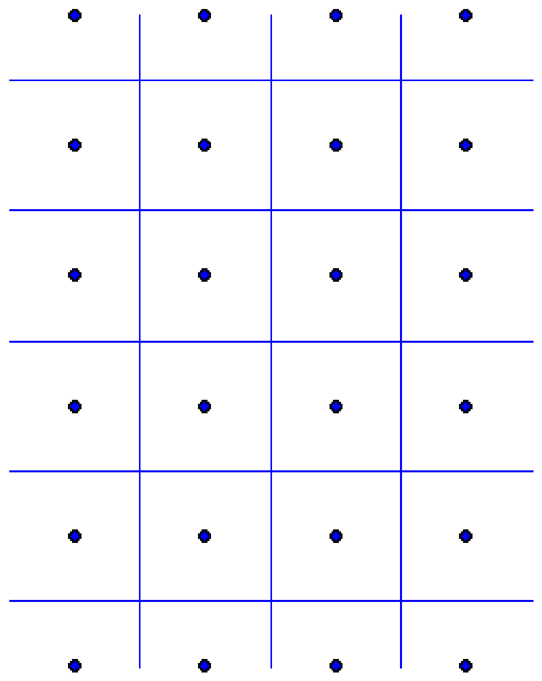
of a discrete set of points (generating points): partition the plane into cells so that each cell contains exactly one generating point and the locus of all points which are nearer to this generating point than to other generating points.



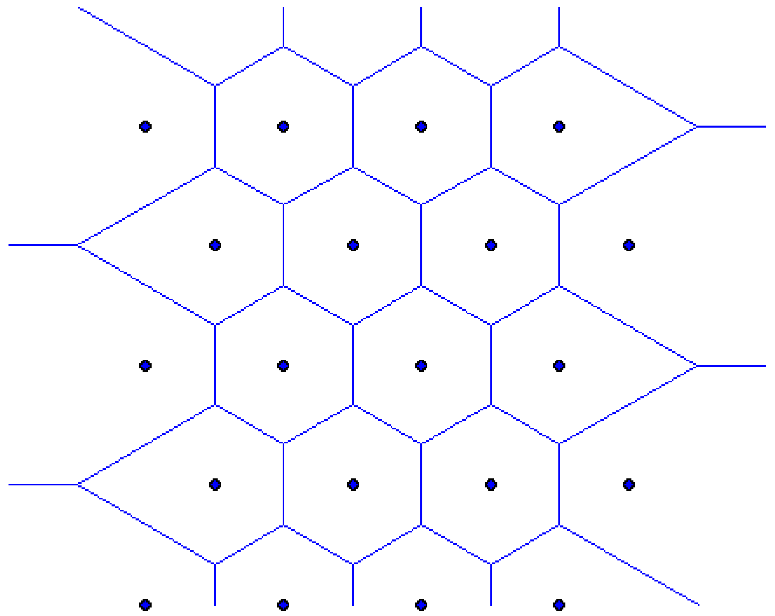


Picture elements (pixels) Voronoi regions

Square grid



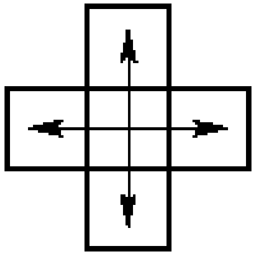
Hexagonal grid



Connectivities (2D)

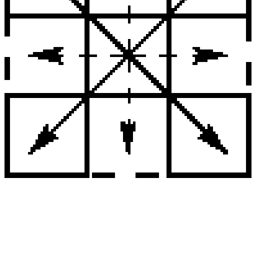
4-connectedness

objects connected through edge-neighbors



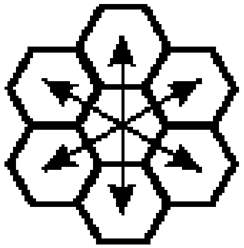
8-connectedness

objects connected through edge- and vertex-neighbors



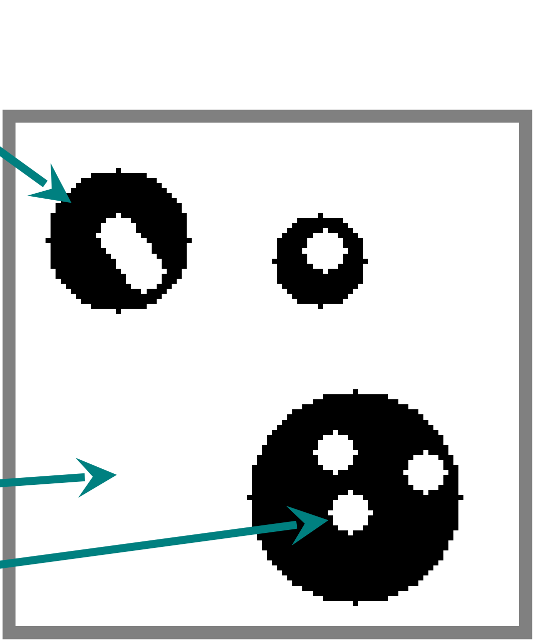
6-connectedness

for hexagonal grid



Connectivity – object and background

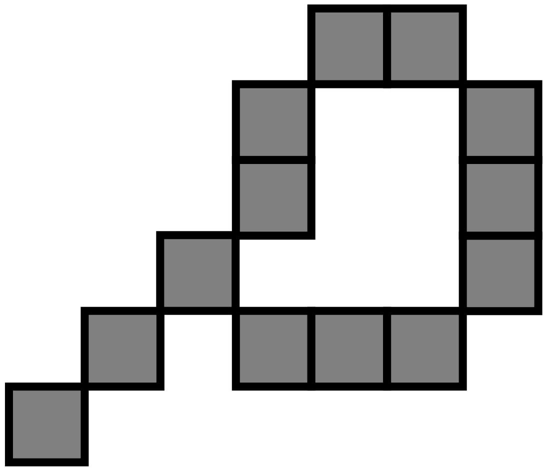
- connected object components (O_i)
- the object O is the union of all O_i
- the complement of O (O^c) consists of
  - background
    - connected to border of image / image limits
  - holes



Connectivity paradox

Euclidean geometry:
A closed (simple) curve divides the plane into two (distinct) connected components.

Digital geometry:

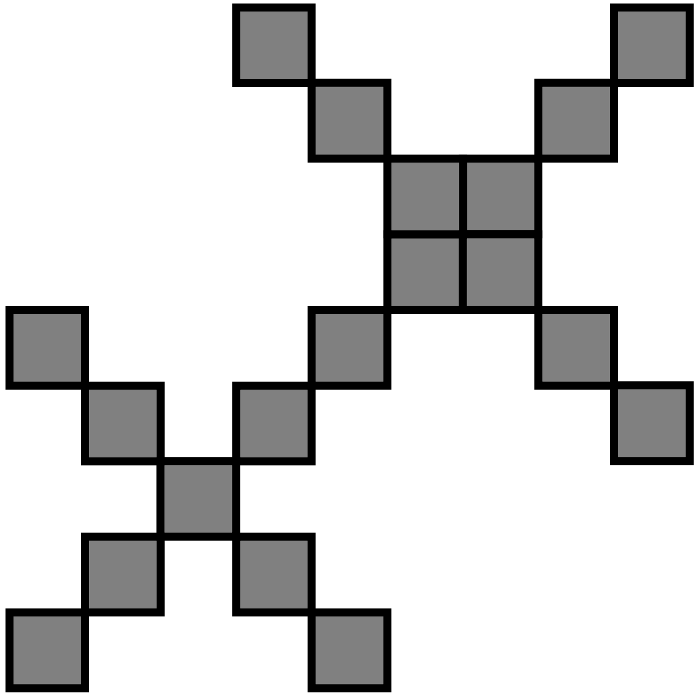


How many background components?!?

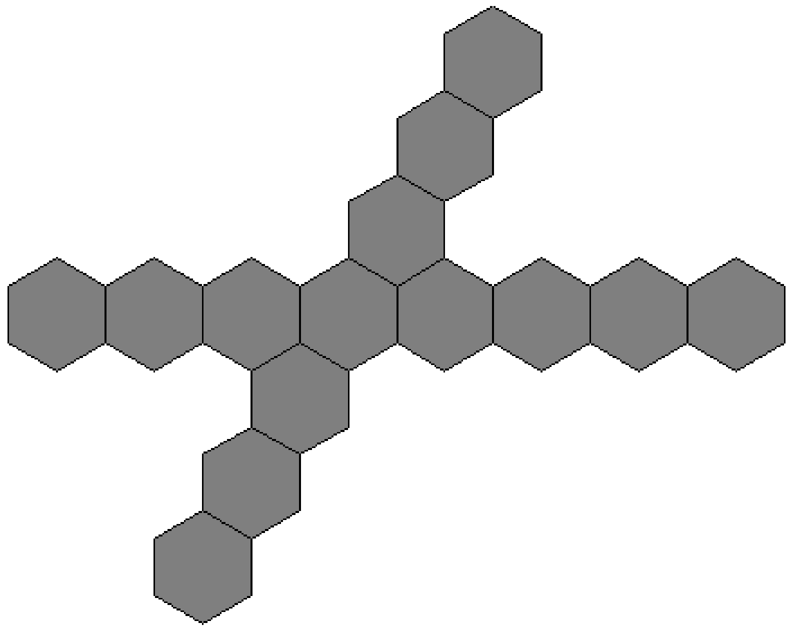
Connectivity paradox: connected or intersecting lines?

Solution 1

Use 4-connectedness for background and 8-connectedness for object (or vice versa)



Connectivity paradox: connected or intersecting lines?

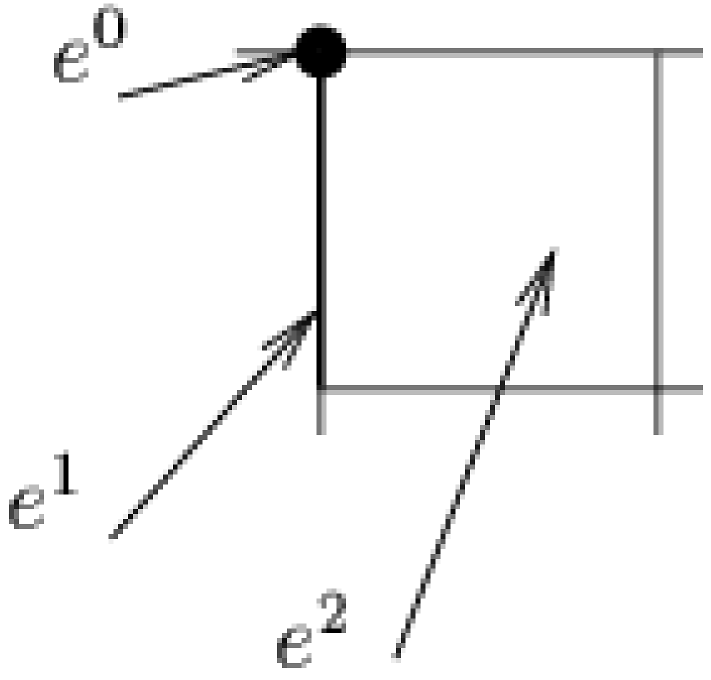


Solution 2
Use hexagonal grid

Connectivity paradox: connected or intersecting lines?

Solution 3
Use cellular complexes:

Elements of dimension
0 (point), 1 (line),
and 2 (area) are used.



Digital Geometry

“The geometry of the computer screen”
The elements are points with integer coordinates.

Different from the Euclidean geometry.

Example: What is a straight line?

Euclidean Geometry

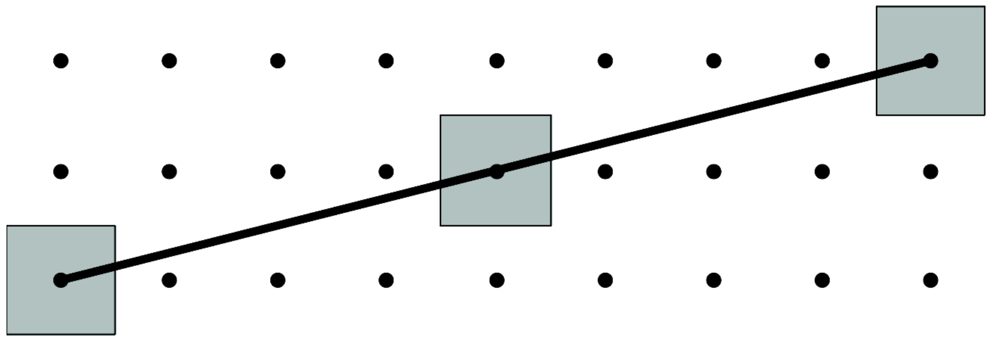
What is a straight line?

Intuitively:

- A curve traced by a point traveling in a constant direction
- A curve of zero curvature
- The distance between two points is the length of the straight line segment between the points

Digital Geometry

What is a straight line?



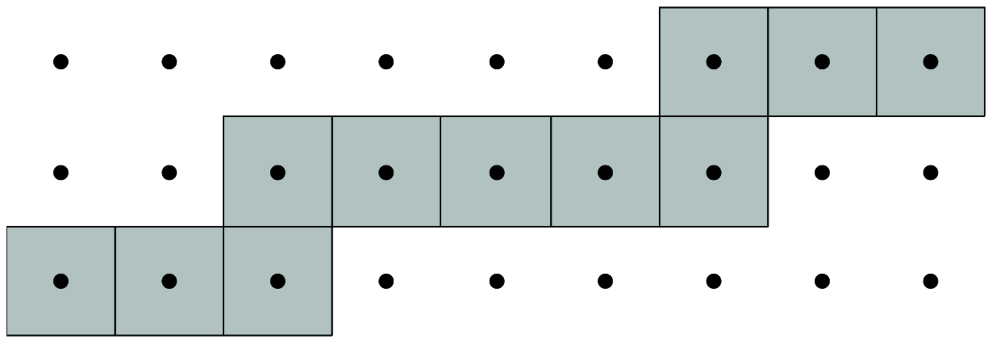
A set of pixels is a *(simple) arc* if it is connected, and all bu
two of its points (the “endpoints") have exactly two
neighbors in the set, while those two have exactly one.

Digital Geometry

In Euclidean geometry:

“The distance between two points is the length of the straight line segment between the points”

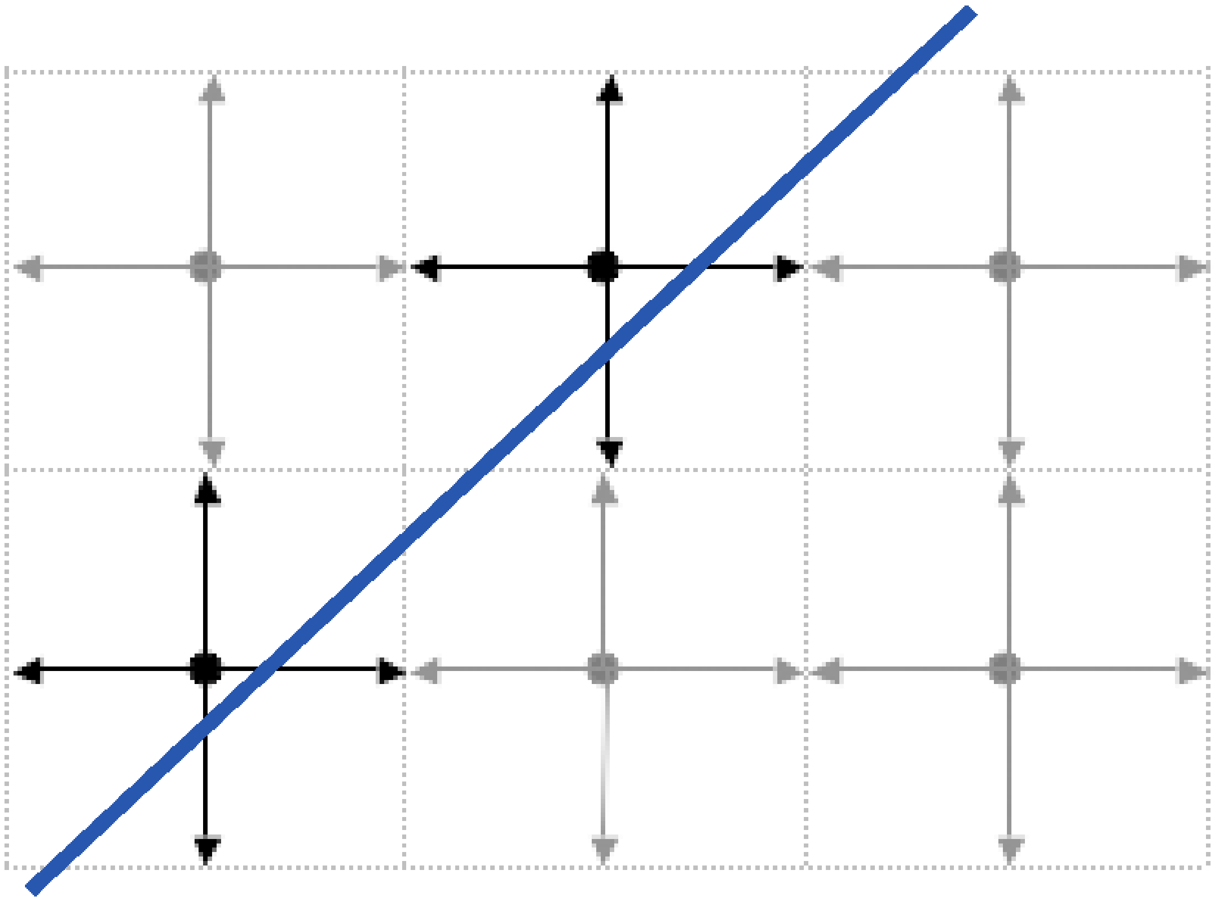
What about digital geometry?



“City-block distance”

Digital Geometry

Digitization of Euclidean straight line by grid intersection.

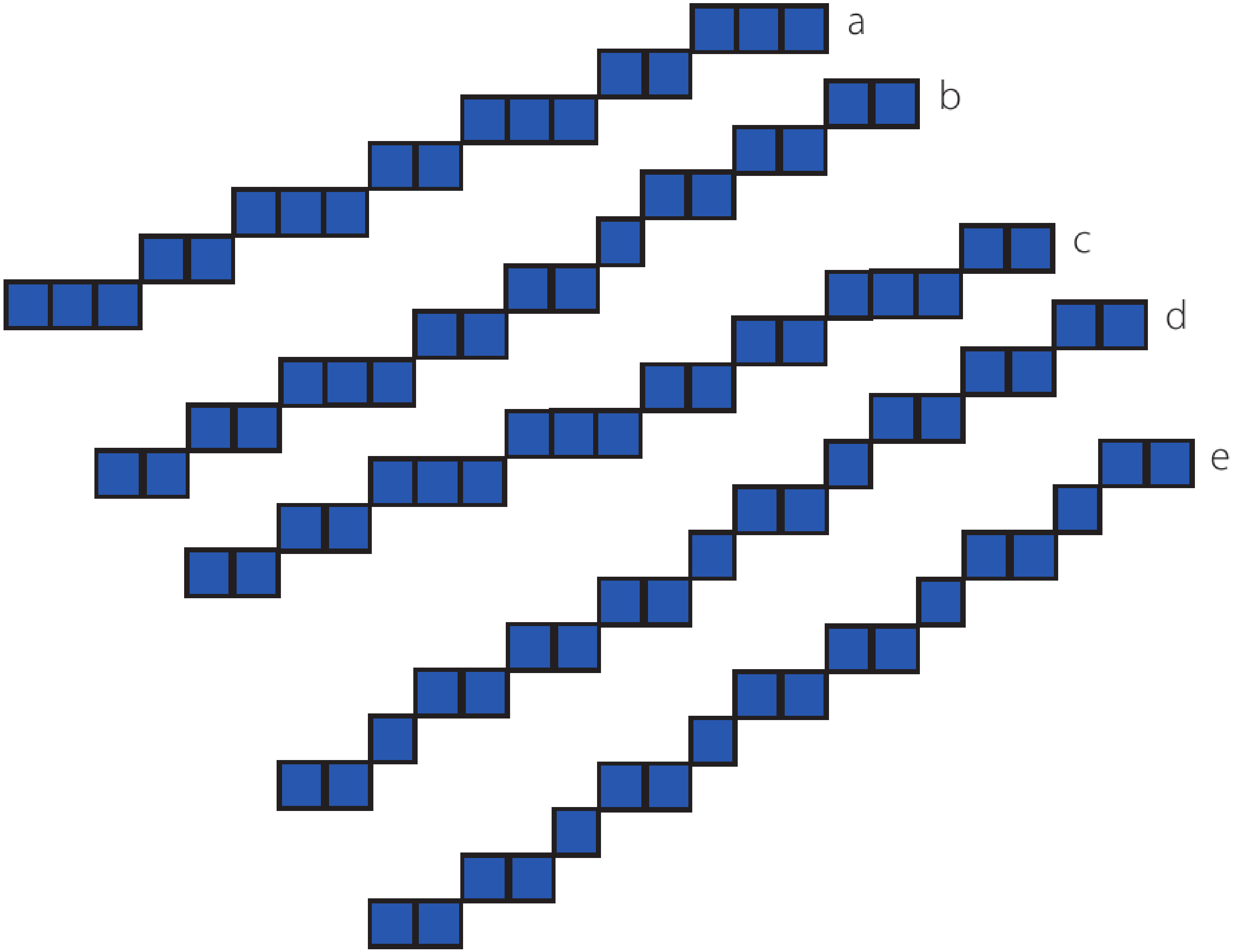


Digital Straight Lines

Recognition of digital straight line segments:

“When is an arc the digitization of a Euclidean straight line?”

Which of these arcs are digital straight line segments?



Digital Straight Lines

Each arc consists of two “blocks” K and L.

Condition 1: There are at most two block lengths, l_L and $l_K=l_L+1$.

(b) has block lengths 1, 2, and 3 and is therefore not a digital straight line segment.

Digital Straight Lines

Condition 2: Each occurrence of *one* of the two blocks should be adjacent to the other block both to the left and right.

Example:

(a): KLKLKLK – allowed by Condition 2

(c): KKLLKKLL – not allowed by Condition 2, so (c) is not a digital straight line.

(d): KLKKKLKLKKK – allowed by Condition 2

(e): KKLKLKKLKLK – allowed by Condition 2

Digital Straight Lines

We have considered blocks of order 0 so far. Blocks of order 1 are obtained by maximal segments of blocks of order 0 with only one transition between K^0 and L^0. (Superscript denotes the order.)

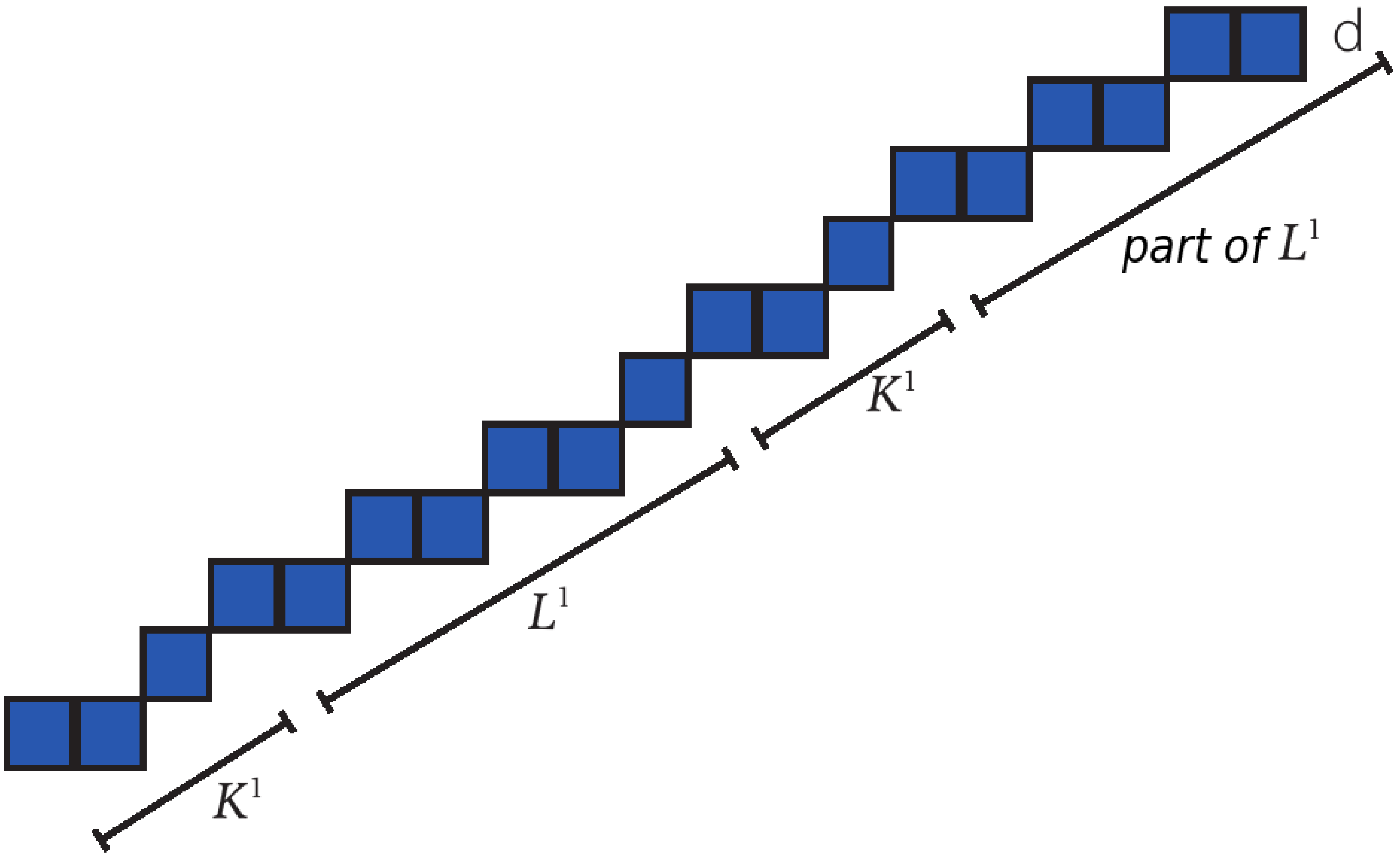
Example:

(a) $K^1K^1K^1$ - length of K^1=1

(d) $K^1L^1K^1L^1$ - length of K^1=1 L^1=3

(e) $K^1L^1K^1L^1$ - length of K^1=2 L^1=1

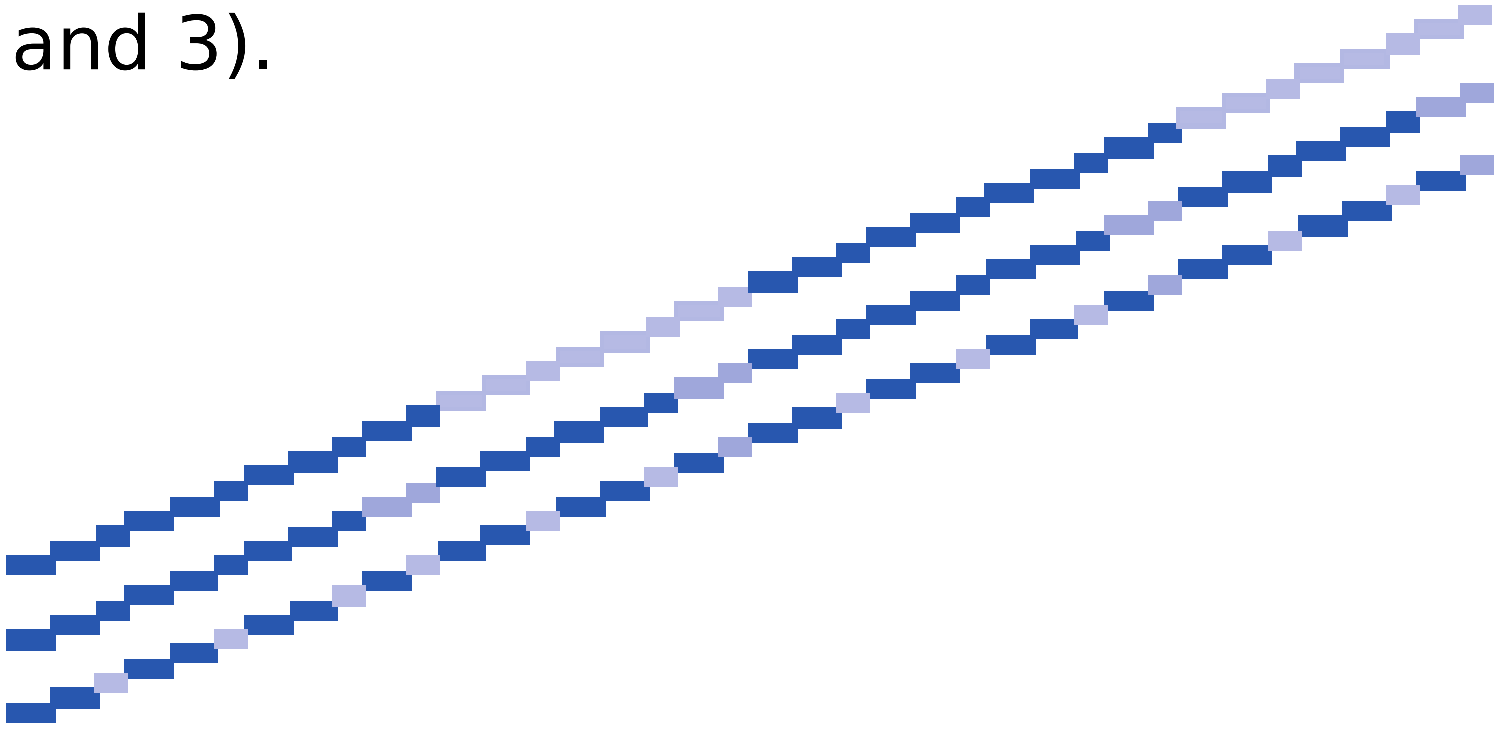
Digital Straight Lines



Digital Straight Lines

Condition 3: Condition 1&2 should hold for blocks of all orders.

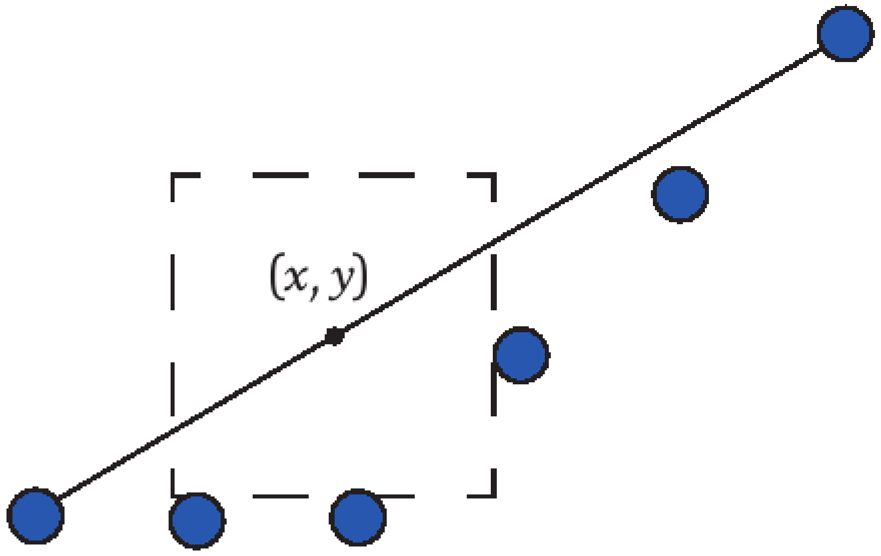
Condition 3 does not hold for (d) (block lengths 1 and 3).



A digital line partitioned into blocks of order 0, 1, and 2.

Digital Straight Lines
Rosenfelds Chord-property

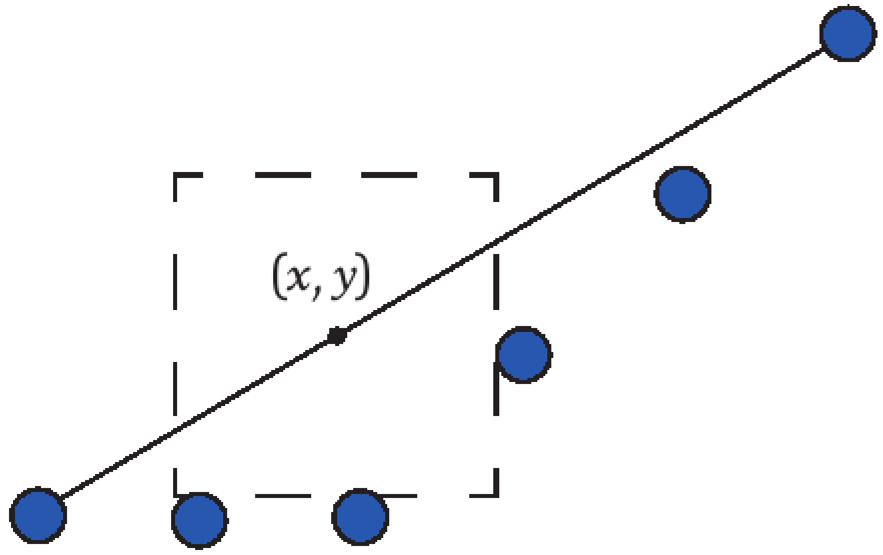
Let pq denote the Euclidean straight line segment between p and q. pq lies *near* a digital object S if, for any (real) point (x,y) of pq, there exists a grid point (i,j) of S such that max (|i-x|,|j-y|) < 1.



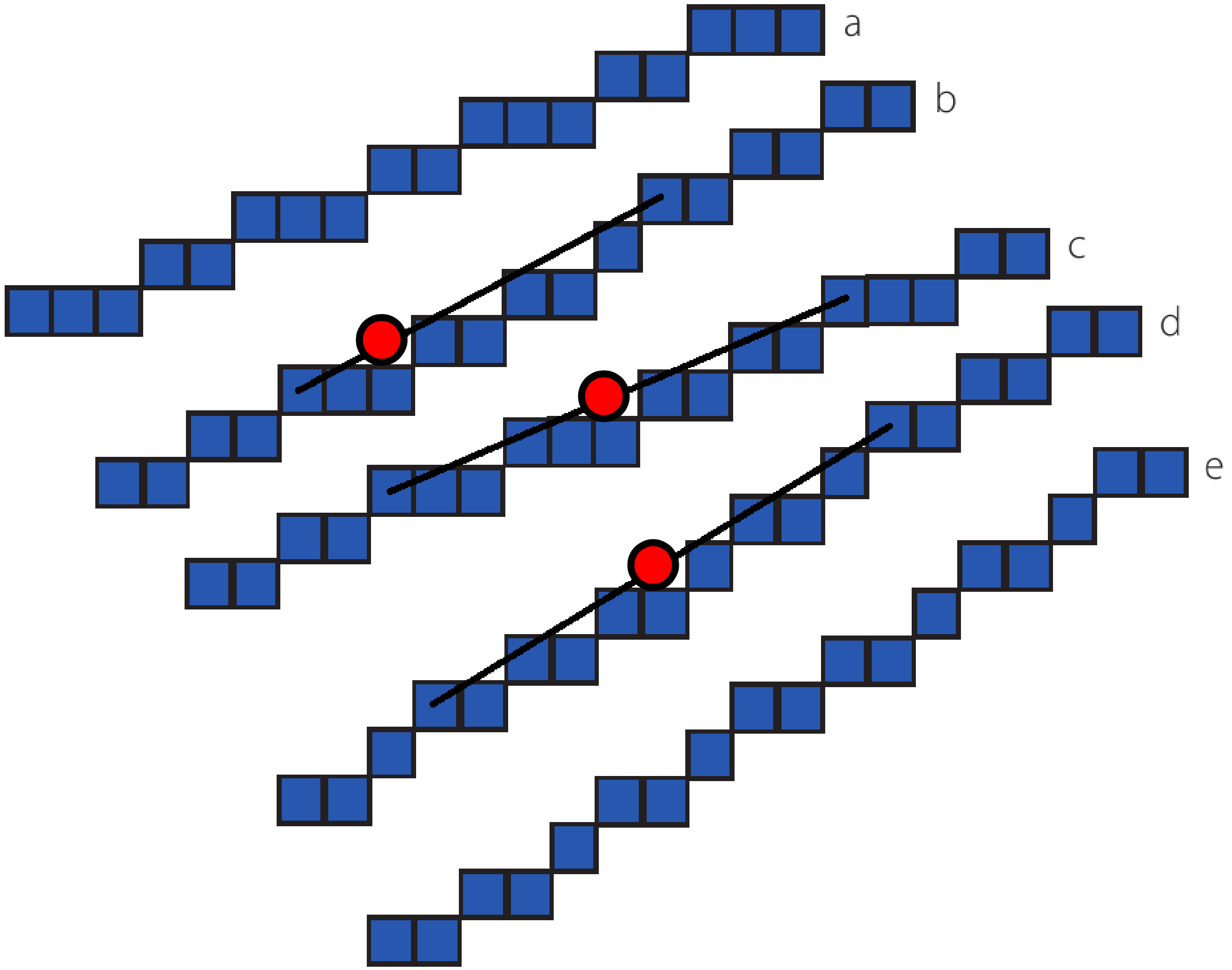
Digital Straight Lines
Rosenfelds Chord-property

Rosenfelds Chord-property from 1974:

A digital arc S is a digital straight line segment if each point on a Euclidean straight line segment between any two points p,q in S is *near* S.



Digital Straight Lines
Rosenfelds Chord-property



Measuring distances in an image

Distance functions $\boldsymbol{x}=(x_1,x_2),\ \boldsymbol{y}=(y_1,y_2) \in \boldsymbol{Z}^2$

$$d_{euclidean} = \|\mathbf{x}-\mathbf{y}\|_2 = \sqrt{\left(x_1-y_1\right)^2+\left(x_2-y_2\right)^2} \qquad l_2$$

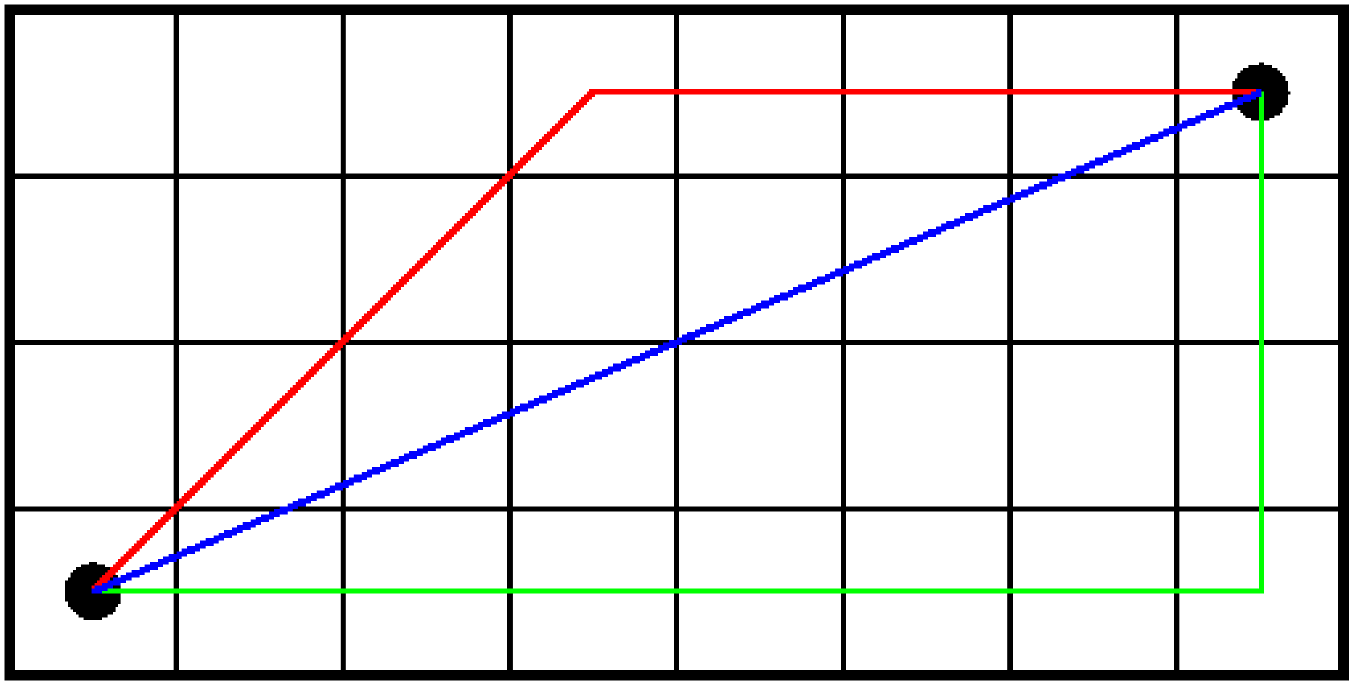
$$d_{cityblock} = \|\mathbf{x}-\mathbf{y}\|_1 = |x_1-y_1|+|x_2-y_2| \qquad l_1$$

$$d_{chessboard} = \|\mathbf{x}-\mathbf{y}\|_\infty = \max\left(|x_1-y_1|,|x_2-y_2|\right) \qquad l_\infty$$

Intuitively: Euclidean distance

Often represented by a distance transform

Each object grid point is labeled with the distance to its closest grid point in background



Euclidean	$\sqrt{7^2+3^2}$
city block, number of steps in 4-path	$7+3$
chessboard, number of steps in 8-path	$3+4$

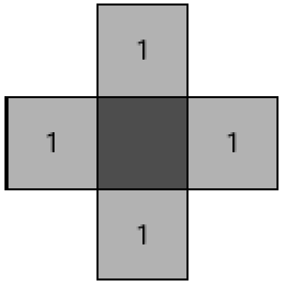
Distance functions in digital images

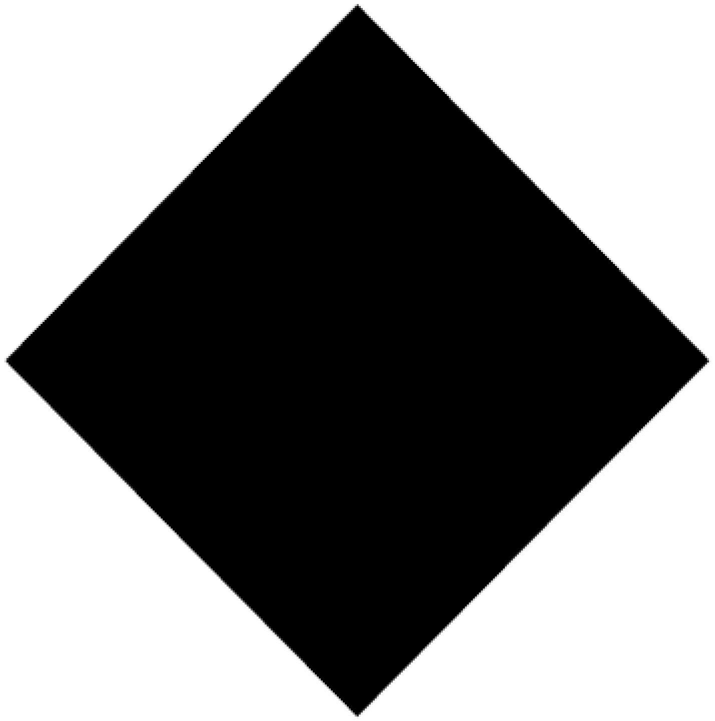
A path-based distance function is defined as the minimal cost-path.

In digital images, two classes of distance functions are considered:
path-based and *not path-based*.

Simple path-based distances

city-block





Weights

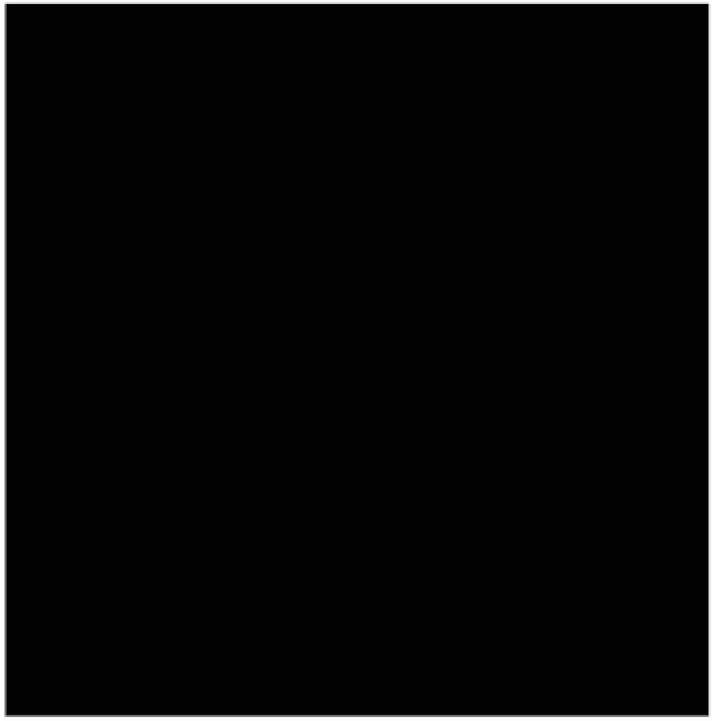
Shape of ball

Simple path-based distances

chessboard

1	1	1
1		1
1	1	1

Weights



Shape of ball

Generalizations of simple path-based distances

-Defined as the cost of a minimal path

city block and **chessboard**:
fixed neighborhood with unit weights

Weighted distances:
fixed neighborhood with weights

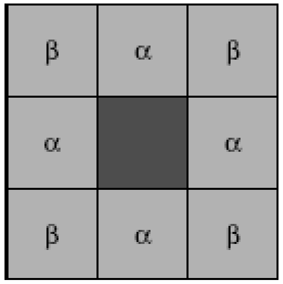
Distances based on neighborhood sequences:
variable neighborhood with unit weights

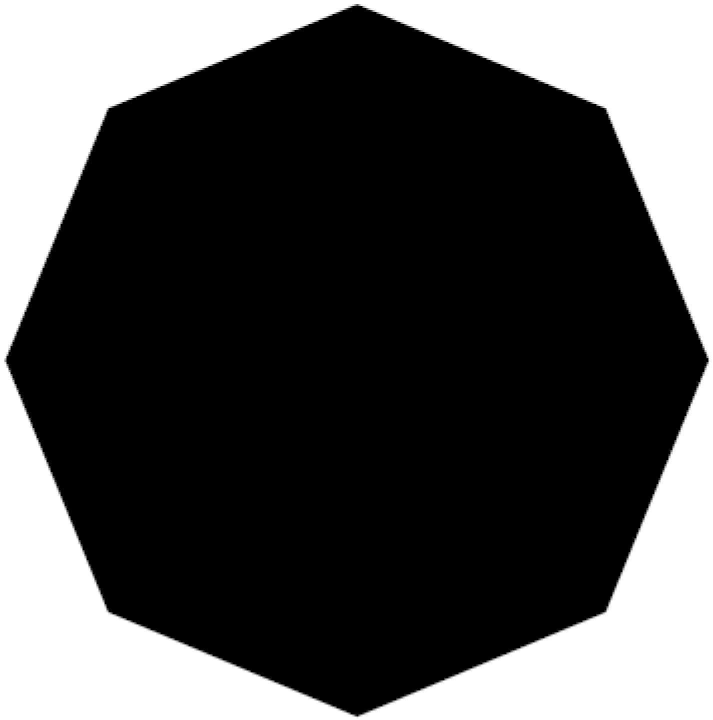
Weighted distances

The minimal cost-path when the local steps are weighted.

Usually one weight for each type of neighbor.

Weighted distances

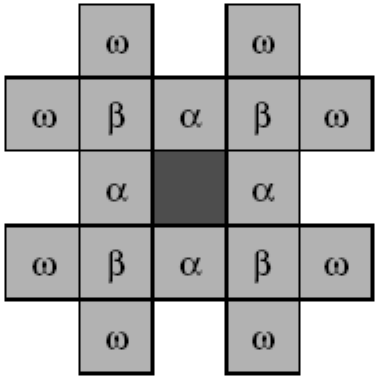


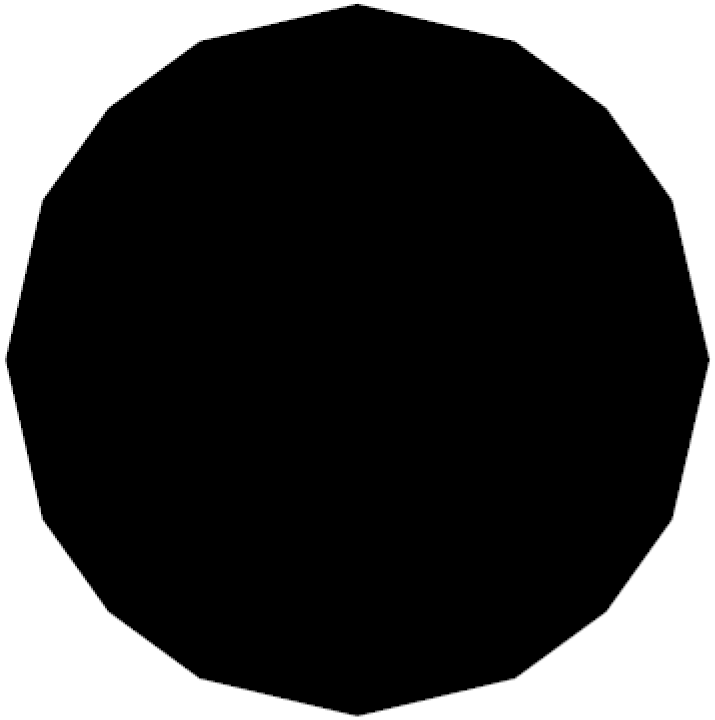


Weights

Shape of ball
with optimal weights

Weighted distances





Weights

Shape of ball
with optimal weights

Distances based on neighborhood sequences

Abbreviated ns-distances

The size of the neighborhood allowed in each step is given by a *neighborhood sequence* B.

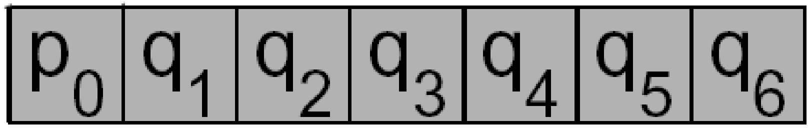
The elements in B are 1:s and 2:s, 1 corresponds to a city-block step and 2 corresponds to a chessboard step.

Element i is denoted b(i)

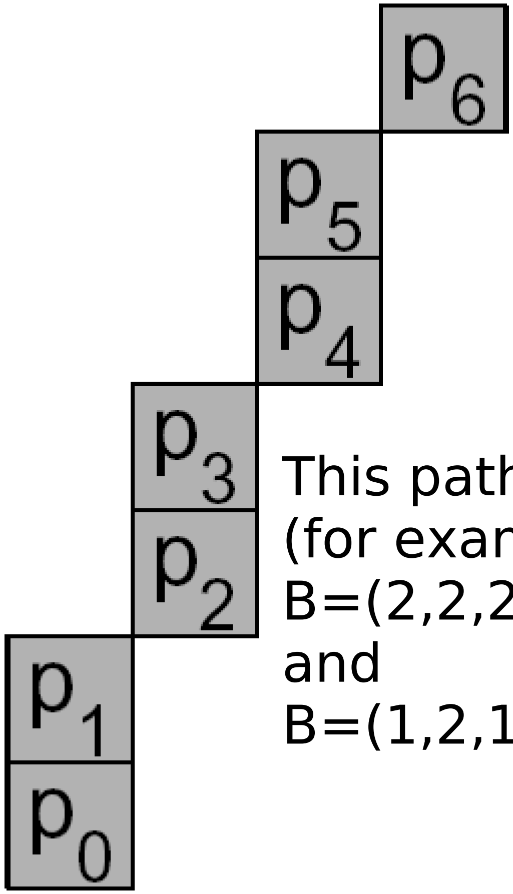
The distance is defined as the shortest path allowed by the neighborhood sequence.

Distances based on neighborhood sequences

Examples

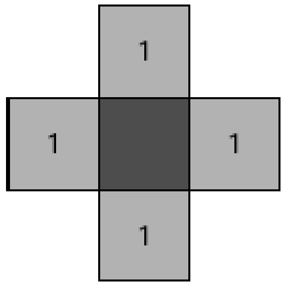


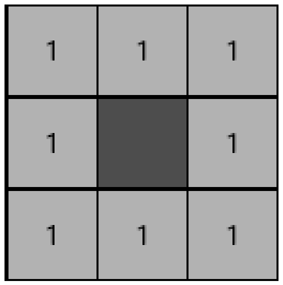
This path is a B-path for any B

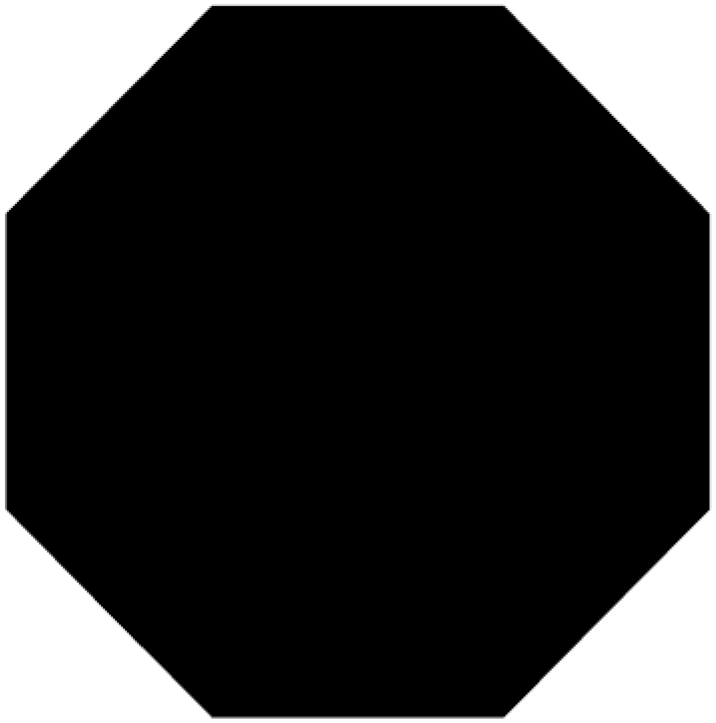


This path is a B-path for (for example)
B=(2,2,2,2,...) [chessboard]
and
B=(1,2,1,2,...) [octagonal]

ns-distances







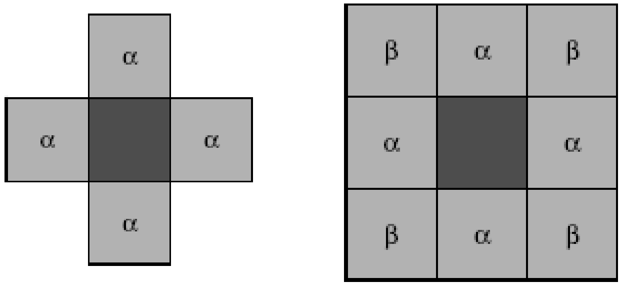
Neighborhoods
and weights

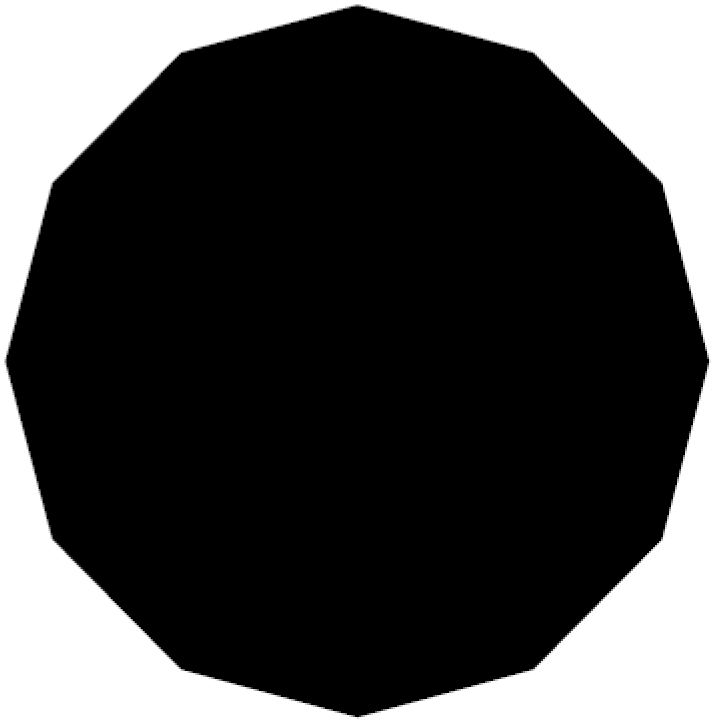
Shape of ball
with optimal
neighborhood sequence

Weighted ns-distances

Using both a neighborhood sequence *and* weights to define distance

Weighted ns-distances





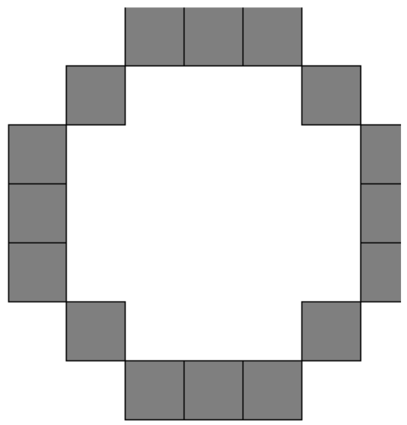
Neighborhoods
and weights

Shape of ball
with optimal weights and
neighborhood sequence

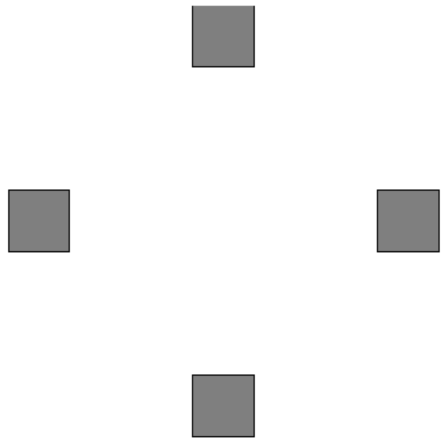
Comparison of distance functions I

Circle, radius 3
Connected circles?

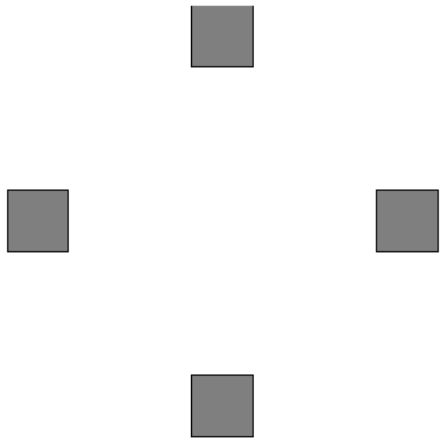
Ns-distance,
B=(1,2,1,2,...)



weighted distance with weights <1,4/3>



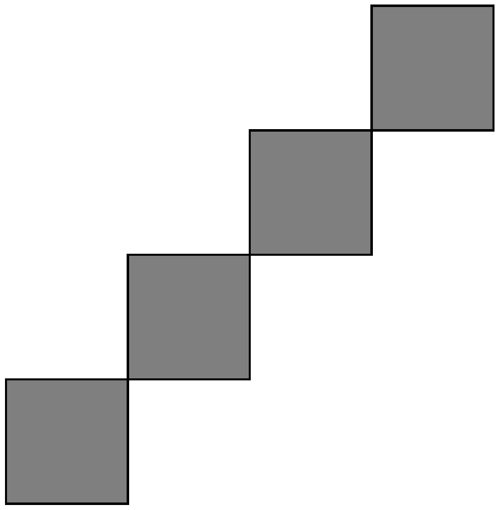
Euclidean distance



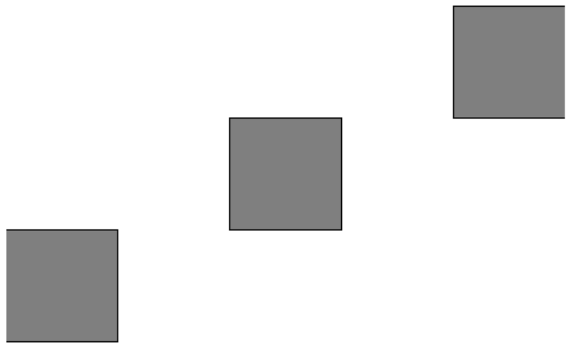
Comparison of distance functions II

Connected paths?

chessboard distance



5x5 weighted distance



Comparison of distance functions III

	Connected circles	Connected paths	Computational efficiency	Rotational invariance
city-block	yes	yes	high	low
chessboard	yes	yes	high	low
weighted	no	yes	high	medium
ns-distance	yes	yes	medium	medium
weighted ns-distance	no	yes	medium	high
weighted 5x5	no	no	medium	high
Euclidean	no	-	low	optimal

Distance transform (DT)

Representation of distances in an image

Gray-level image

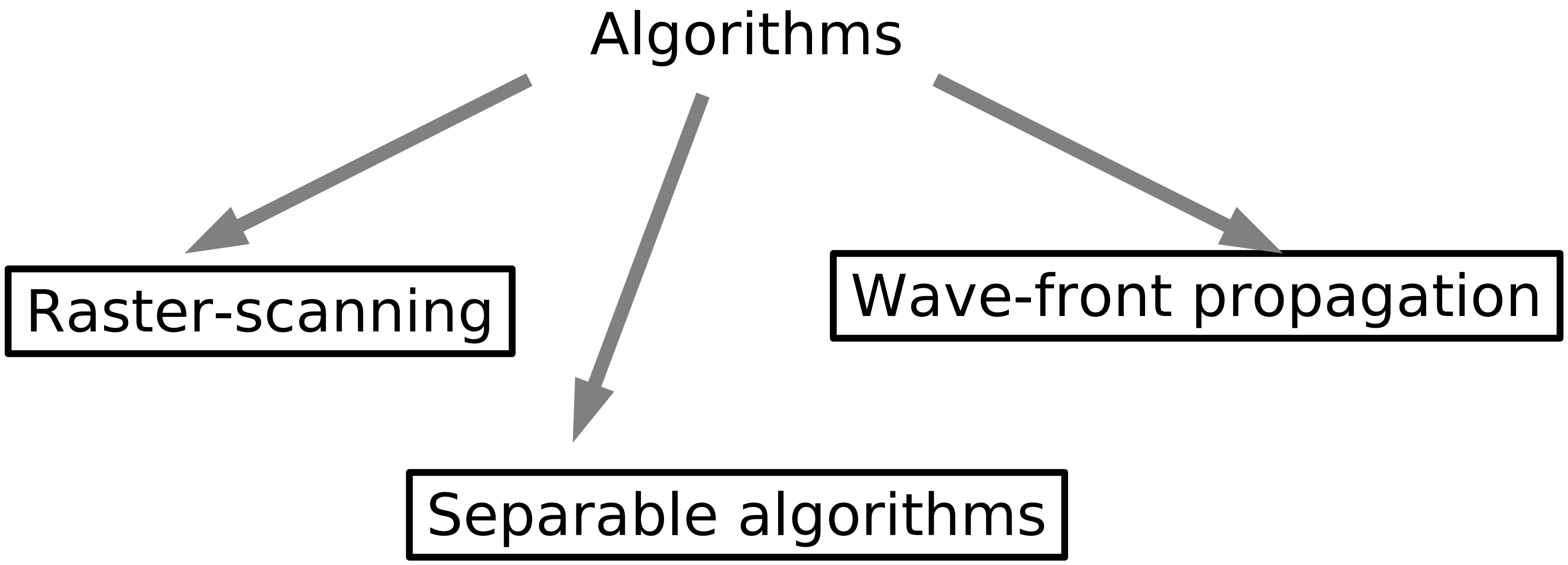
non-zero values on object pixels

each object pixel is labeled with the distance to its closest pixel in the background

Def:

$$\boxed{\begin{aligned} DT : \mathcal{I}_{\mathbb{G}} &\rightarrow \mathbb{R}_0^+ \text{ defined by} \\ \mathbf{p} &\mapsto d\left(\mathbf{p}, \overline{X}\right), \text{ where} \\ d\left(\mathbf{p}, \overline{X}\right) &= \min_{\mathbf{q} \in \overline{X}} \left\{ d\left(\mathbf{p}, \mathbf{q}\right) \right\}. \end{aligned}}$$

Computing distance transforms



Raster-scanning

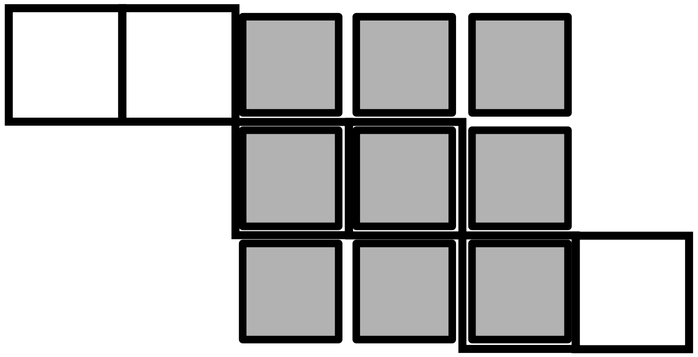
The image is scanned row-by-row or column-by-column in a predefined order.

Distance information (scalars or vectors) are propagated using a small neighborhood

See image analysis, first course

DTs for the path-based distances are error-free

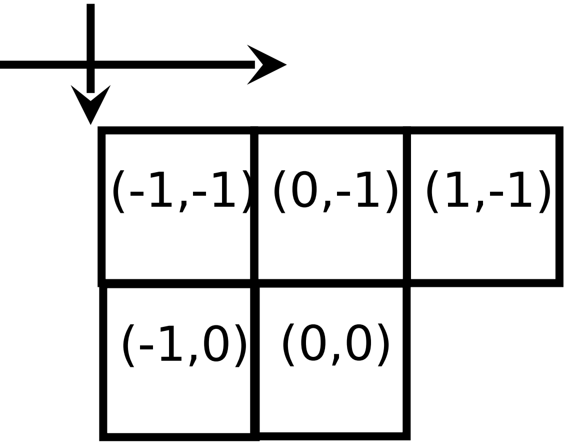
By using a 3x3 neighborhood,
any 8-connected path can be “tracked”.
Since the distance is defined as
the cost of a path between pixels,
the propagation is complete.



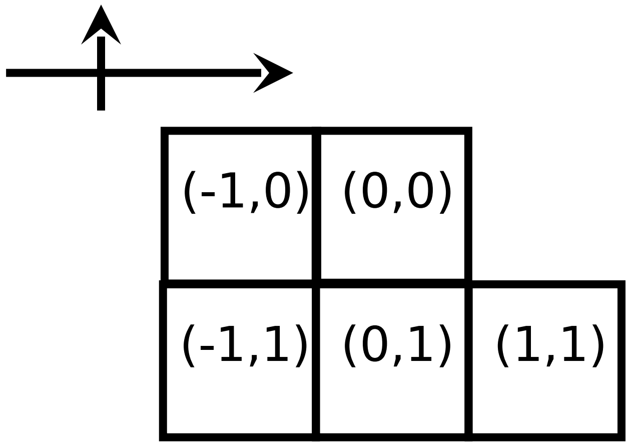
Raster-scanning algorithm for Euclidean DT

Propagate vectors.

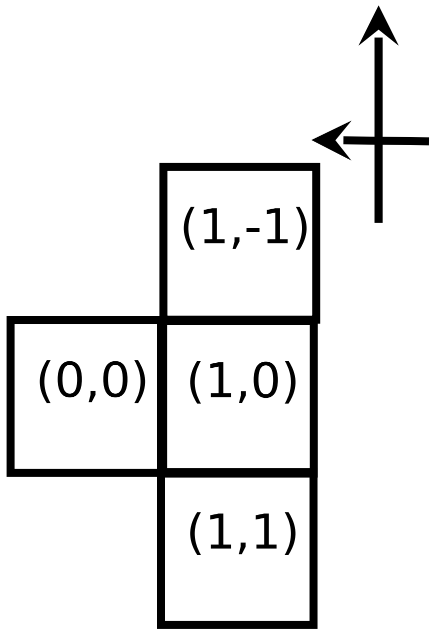
Idea: The shortest vector to a background grid point is propagated using (small) masks.



Mask 1



Mask 2



Mask 3

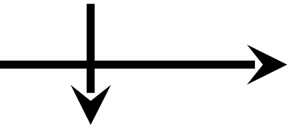
Raster-scanning algorithm for Euclidean DT

Example   Initial image

0	0	0	0	0	0	0	0
0	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	0	0	0
0	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	0	0
0	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	0
0	0	(∞,∞)	(∞,∞)	(∞,∞)	(∞,∞)	0	0
0	0	0	(∞,∞)	0	0	0	0

Raster-scanning algorithm for Euclidean DT

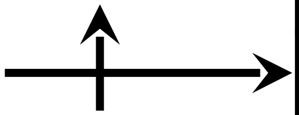
Example After first scan



0	0	0	0	0	0	0	0
0	(-1,0)	(0,-1)	(0,-1)	(0,-1)	0	0	0
0	(-1,0)	(-2,0)	(0,-2)	(1,-1)	(0,-1)	0	0
0	(-1,0)	(-2,0)	(2,-2)	(1,-2)	(1,-1)	(0,-1)	0
0	0	(-1,0)	(-2,0)	(2,-2)	(1,-2)	0	0
0	0	0	(-1,0)	0	0	0	0

Raster-scanning algorithm for Euclidean DT

Example After second scan

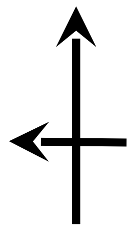


0	0	0	0	0	0	0	0
0	(-1,0)	(0,-1)	(0,-1)	(0,-1)	0	0	0
0	(-1,0)	(-2,0)	(0,-2)	(1,-1)	(0,-1)	0	0
0	(-1,0)	(-1,1)	(-2,1)	(0,2)	(1,-1)	(0,-1)	0
0	0	(-1,0)	(-1,1)	(0,1)	(0,1)	0	0
0	0	0	(-1,0)	0	0	0	0

Raster-scanning algorithm for Euclidean DT

Example   After third scan

0	0	0	0	0	0	0	0
0	(-1,0)	(0,-1)	(0,-1)	(0,-1)	0	0	0
0	(-1,0)	(-2,0)	(0,-2)	(1,-1)	(0,-1)	0	0
0	(-1,0)	(-1,1)	(-2,1)	(0,2)	(1,-1)	(0,-1)	0
0	0	(-1,0)	(-1,1)	(0,1)	(0,1)	0	0
0	0	0	(-1,0)	0	0	0	0



Raster-scanning algorithm for Euclidean DT

Example. Last step: compute the distance values from the vectors.

0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0
0	1	2	2	$\sqrt{2}$	1	0	0
0	1	$\sqrt{2}$	$\sqrt{5}$	2	$\sqrt{2}$	1	0
0	0	1	$\sqrt{2}$	1	1	0	0
0	0	0	1	0	0	0	0

Errors in the Euclidean DT Vector propagation

A small neighborhood
does not hold enough information

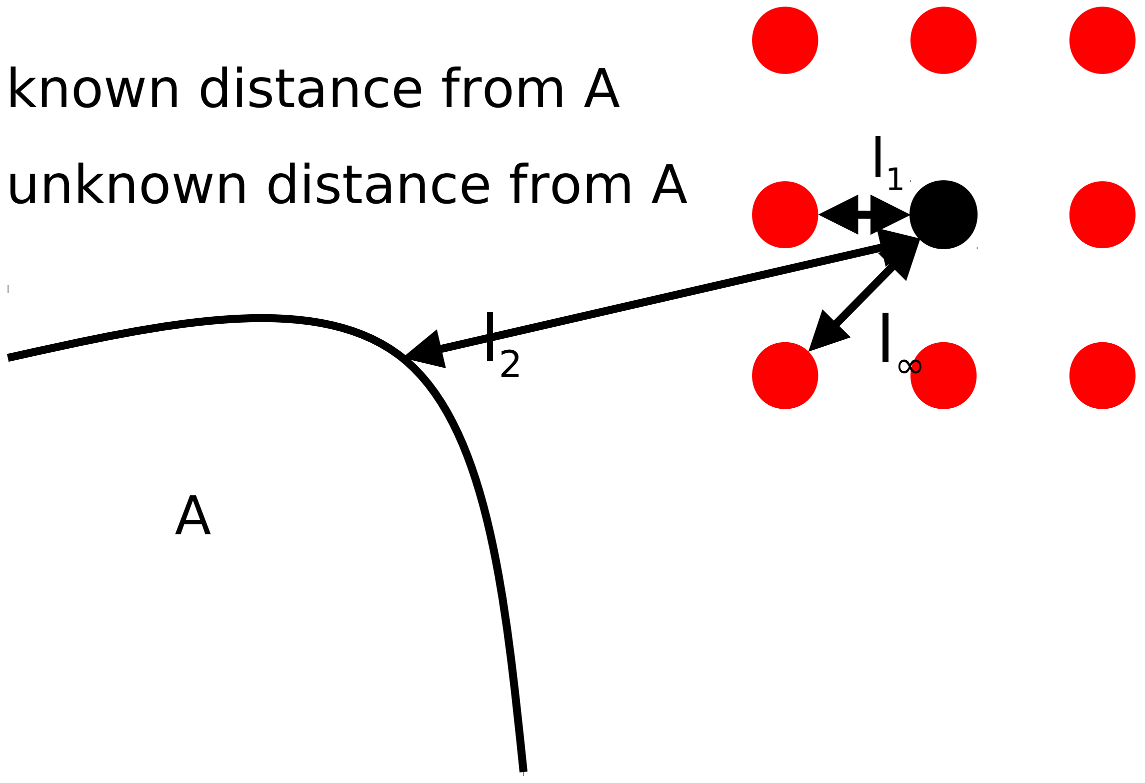
The reason is that
the Euclidean distance is
not a path-based distance.

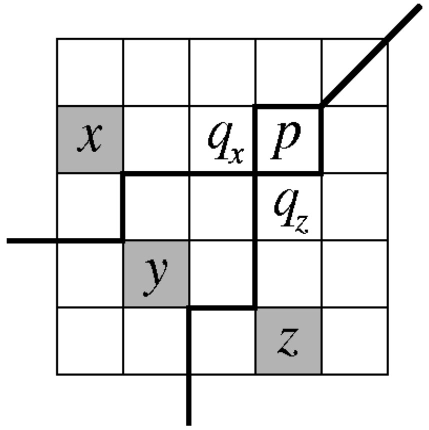
Errors in Euclidean DT using local neighborhoods

A 3x3 neighborhood does not hold enough information for the Euclidean distance.
-Even if vectors are propagated(!)

🟥 known distance from A

● unknown distance from A





Errors in the Euclidean DT

A small neighborhood
does not hold enough information

Solution:
Use a larger neighborhood
Increase size of neighborhood only when needed

Wave-front propagation

Distance information (scalars or vectors) are propagated using a small neighborhood at each pixel in the wave-front starting with small distance values at the border.

- Propagating scalars
- Propagating vectors
- Approx. DE numerically – Fast Marching Methods (FMM)

Wave-front propagation for weighted DT

Initially, construct a list with pixels 8-connected to the object.
Propagate distance values from the wave-front.
Add new elements to the wave-front.

Mask (general)

+b	+a	+b
+a	+0	+a
+b	+a	+b

Mask, <3,4>-weighted distance

+4	+3	+4
+3	+0	+3
+4	+3	+4

Wave-front propagation for <3,4>-weighted DT

Example. First step: initialize wave-front

⓪	⓪	⓪	⓪	⓪	⓪	0	0
⓪	∞	∞	∞	∞	⓪	⓪	0
⓪	∞	∞	∞	∞	∞	⓪	⓪
⓪	∞	∞	∞	∞	∞	∞	⓪
⓪	⓪	∞	∞	∞	∞	⓪	⓪
0	⓪	⓪	∞	⓪	⓪	⓪	0

Wave-front propagation for <3,4>-weighted DT

Example. Propagate values from the wave-front

0	0	0	0	0	0	0	0
0	3	3	3	3	0	0	0
0	3	∞	∞	4	3	0	0
0	3	4	∞	∞	4	3	0
0	0	3	4	3	3	0	0
0	0	0	3	0	0	0	0

Wave-front propagation for <3,4>-weighted DT

Example. Propagate values from the wave-front

0	0	0	0	0	0	0	0
0	3	3	3	3	0	0	0
0	3	⑥	⑥	4	3	0	0
0	3	4	⑦	⑥	4	3	0
0	0	3	4	3	3	0	0
0	0	0	3	0	0	0	0

Wave-front propagation for ns-distance DT

Propagate distance values from the neighborhood that is allowed.

Masks:

	1	
1	0	1
	1	

1	1	1
1	0	1
1	1	1

Wave-front propagation for ns-distance DT

Example. B=(1,2,1,2,...) First step: initialize wave-front

● 0	● 0	● 0	● 0	● 0	● 0	0	0
● 0	∞	∞	∞	∞	● 0	● 0	0
● 0	∞	∞	∞	∞	∞	● 0	● 0
● 0	∞	∞	∞	∞	∞	∞	● 0
● 0	● 0	∞	∞	∞	∞	● 0	● 0
0	● 0	● 0	∞	● 0	● 0	● 0	0

Wave-front propagation for ns-distance DT

Example. Propagate unit values from the wave-front using the *first* element in B=(1,2,1,2,...)

0	0	0	0	0	0	0	0
0	1	1	1	1	0	0	0
0	1	∞	∞	∞	1	0	0
0	1	∞	∞	∞	∞	1	0
0	0	1	∞	1	1	0	0
0	0	0	1	0	0	0	0

Wave-front propagation for ns-distance DT

Example. Propagate unit values from the wave-front using the *second* element in B=(1,2,1,2,...)

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

Computing Euclidean DT by the fast-marching method

A differential equation is approximated:

$$\|\nabla I\| = 1.$$

Background pixels are frozen.
Create a list with pixels that should be updated.
Update all pixels in the list
    (using a finite difference approximation).
For the updated pixel with lowest distance value:
    remove from queue and freeze.

Fast-Marching

Example. First step: initialize wave-front and update distance values

0	0	0	0	0	0	0	0
0	● ∞	● ∞	● ∞	● ∞	0	0	0
0	● ∞	∞	∞	∞	● ∞	0	0
0	● ∞	∞	∞	∞	∞	● ∞	0
0	0	● ∞	∞	● ∞	● ∞	0	0
0	0	0	● ∞	0	0	0	0

Fast-Marching

Example. First step: initialize wave-front and update distance values

$(T-0)^2+(T-0)^2=1$
$\Rightarrow T=1/\sqrt{2}\approx 0.7$

$(T-0)^2=1$
$\Rightarrow T=1$

0	0	0	0	0	0	0	0
0	● ~0.7	● 1	● 1	● ~0.7	0	0	0
0	● 1	∞	∞	∞	● ~0.7	0	0
0	● ~0.7	∞	∞	∞	∞	● ~0.7	0
0	0	● ~0.7	∞	● 1	● ~0.7	0	0
0	0	0	● 1	0	0	0	0

Fast-Marching

Example. For each element in the list:

approximate |grad T|=1 numerically using frozen values.

$$(\partial T/\partial x)^2+(\partial T/\partial y)^2=1$$

$(T-1/\sqrt{2})^2=1$

$\Rightarrow T=1/\sqrt{2}+1\approx 1.7$

0	0	0	0	0	0	0	0
0	~0.7	● 1	● 1	● ~0.7	0	0	0
0	● 1	∞	∞	∞	● ~0.7	0	0
0	~0.7	● ~1.7	∞	∞	∞	● ~0.7	0
0	0	● ~0.7	∞	● 1	● ~0.7	0	0
0	0	0	● 1	0	0	0	0

Fast-Marching

Example. For each element in the list:

approximate |grad T|=1 numerically using frozen values.

$$(\partial T/\partial x)^2+(\partial T/\partial y)^2=1$$

0	0	0	0	0	0	0	0
0	~0.7	● 1	● 1	● ~0.7	0	0	0
0	● 1	∞	∞	∞	● ~0.7	0	0
0	~0.7	● ~1.4	∞	∞	∞	● ~0.7	0
0	0	~0.7	● ~1.7	● 1	● ~0.7	0	0
0	0	0	● 1	0	0	0	0

Fast-Marching

Example. For each element in the list:

approximate |grad T|=1 numerically using frozen values.

$$(\partial T/\partial x)^2+(\partial T/\partial y)^2=1$$

0	0	0	0	0	0	0	0
0	~0.7	● 1	● 1	~0.7	0	0	0
0	● 1	∞	∞	● ~1.4	~0.7	0	0
0	~0.7	● ~1.4	∞	∞	● ~1.4	~0.7	0
0	0	~0.7	● ~1.7	● 1	~0.7	0	0
0	0	0	● 1	0	0	0	0

Fast-Marching

Final result

0	0	0	0	0	0	0	0
0	~0.7	1	1	~0.7	0	0	0
0	1	~1.7	~1.9	~1.4	~0.7	0	0
0	~0.7	~1.4	~2.2	~1.9	~1.4	~0.7	0
0	0	~0.7	~1.5	1	~0.7	0	0
0	0	0	1	0	0	0	0

Errors in the Euclidean DT
Fast marching

- A small neighborhood does not hold enough information.
- Errors due to the finite difference approximations.

Solutions:
- Use a larger neighborhood.
- Use higher-order approximation of derivatives.

Errors in the DTs

Fast marching

Worst case:

∞	∞	∞
∞	0	∞
∞	∞	∞

$\Rightarrow$

~1.7	1	~1.7
1	0	1
~1.7	1	~1.7

$(T-1)^2+(T-1)^2=1 \Rightarrow T=1+\sqrt{2}/2\approx 1.7.$ Should be $\sqrt{2}$.

Computing Euclidean DT by separable algorithms

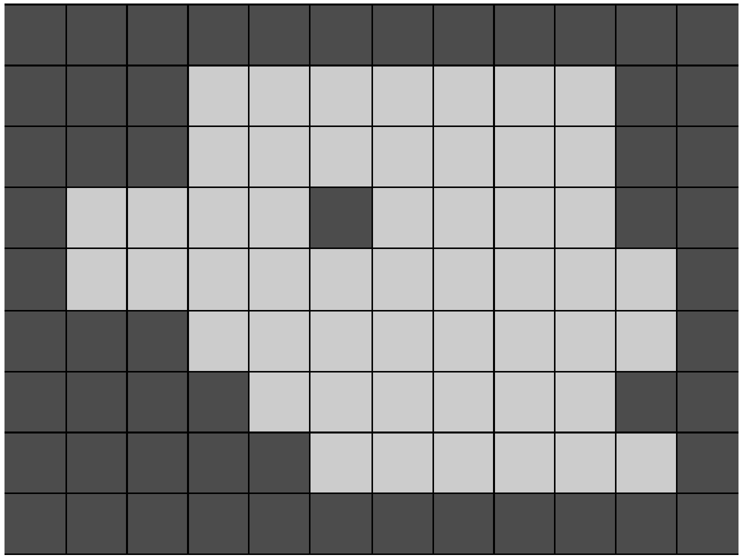
Idea: x^2+y^2 is (additively) separable.

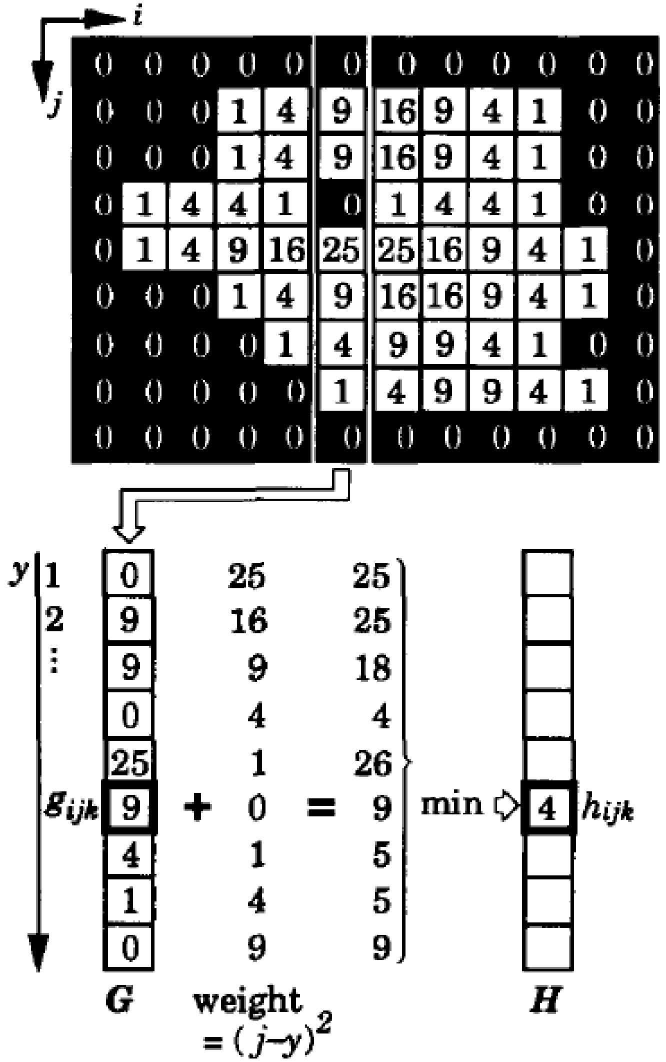
Compute DT in x-direction first
 (two 1D scans needed per row)

Then compute DT in y-direction
 (two 1D scans needed per column)

Computing Euclidean DT by separable algorithms

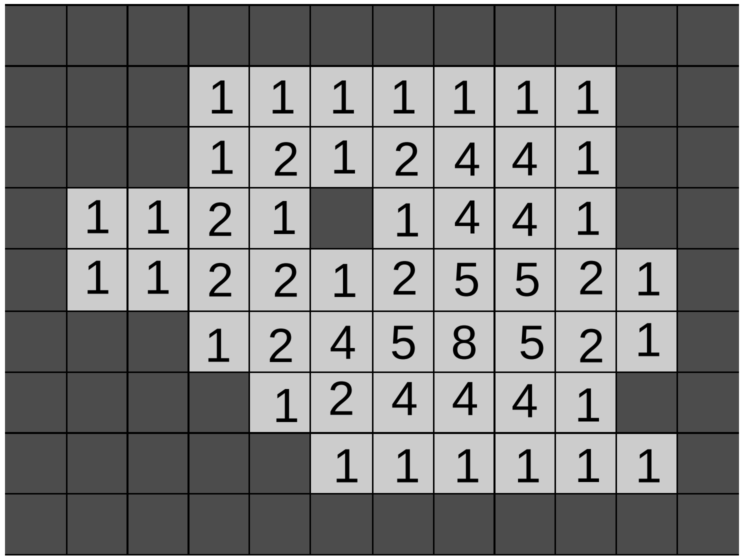
Example: simple
(not linear) algorithm:





Computing Euclidean DT by separable algorithms

Last step: Compute the square root of the values.



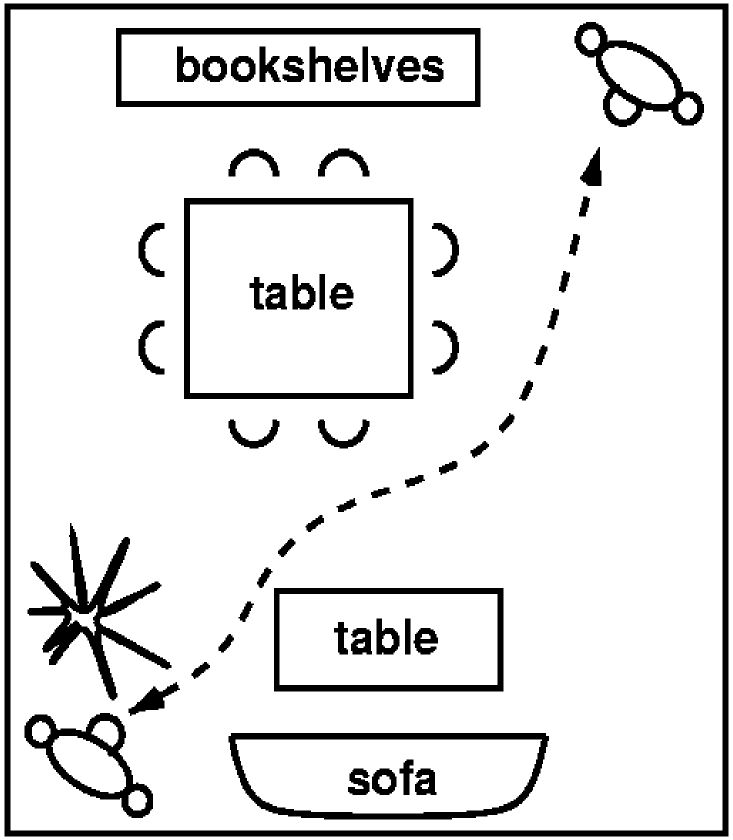
			1	1	1	1	1	1	1		
			1	2	1	2	4	4	1		
	1	1	2	1		1	4	4	1		
	1	1	2	2	1	2	5	5	2	1	
			1	2	4	5	8	5	2	1	
				1	2	4	4	4	1		
					1	1	1	1	1	1	

By handling the list efficiently, the algorithm is linear.
Separable algorithm is error-free!

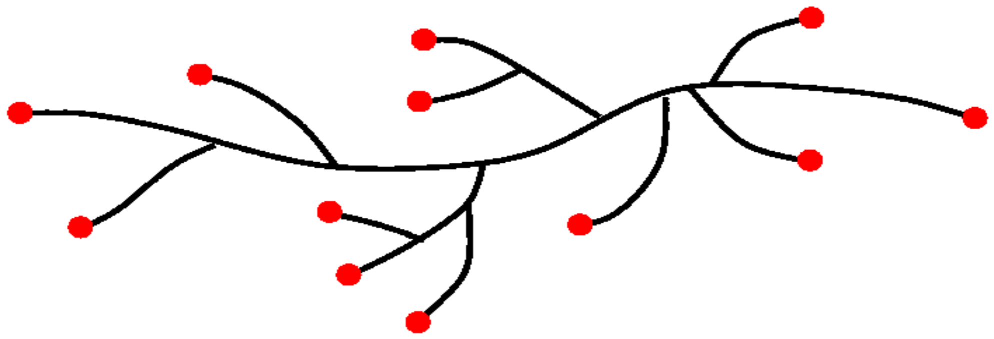
Constrained DT

geodesic DT / DT of non-convex region

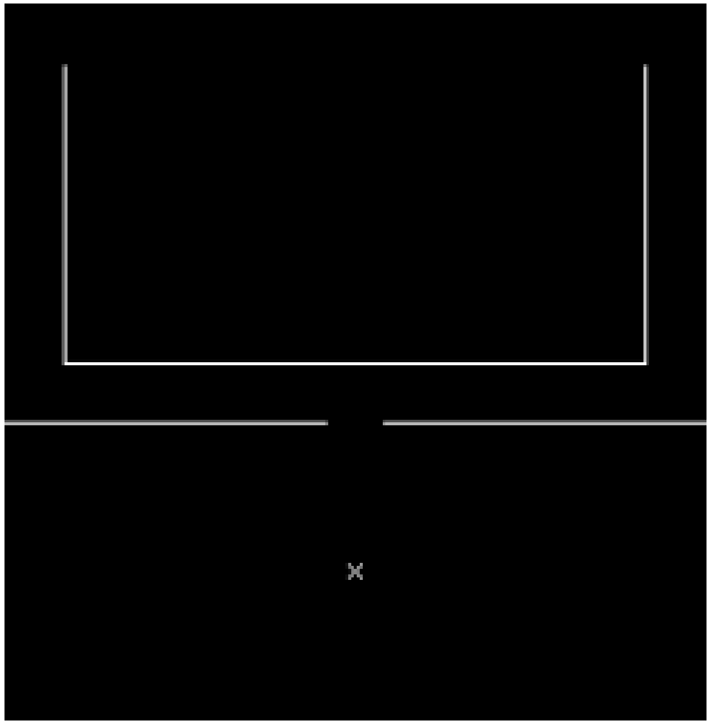
shortest path, avoiding obstacles

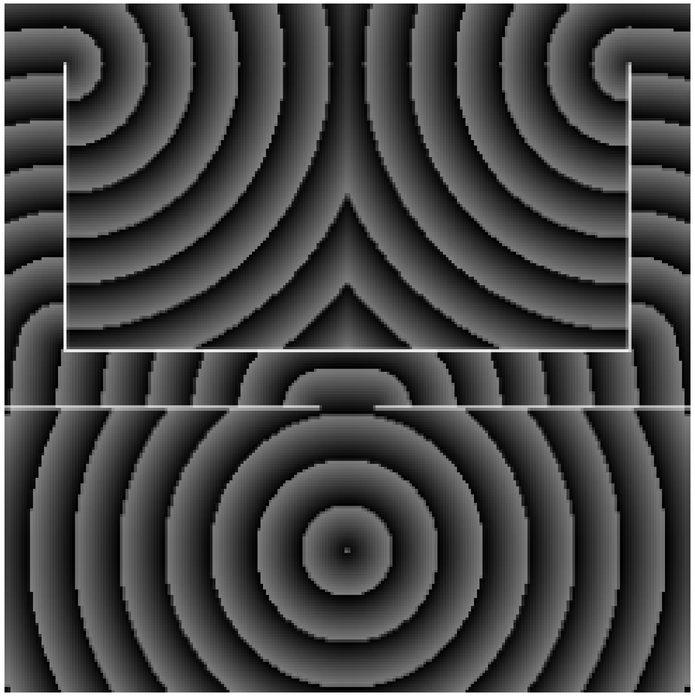


special case: DT of line patterns



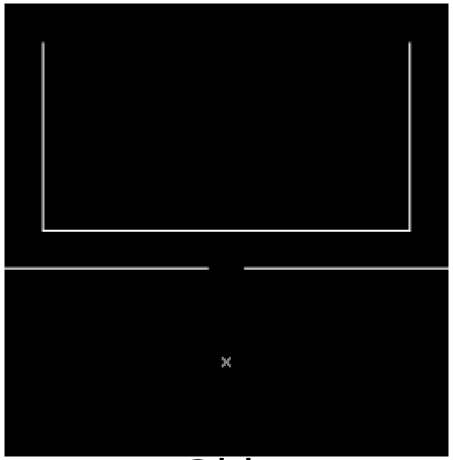
Constrained DT

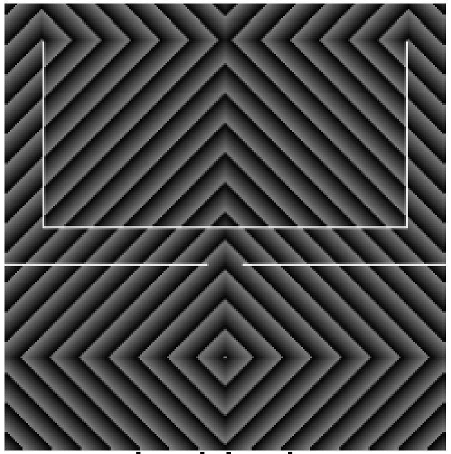

Source pixel and obstacles

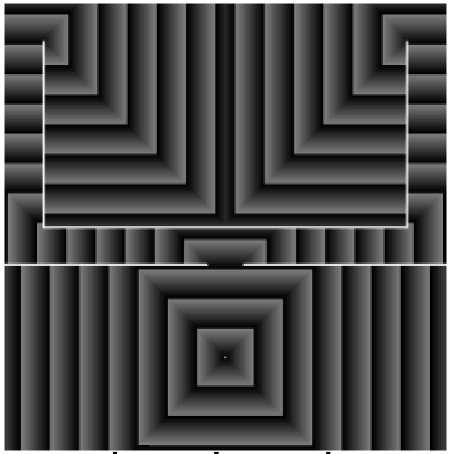

Constrained DT (Euclidean distance)

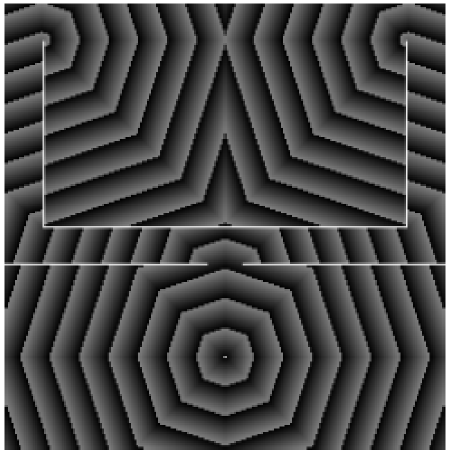
Distance values are shown modulo 16

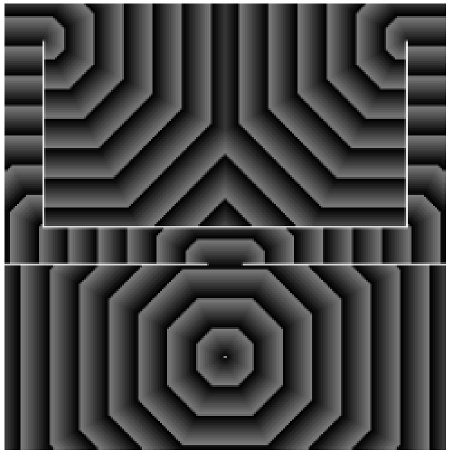
Constrained DT (by wave-front)

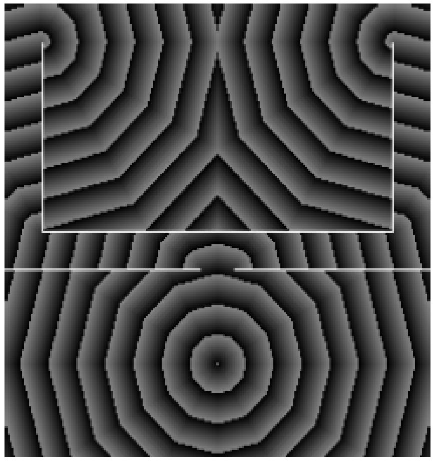

Object

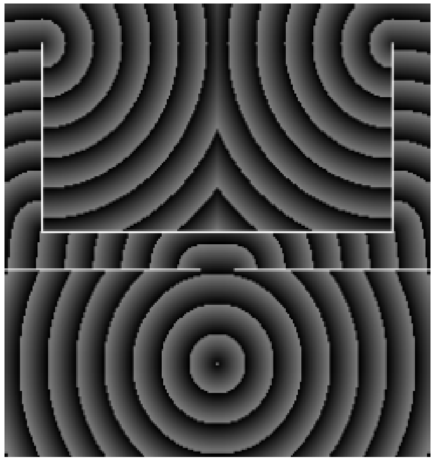

city-block


chessboard


weighted


ns-distance


weighted ns-distance


Euclidean

Distance values are shown modulo 16

Computing distance transforms

Raster-scanning

+Fast, O(n), where n is the number of pixels.
-Not suited for constrained DT.
-Not exact for the Euclidean distance.

Wave-front propagation

+Slower than raster-scanning, O(n log(n)).
+Suited for constrained DT (weighted, ns, FMM).
- Not suited for constrained DT (vector propagation).
- Not exact for the Euclidean distance.

Separable algorithms

+Fast, O(n). But slower than raster-scanning.
+Exact for the Euclidean distance.
-Not suited for constrained DT.

Summary

- Grids, connectivities
- Straight line
- Distance functions and their properties
  - city block, chessboard, weighted distance, ns-distances, Euclidean metric
- Distance transforms and their properties
  - Raster scan algorithm
    - propagating scalars, vectors
  - Wavefront propagation
    - propagation of scalars and vectors
    - Fast marching
  - Separable algorithm
    - Euclidean
  - Constrained DT