Distance Transformations

Minyoung Chung

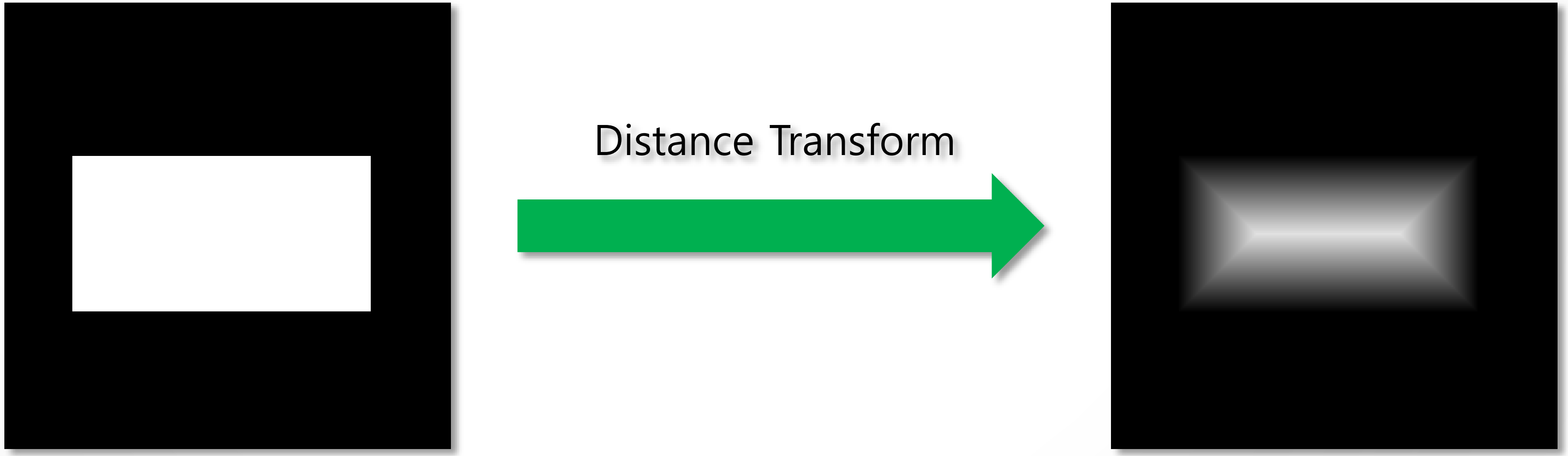
Nov, 2015

Computer Graphics and Image Processing Laboratory
Dept. of Computer Science and Engineering
Seoul National University

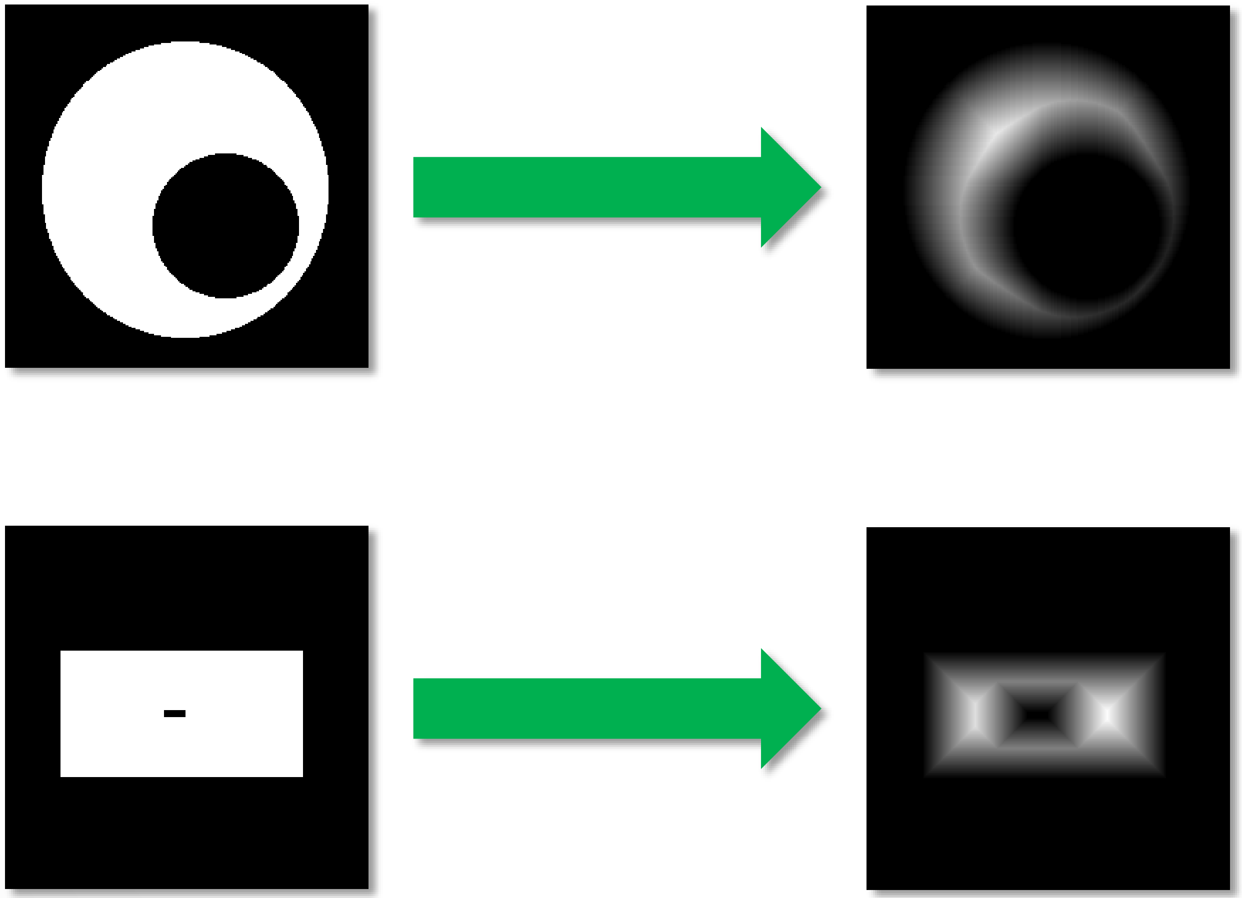
Contents

What is Distance Transform ?

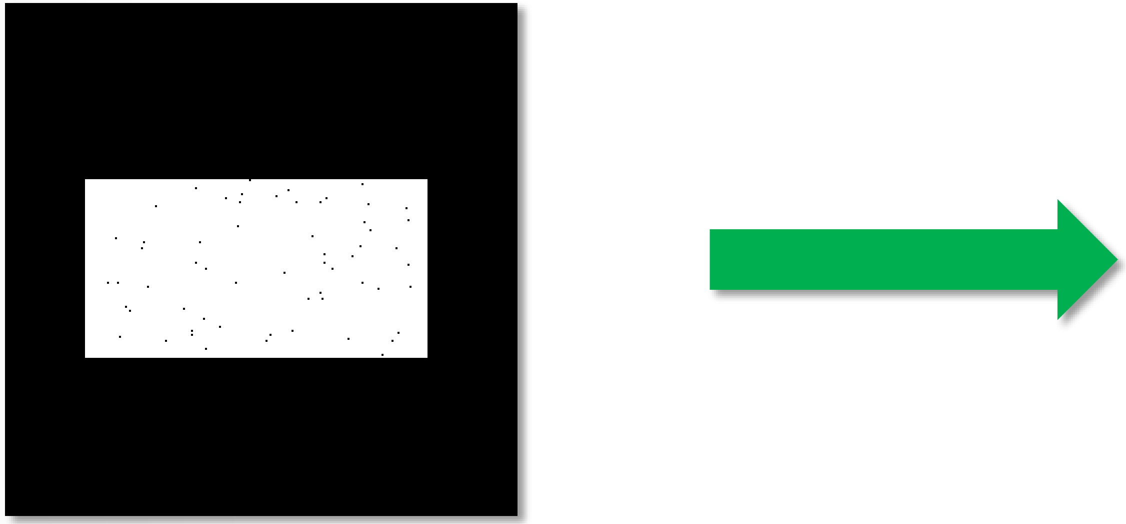
- Labeling of each pixel ***x*** by the distance to the closest point ***y*** in the background.
- $DT(P)[\boldsymbol{x}] = \ min_{y \in P} \mathbf{dist}(\boldsymbol{x}, y)$
  - P : point set of background
  - ***x*** : vector of image position
  - DT(P) : distance map



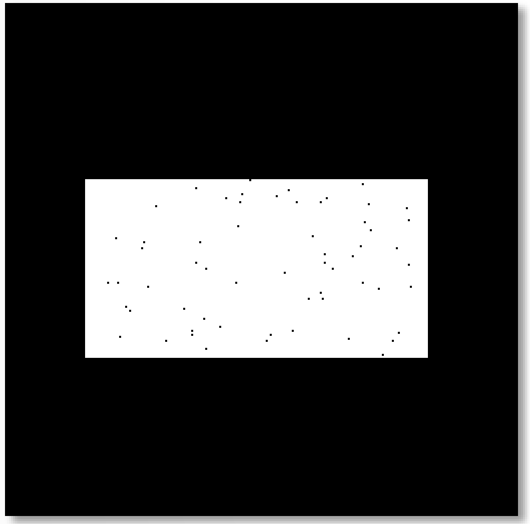
What is Distance Transform ?

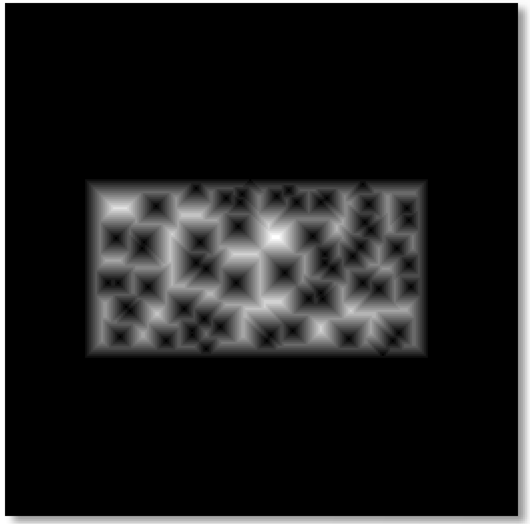


What is Distance Transform ?



What is Distance Transform ?





Applications of Distance Transform

- Morphological image processing
  - Thinning, thickening, …
  - Exact Euclidean distance transform can produce an accurate, reversible skeleton
- Pattern matching and object recognition
- Robot collision avoidance
- Path planning and navigation
- Medical image processing
  - Surface registration
  - Non-rigid image registration
  - Point-to-surface distance
  - Morphological image segmentation
  - Visualization
  - …
- …

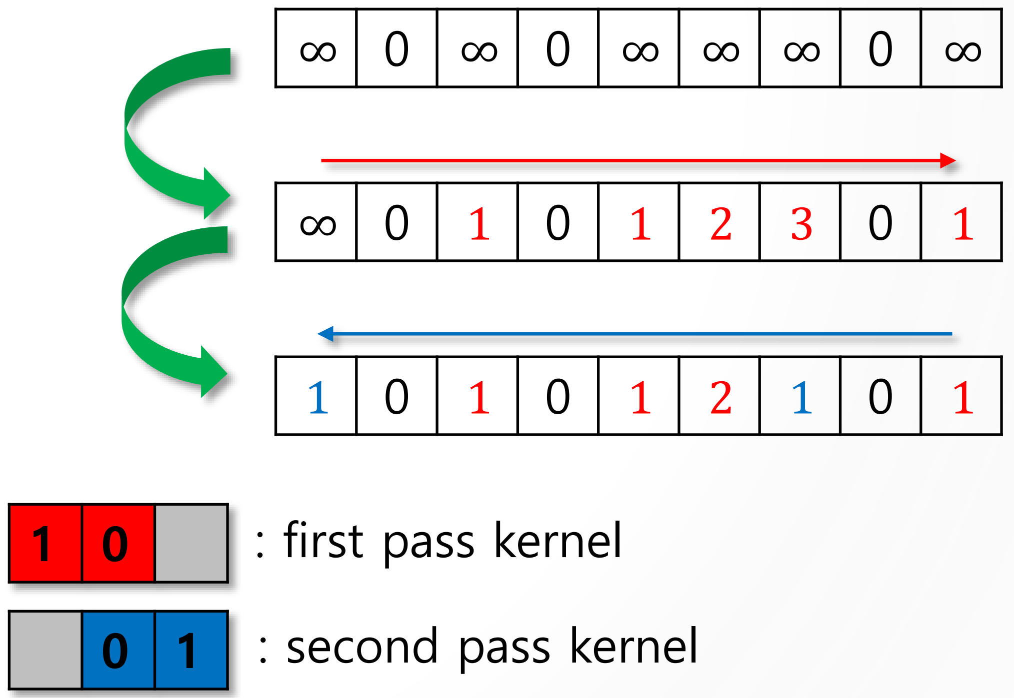
Distance Transform Methods

- Euclidean distance ($L_2 - norm$)
  - $dist(x, y) = sqrt((x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots)$
- City-bock distance ($L_1 - norm$)
  - $dist(x, y) = |x_1 - y_1| + |x_2 - y_2| + \cdots$
- Chessboard distance ($L_\infty - norm$)
  - $dist(x, y) = \max(|x_1 - y_1|, |x_2 - y_2|, \dots)$
- Chamfer distance
  - Approximation version of Euclidean distance.
  - Design-dependent algorithms.
- *Distance propagation*
  - *Narrow band distance transform*

Distance Transform Computation

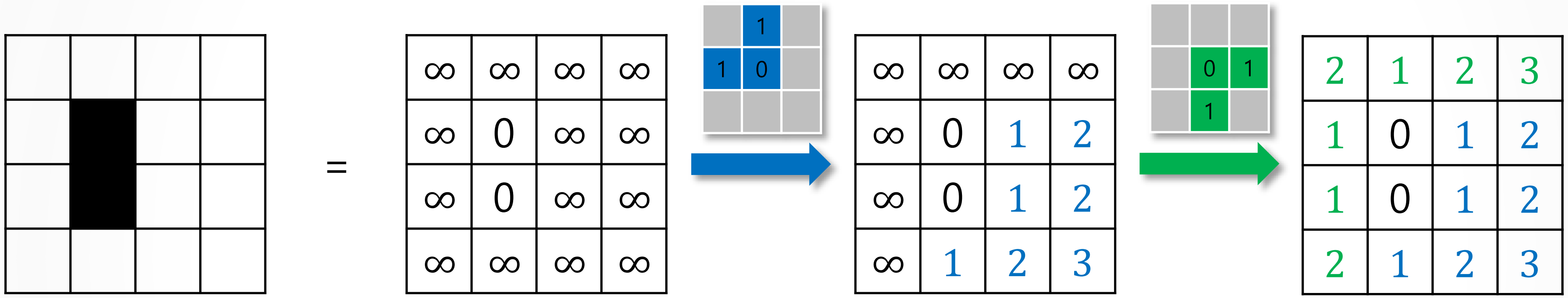
- Naïve approach
  - For each point on the grid, explicitly consider each point of P and minimize.
  - $O(n^2)$ *time complexity.*

- Better methods
  - Simple idea from 1D-case.
  - Two passes :
    - Find closest point on the left
    - Find closest point on the right if closer than one on left
  - Incremental :
    - Moving left-to-right, update distance
    - Analogous for moving right-to-left
  - $O(n)$ *time complexity.*

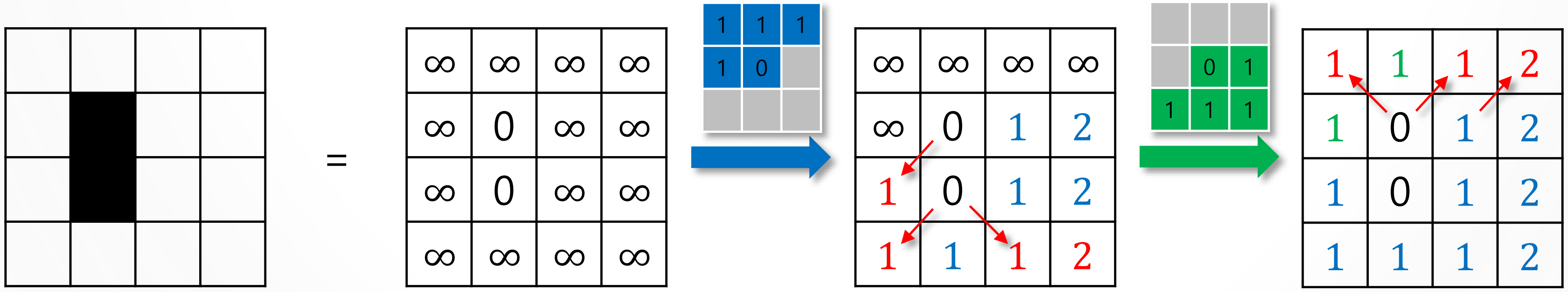


Distance Transform Computation

- City block distance computation ($L_1 - norm$)



- Chessboard distance computation ($L_\infty - norm$)



Distance Transform Computation

- Chamfer distance
  - Approximation version of Euclidean distance.
  - Two pass algorithm.
  - Various windows can be used by Chamfer distance algorithm.

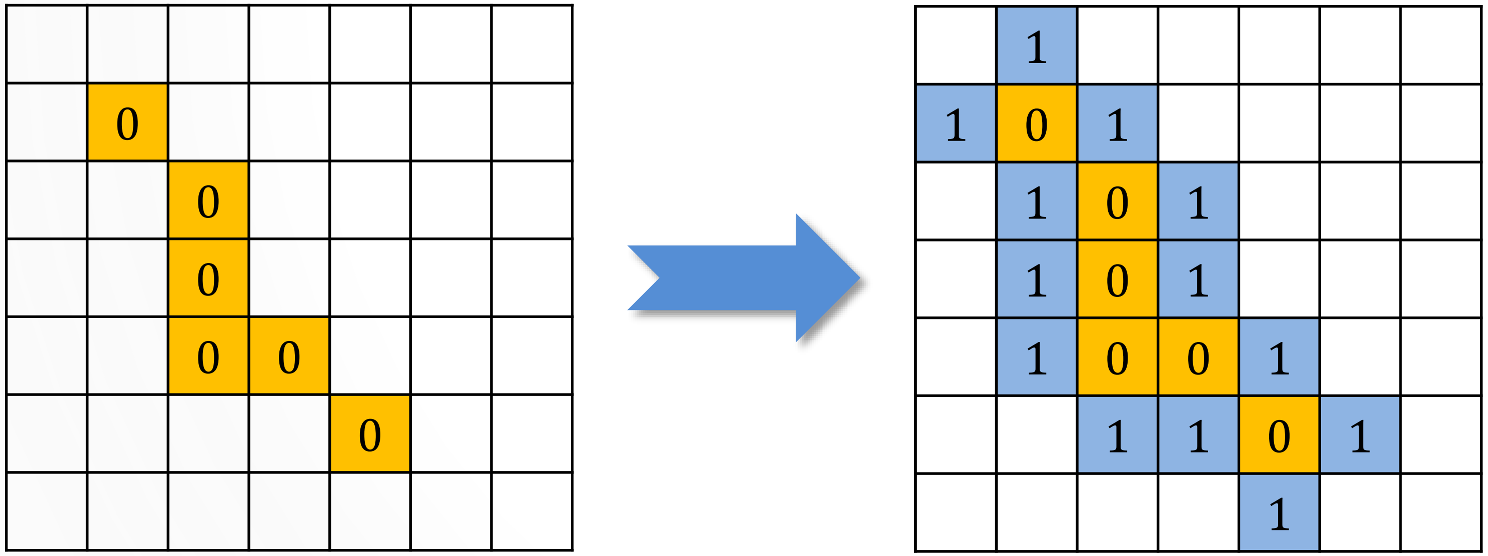
Forward Chamfer-mask

$$f_1(p) = \begin{cases} 0 & p \in surface \\ min\{f_1(q) + 1 : q \in F(p)\} & p \notin surface \end{cases}$$

$$f_2(p) = min\{f_1(p),\ f_2(q) + 1 : q \in B(p)\}$$

Backward Chamfer-mask

- Distance propagation
  - Chamfer distance computation (two pass algorithm) is relatively too expensive for some applications.
  - Propagate distance from edge list
    - Use explicit edge representation



Distance Transform Computation

- Euclidean distance computation ($L_2 - norm$)
  - Simple local propagation methods are not correct.
  - Introduces considerable error, particularly at larger distances.
  - Approximation : *Chamfer distance*

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

?

Euclidean Distance Transformation

- Version 1.
- Version 2.
- Version 3.
- Version 4. (A Linear Time Algorithm)

3D Euclidean Distance Transformation – Version 1.

- Input image : $F = \{f_{ijk}\}$
  - $f_{i,j,k} = \begin{cases} 0, & if\ (i,j,k) \in P(point\ set\ of\ background) \\ \infty, & otherwise \end{cases}$
  - $1 \le i \le W, 1 \le j \le H, 1 \le k \le D$
  - W, H, D is width, height and depth of an input image respectively.

- Output image : $S = \{s_{ijk}\}$
  - $s_{ijk} = min_{i',j',k'}((i-i')^2 + (j-j')^2 + (k-k')^2)$
  - $(i', j', k') \in P$
    - $(f_{i',j',k'} = 0)$
  - $1 \le i \le W, 1 \le j \le H, 1 \le k \le D$
  - The square of the Euclidean distance.

3D Euclidean Distance Transformation – Version 1.

- Decompose 3-dimension into three 1D-transforms

- Transformation 1.
  - $F \rightarrow G = \{g_{ijk}\}$
  - $g_{ijk} = min_{x}\{(i-x)^2;\ f_{xjk} = 0, 1 \leq x \leq W\}$
- Transformation 2.
  - $G \rightarrow H = \{h_{ijk}\}$
  - $h_{ijk} = min_{y}\{g_{iyk} + (j-y)^2; 1 \leq y \leq H\}$
- Transformation 3.
  - $H \rightarrow S = \{s_{ijk}\}$
  - $s_{ijk} = min_{z}\{h_{ijz} + (k-z)^2; 1 \leq z \leq D\}$

3D Euclidean Distance Transformation – Version 1.

- Transformation 1.
  - $F \rightarrow G = \{g_{ijk}\}$
  - $g_{ijk} = min_{x}\{(i-x)^2;\ f_{xjk} = 0, 1 \le x \le W\}$

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

$f_{ijk} = 0$

0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	4	16	25	25	16	9	4	1	0
0	0	0	1	4	9	25	16	9	4	1	0

3D Euclidean Distance Transformation – Version 1.

- Transformation 2.
  - $G \rightarrow H = \{h_{ijk}\}$
  - $h_{ijk} = min_{y}\{g_{iyk} + (j-y)^2; 1 \leq y \leq H\}$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	9	16	25	25	16	9	4	1	0
0	0	0	1	4	9	16	16	9	4	1	0
0	0	0	0	1	4	9	9	4	1	0	0
0	0	0	0	0	1	4	9	9	4	1	0
0	0	0	0	0	0	0	0	0	0	0	0

3D Euclidean Distance Transformation – Version 1.

- Transformation 2.
  - $G \rightarrow H = \{h_{ijk}\}$
  - $h_{ijk} = min_{y}\{g_{iyk} + (j - y)^2; 1 \leq y \leq H\}$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	9	16	25	25	16	9	4	1	0
0	0	0	1	4	9	16	16	9	4	1	0
0	0	0	0	1	4	9	9	4	1	0	0
0	0	0	0	0	1	4	9	9	4	1	0
0	0	0	0	0	0	0	0	0	0	0	0

	0
	9
	9
	0
	25
g_{iyk}	$\boxed{9}$
	4
	1
	0

G

3D Euclidean Distance Transformation – Version 1.

- Transformation 2.
  - $G \rightarrow H = \{h_{ijk}\}$
  - $h_{ijk} = min_{\color{red}{y}}\{g_{i\color{red}{y}k} + (j-y)^2; 1 \leq y \leq H\}$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	9	16	25	25	16	9	4	1	0
0	0	0	1	4	9	16	16	9	4	1	0
0	0	0	0	1	4	9	9	4	1	0	0
0	0	0	0	0	1	4	9	9	4	1	0
0	0	0	0	0	0	0	0	0	0	0	0

	G	+	$(j-y)^2$	=	
	0		25		25
	9		16		25
	9		9		18
	0		4		**4**
	25		1		26
$g_{i\color{red}{y}k}$	$\boxed{9}$		0		9
	4		1		5
	1		4		5
	0		9		9

3D Euclidean Distance Transformation – Version 1.

- Transformation 2.
  - $G \rightarrow H = \{h_{ijk}\}$
  - $h_{ijk} = min_y\{g_{iyk} + (j-y)^2; 1 \leq y \leq H\}$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	9	16	25	25	16	9	4	1	0
0	0	0	1	4	9	16	16	9	4	1	0
0	0	0	0	1	4	9	9	4	1	0	0
0	0	0	0	0	1	4	9	9	4	1	0
0	0	0	0	0	0	0	0	0	0	0	0

	G	+	$(j-y)^2$	=			
	0		25		25		
	9		16		25		
	9		9		18		
	0		4		**4**		
	25		1		26		
g_{iyk}	9		0		9	h_{iyk}	**4**
	4		1		5		
	1		4		5		
	0		9		9	*min*	

3D Euclidean Distance Transformation – Version 1.

- Transformation 3.
  - $H \rightarrow S = \{s_{ijk}\}$
  - $s_{ijk} = min_{z}\{h_{ijz} + (k-z)^2; 1 \leq z \leq D\}$
  - Same as Transformation 2, only for z-direction
- The image $S = \{s_{ijk}\}$ is the square Euclidean distance transform of an image $F = \{f_{ijk}\}$
- Proof:

$$g_{ijk} = min_{x}\{(i-x)^2;\ f_{xjk} = 0, 1 \leq x \leq W\}$$

$$h_{ijk} = min_{y}\{g_{iyk} + (j-y)^2; 1 \leq y \leq H\}$$

$$s_{ijk} = min_{z}\{h_{ijz} + (k-z)^2; 1 \leq z \leq D\}$$

3D Euclidean Distance Transformation – Version 1.

$$
\begin{aligned}
h_{ijk} &= min_{\mathbf{y}}\{g_{iyk} + (j-y)^2; 1 \le y \le H\} \\
&= min_{\mathbf{y}}\{min_{\mathbf{x}}\{(i-x)^2;\ f_{xyk} = 0, 1 \le x \le W\} + (j-y)^2; 1 \le y \le H\} \\
&= min_{\mathbf{y}}\{min_{\mathbf{x}}\{(i-x)^2 + (j-y)^2;\ f_{xyk} = 0, 1 \le x \le W\}; 1 \le y \le H\} \\
&= min_{(\mathbf{x},\mathbf{y})}\{(i-x)^2 + (j-y)^2; f_{xyk} = 0, 1 \le x \le W, 1 \le y \le H\}
\end{aligned}
$$

$$
\begin{aligned}
s_{ijk} &= min_{\mathbf{z}}\{h_{ij\mathbf{z}} + (k-z)^2; 1 \le z \le D\} \\
&= min_{\mathbf{z}}\{min_{(\mathbf{x},\mathbf{y})}\{(i-x)^2 + (j-y)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H\} + (k-z)^2; 1 \le z \le D\} \\
&= min_{\mathbf{z}}\{min_{(\mathbf{x},\mathbf{y})}\{(i-x)^2 + (j-y)^2 + (k-z)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H\}; 1 \le z \le D\} \\
&= min_{(\mathbf{x},\mathbf{y},\mathbf{z})}\{(i-x)^2 + (j-y)^2 + (k-z)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H, 1 \le z \le D\}
\end{aligned}
$$

3D Euclidean Distance Transformation – Version 1.

$$
\begin{aligned}
h_{ijk} &= min_{\boldsymbol{y}}\left\{g_{iyk} + (j-y)^2; 1 \le y \le H\right\} \\
&= min_{\boldsymbol{y}}\left\{min_{\boldsymbol{x}}\left\{(i-x)^2;\ f_{xyk} = 0, 1 \le x \le W\right\} + (j-y)^2; 1 \le y \le H\right\} \\
&= min_{\boldsymbol{y}}\left\{min_{\boldsymbol{x}}\left\{(i-x)^2 + (j-y)^2;\ f_{xyk} = 0, 1 \le x \le W\right\}; 1 \le y \le H\right\} \\
&= min_{(\boldsymbol{x},\boldsymbol{y})}\left\{(i-x)^2 + (j-y)^2; f_{xyk} = 0, 1 \le x \le W, 1 \le y \le H\right\}
\end{aligned}
$$

$$
\begin{aligned}
s_{ijk} &= min_{\boldsymbol{z}}\left\{h_{ij\boldsymbol{z}} + (k-z)^2; 1 \le z \le D\right\} \\
&= min_{\boldsymbol{z}}\left\{min_{(\boldsymbol{x},\boldsymbol{y})}\left\{(i-x)^2 + (j-y)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H\right\} + (k-z)^2; 1 \le z \le D\right\} \\
&= min_{\boldsymbol{z}}\left\{min_{(\boldsymbol{x},\boldsymbol{y})}\left\{(i-x)^2 + (j-y)^2 + (k-z)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H\right\}; 1 \le z \le D\right\} \\
&\boxed{= min_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})}\left\{(i-x)^2 + (j-y)^2 + (k-z)^2; f_{xyz} = 0, 1 \le x \le W, 1 \le y \le H, 1 \le z \le D\right\}}
\end{aligned}
$$

- $S = \{s_{ijk}\}$ is the square Euclidean distance image of F

3D Euclidean Distance Transformation – Version 1.

- Cuboid voxel ?
  - Pixel spacing(α) & slice spacing(β) in medical CT images.
  - $g_{ijk} = min_x\{(\alpha(i - x))^2;\ f_{xjk} = 0, 1 \le x \le W\}$
  - $h_{ijk} = min_y\{g_{iyk} + (\alpha(j - y))^2; 1 \le y \le H\}$
  - $s_{ijk} = min_z\{h_{ijz} + (\beta(k - z))^2; 1 \le z \le D\}$

- Transformation 1.
  - Input : $F = \{f_{ijk}\}$
  - Output : $G = \{g_{ijk}\}$
  - **Step 1.1**
    - $Initialize\ G = \{g_{ijk}\}$
      - $g_{ijk} = \begin{cases} \infty, if\ f_{ijk} \neq 0 \\ 0,\ if\ f_{ijk} = 0 \end{cases}$
    - $g_{ijk} \leftarrow \left(\sqrt{g_{(i-1)jk}} + 1\right)^2, if\ f_{ijk} \neq 0,$
    - $g_{ijk} \leftarrow 0, otherwise$
      - $for\ i = 2, 3, \dots, W.$
  - **Step 1.2**
    - $g_{ijk} \leftarrow min\left\{\left(\sqrt{g_{(i+1)jk}} + 1\right)^2, g_{ijk}\right\}$
      - $for\ i = W-1, W-2, \dots, 1.$

$f_{ijk} = 0$

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

0	1	4	9	16	0	1
0	1	4	9	16	25	36
0	0	0	1	4	9	16

0	1	4	4	1	0	1
0	1	4	9	16	25	36
0	0	0	1	4	9	16

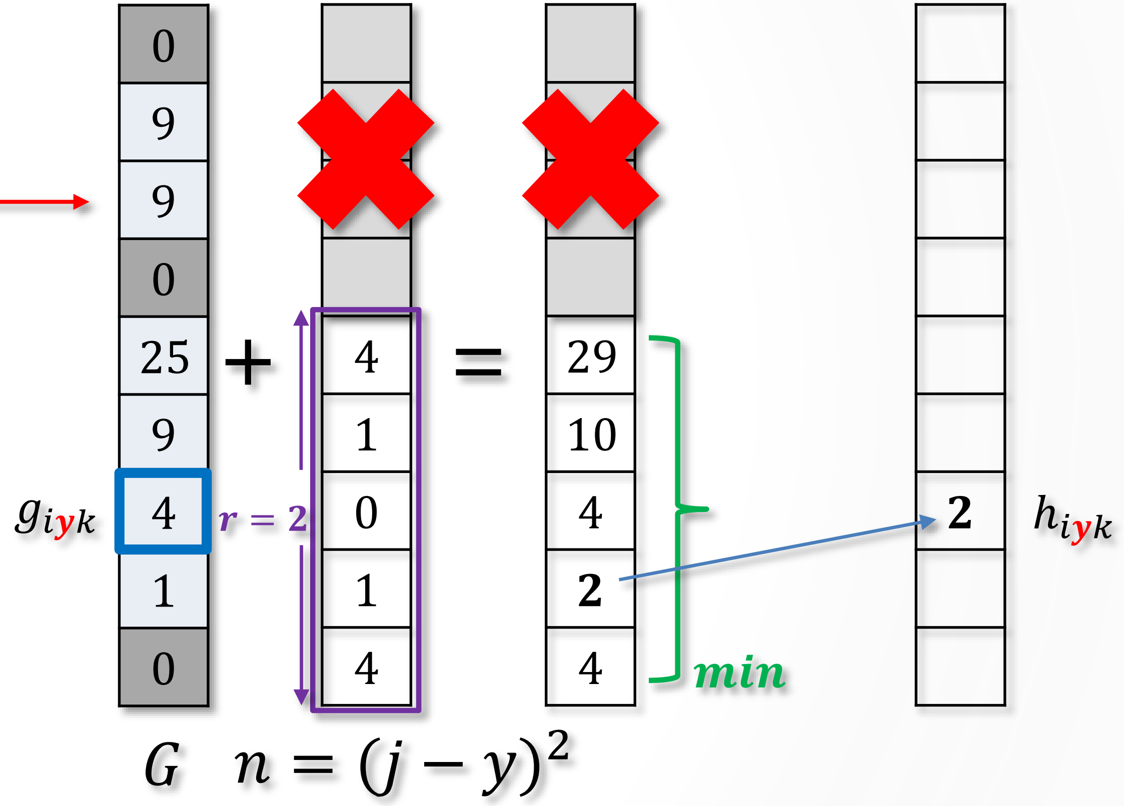
3D Euclidean Distance Transformation – Version 2.

- Transformation 2.
  - Input : $G = \{g_{ijk}\}, output\ of\ Transformation\ 1.$
  - Output : $H = \{h_{ijk}\}$
  - $h_{ijk} \leftarrow min_{-r\leq n\leq r}\{g_{i(j+n)k} + n^2\}, where\ r = \sqrt{g_{ijk}}$
    - $for\ 1 \leq j + n \leq H$
- Transformation 3.
  - Input : $H = \{h_{ijk}\}, output\ of\ Transformation\ 2.$
  - Output : $S = \{s_{ijk}\}$
  - $s_{ijk} \leftarrow min_{-r\leq n\leq r}\{h_{ij(k+n)} + n^2\}, where\ r = \sqrt{h_{ijk}}$
    - $for\ 1 \leq k + n \leq D$

- Bounds of the searching interval $\pm r$, there is at least one $0 - voxel$.
- Limiting search areas for minimization. (*more limitation in ver.2*)

- **Limited search area in ver.1**

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	0	0	1	4	9	16	9	4	1	0	0
0	1	4	4	1	0	1	4	4	1	0	0
0	1	4	9	16	25	25	16	9	4	1	0
0	0	0	1	4	9	16	16	9	4	1	0
0	0	0	0	1	4	9	9	4	1	0	0
0	0	0	0	0	1	4	9	9	4	1	0
0	0	0	0	0	0	0	0	0	0	0	0



3D Euclidean Distance Transformation – Version 3.

- Transformation 1. - Same as ver.1
- Transformation 2.
  - Input : $G = \{g_{ijk}\}$, Output : $H = \{h_{ijk}\}$
  - Step 2.1

$for\ j = [2..H]$
$\{$
$\quad if\ (\ g_{ijk} > g_{i(j-1)k} + 1\ )$
$\quad \{$
$\quad\quad for\ n = [0..\frac{g_{ijk} - g_{i(j-1)k} - 1}{2}]$ *// w/ bounding check.*
$\quad\quad\quad if\ (\ g_{i(j-1)k} + (n+1)^2 \geq g_{i(j+n)k}\ )$
$\quad\quad\quad\quad break;$
$\quad\quad\quad else$
$\quad\quad\quad\quad h'_{i(j+n)k} \leftarrow g_{i(j-1)k} + (n+1)^2$
$\quad \}$
$\quad else$
$\quad\quad h'_{ijk} \leftarrow g_{ijk}$
$\}$

Parabola lower envelope computation.

3D Euclidean Distance Transformation – Version 3.

- Transformation 1. - Same as ver.1
- Transformation 2.
  - Input : $G = \{g_{ijk}\}$, Output : $H = \{h_{ijk}\}$
  - Step 2.1

$for\ j = [2..H]$

$\{$

$\quad if\ (\ g_{ijk} > g_{i(j-1)k} + 1\ )$

$\quad \{$

$\quad\quad for\ n = [0..\frac{g_{ijk} - g_{i(j-1)k} - 1}{2}]$ // w/ bounding check.

$\quad\quad\quad if\ (\ g_{i(j-1)k} + (n+1)^2 \geq g_{i(j+n)k}\ )$

$\quad\quad\quad\quad break;$

$\quad\quad\quad else$

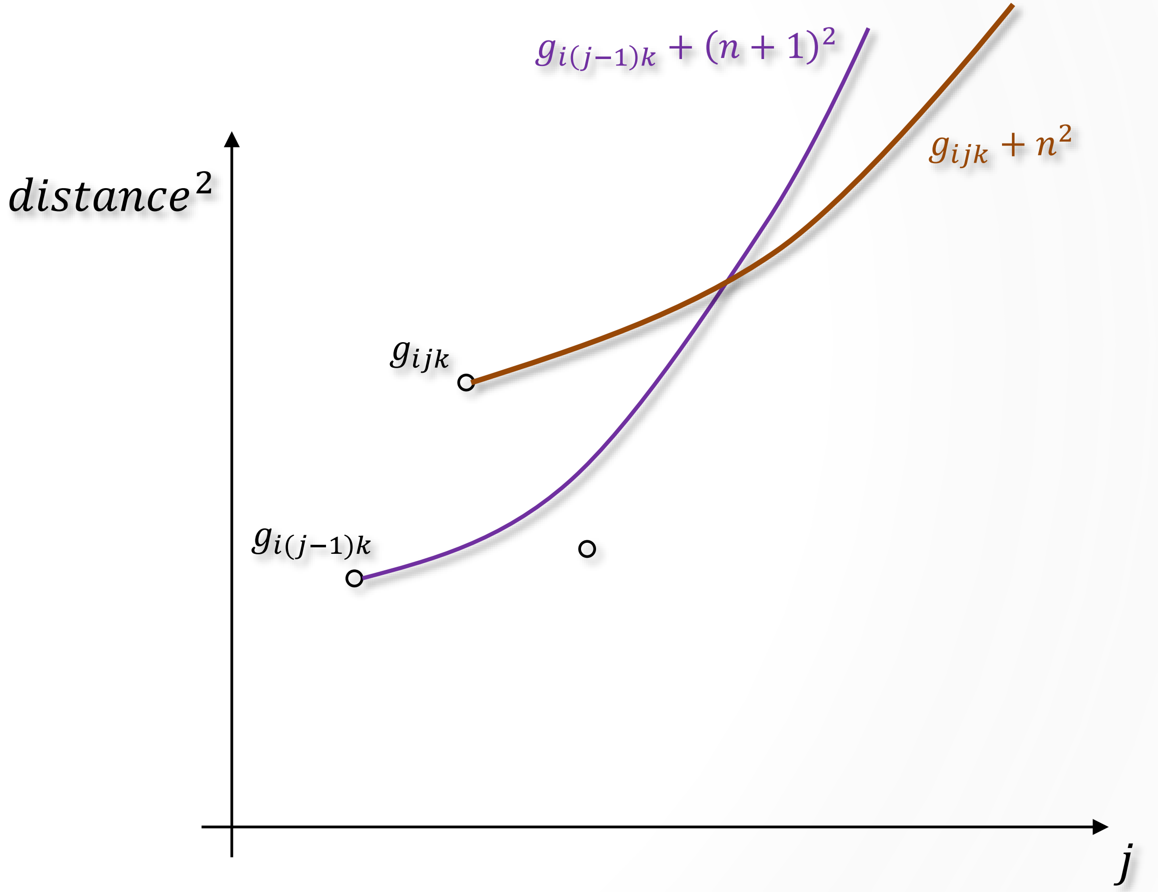
$\quad\quad\quad\quad h'_{i(j+n)k} \leftarrow g_{i(j-1)k} + (n+1)^2$

$\quad \}$

$\quad else$

$\quad\quad h'_{ijk} \leftarrow g_{ijk}$

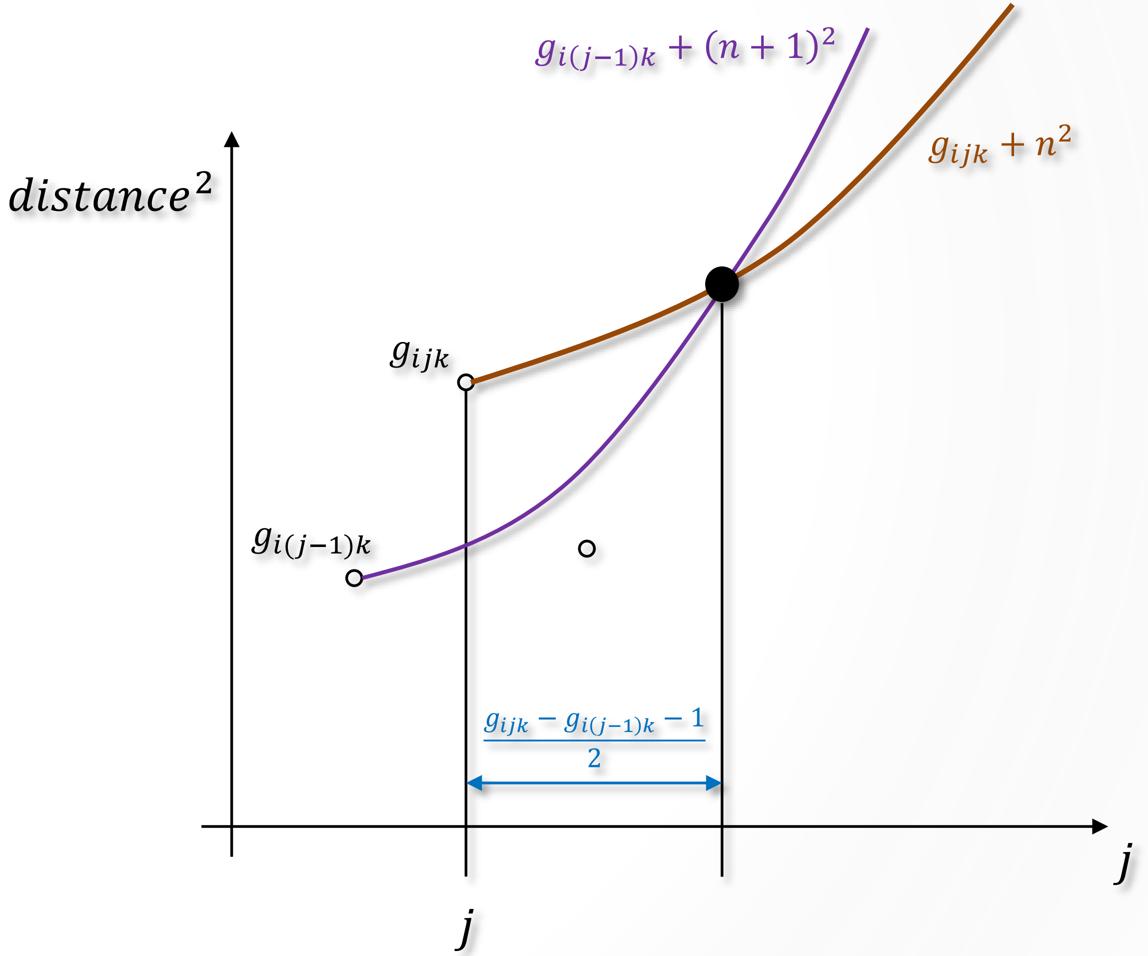
$\}$



3D Euclidean Distance Transformation – Version 3.

- Transformation 1. - Same as ver.1
- Transformation 2.
  - Input : $G = \{g_{ijk}\}$, Output : $H = \{h_{ijk}\}$
  - Step 2.1

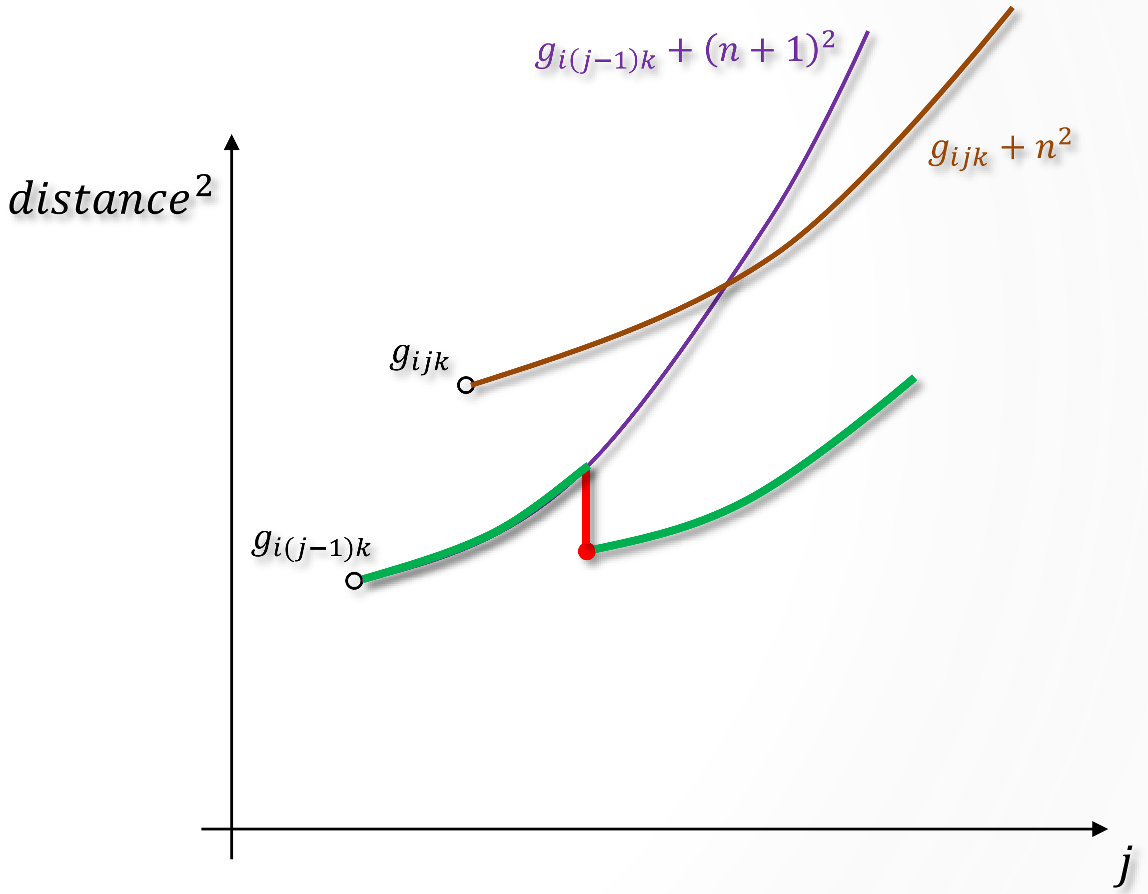
$$
\begin{aligned}
& for\ j = [2..H] \\
& \{ \\
& \quad if\ (\ g_{ijk} > g_{i(j-1)k} + 1\ ) \\
& \quad \{ \\
& \qquad for\ n = [0..\tfrac{g_{ijk}-g_{i(j-1)k}-1}{2}]\ \text{// w/ bounding check.} \\
& \qquad \quad if\ (\ g_{i(j-1)k} + (n+1)^2 \geq g_{i(j+n)k}\ ) \\
& \qquad \qquad break; \\
& \qquad \quad else \\
& \qquad \qquad h'_{i(j+n)k} \leftarrow g_{i(j-1)k} + (n+1)^2 \\
& \quad \} \\
& \quad else \\
& \qquad h'_{ijk} \leftarrow g_{ijk} \\
& \}
\end{aligned}
$$



3D Euclidean Distance Transformation – Version 3.

- Transformation 1. - Same as ver.1
- Transformation 2.
  - Input : $G = \{g_{ijk}\}$, Output : $H = \{h_{ijk}\}$
  - Step 2.1

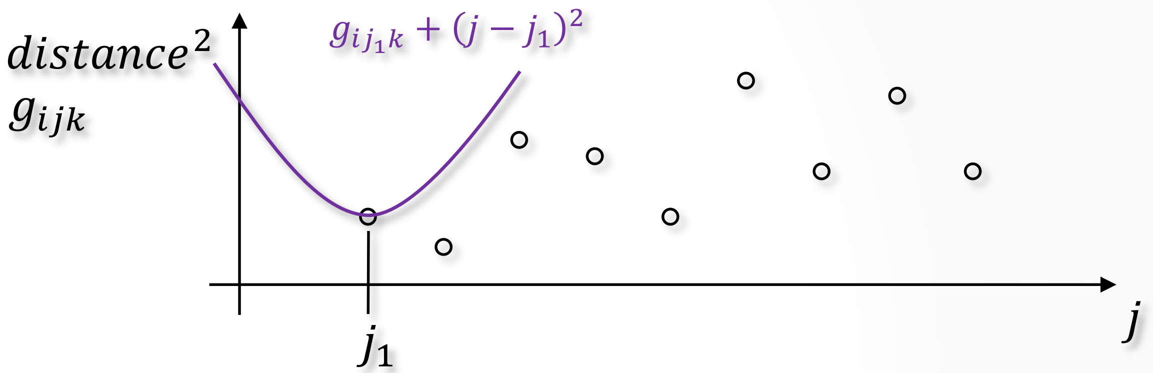
$$
\begin{array}{l}
for\ j = [2..H] \\
\{ \\
\quad if\ (\ g_{ijk} > g_{i(j-1)k} + 1\ ) \\
\quad \{ \\
\quad\quad for\ n = [0..\frac{g_{ijk} - g_{i(j-1)k} - 1}{2}]\ \text{// w/ bounding check.} \\
\quad\quad\quad if\ (\ g_{i(j-1)k} + (n+1)^2 \geq g_{i(j+n)k}\ ) \\
\quad\quad\quad\quad break; \\
\quad\quad\quad else \\
\quad\quad\quad\quad \boldsymbol{h'_{i(j+n)k} \leftarrow g_{i(j-1)k} + (n+1)^2} \\
\quad \} \\
\quad else \\
\quad\quad h'_{ijk} \leftarrow g_{ijk} \\
\}
\end{array}
$$

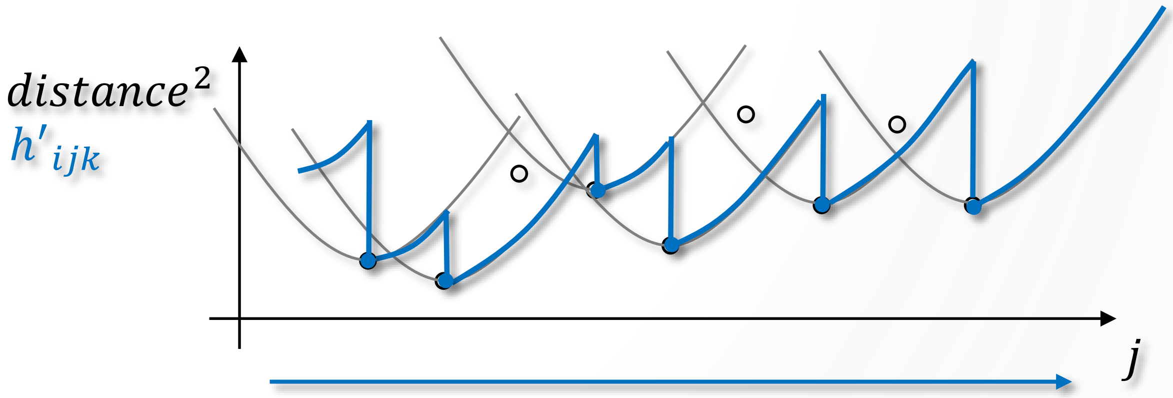


3D Euclidean Distance Transformation – Version 3.

- Transformation 1. - Same as ver.1
- Transformation 2.
  - Input : $G = \{g_{ijk}\}$, Output : $H = \{h_{ijk}\}$
  - Step 2.1 (forward scan)

$for\ j \leftarrow 2\ to\ H$
$\{$
$\quad if\ (\ g_{ijk} > g_{i(j-1)k} + 1\ )$
$\quad \{$
$\quad\quad for\ n = [0..\frac{g_{ijk} - g_{i(j-1)k} - 1}{2}]$ // w/ bounding check.
$\quad\quad\quad if\ (\ g_{i(j-1)k} + (n+1)^2 \geq g_{i(j+n)k}\ )$
$\quad\quad\quad\quad break;$
$\quad\quad\quad else$
$\quad\quad\quad\quad h'_{i(j+n)k} \leftarrow g_{i(j-1)k} + (n+1)^2$
$\quad \}$
$\quad else$
$\quad\quad h'_{ijk} \leftarrow g_{ijk}$
$\}$

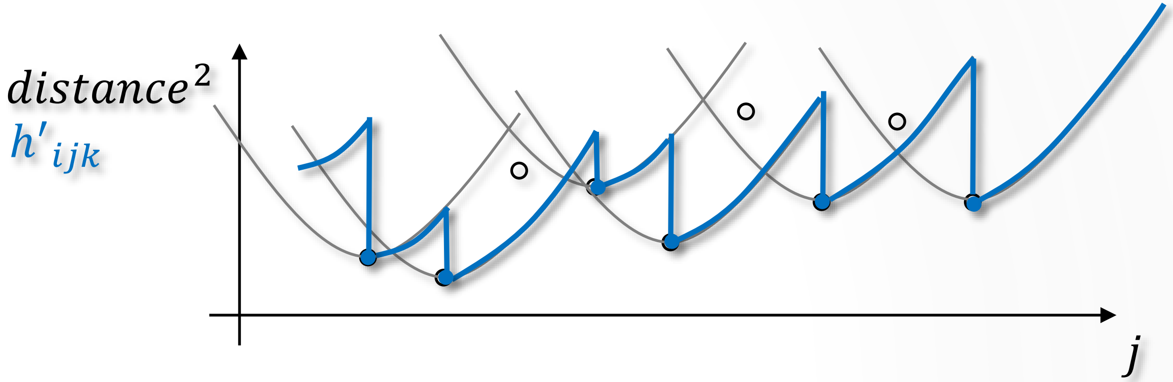


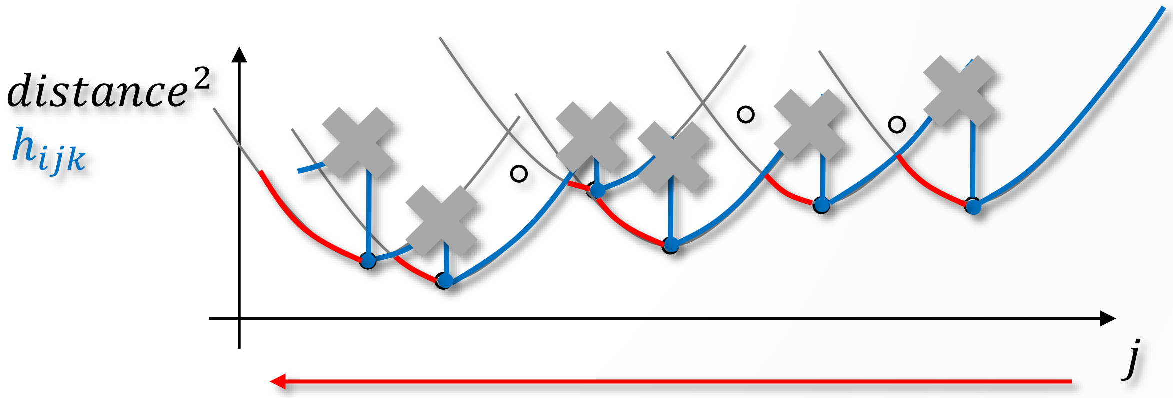


3D Euclidean Distance Transformation – Version 3.

– Step 2.2 (backward scan)

$for\ j \leftarrow H - 1\ to\ 1$
$\{$
$\quad if\ (\ h'_{ijk} > h'_{i(j+1)k} + 1\ )$
$\quad \{$
$\quad\quad for\ \ n = [0..\frac{(h'_{ijk} - h'_{i(j+1)k} - 1)}{2}]$ *// w/ bounding check*
$\quad\quad\quad if\ (\ h'_{i(j+1)k} + (n+1)^2 > h'_{i(j-n)k}\ )$
$\quad\quad\quad\quad break;$
$\quad\quad\quad else$
$\quad\quad\quad\quad h_{i(j-n)k} \leftarrow h'_{i(j+1)k} + (n+1)^2$
$\quad \}$
$\quad else$
$\quad\quad h_{ijk} \leftarrow h'_{ijk}$
$\}$





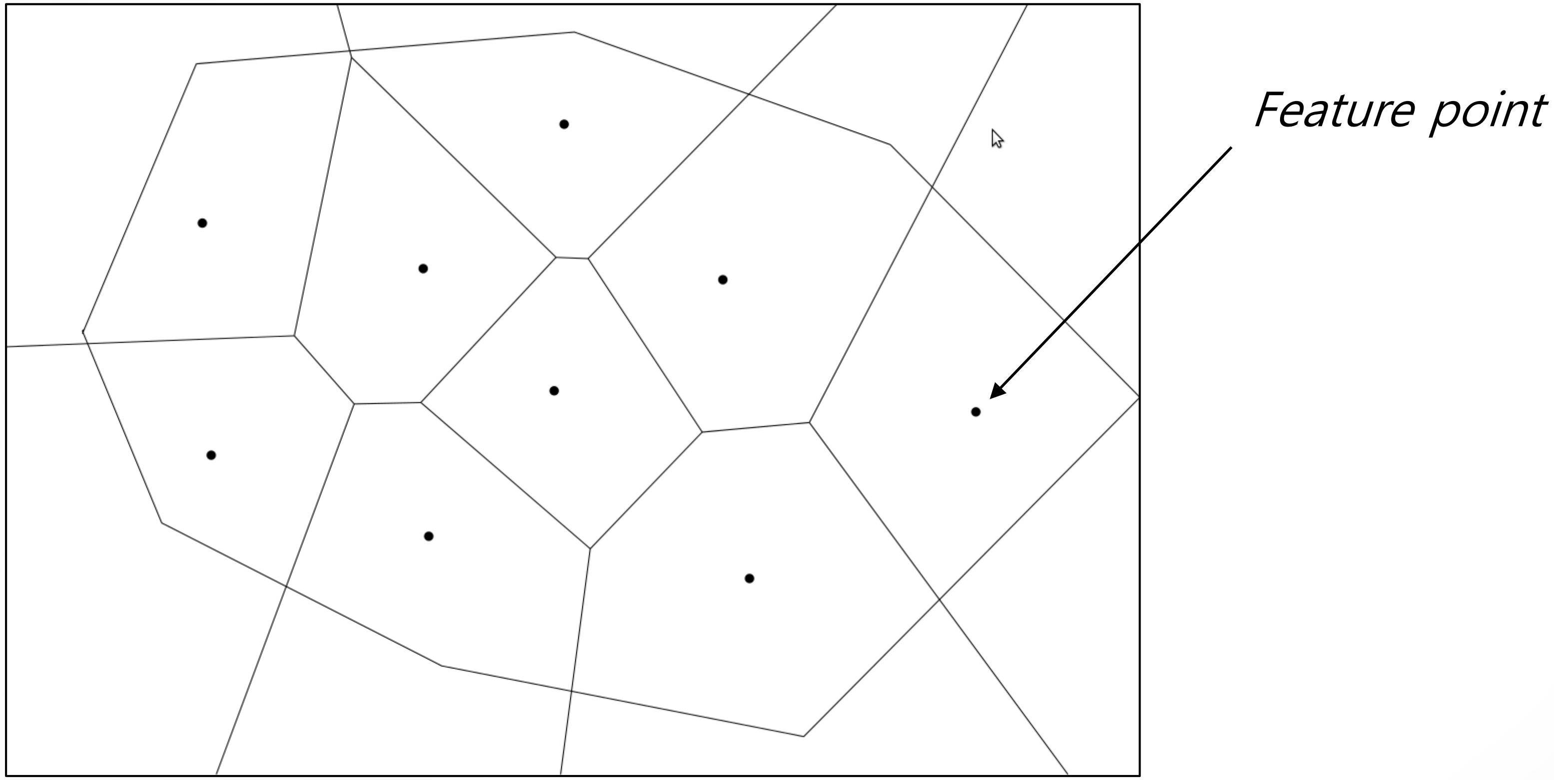
3D Euclidean Distance Transformation – Version 3.

- Transformation 3.
  - Input : $\mathrm{H} = \{h_{ijk}\}$, Output : $\mathrm{S} = \{s_{ijk}\}$
  - Same as Transformation 3. for z-direction.

- Performance evaluations
  - Version 1.
  - Version 2.
  - Version 3.

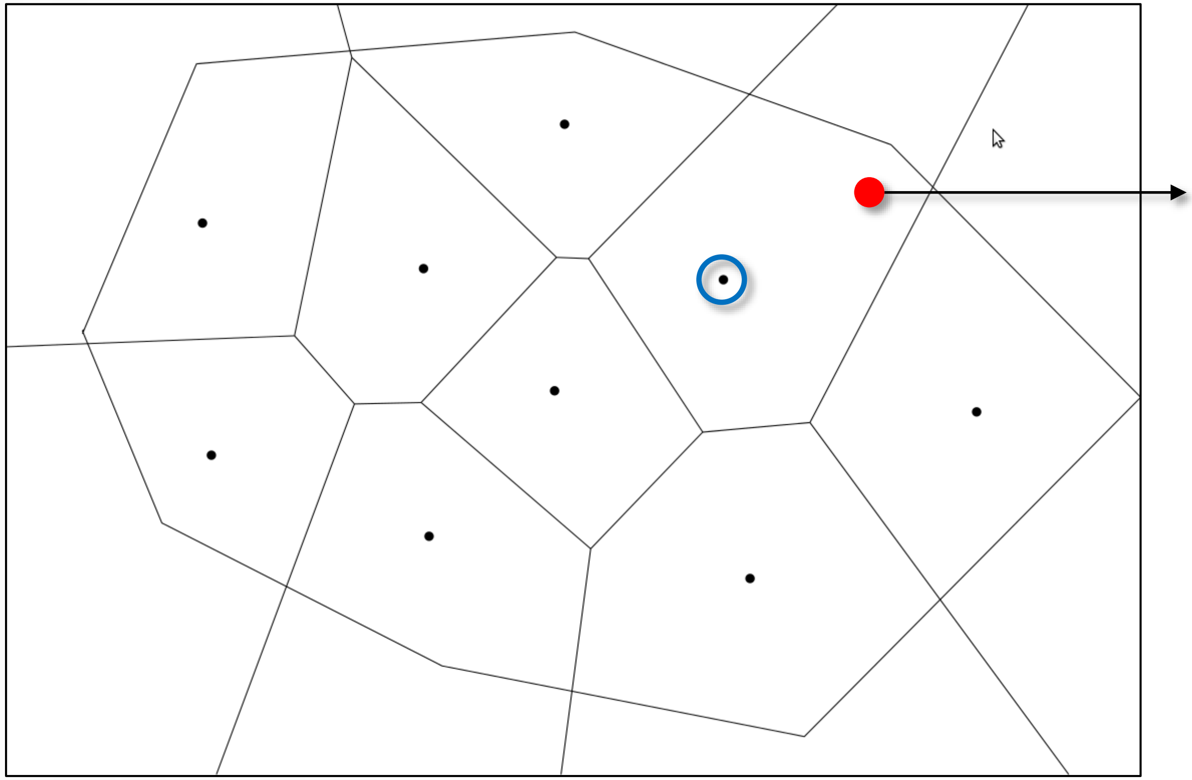
3D Euclidean Distance Transformation – Version 4.

- A linear time algorithm for computing exact Euclidean distance transform.
  - Idea : *Voronoi diagram, dimensionality reduction*
- Direct calculation of Euclidean distance via Voronoi diagram.



3D Euclidean Distance Transformation – Version 4.

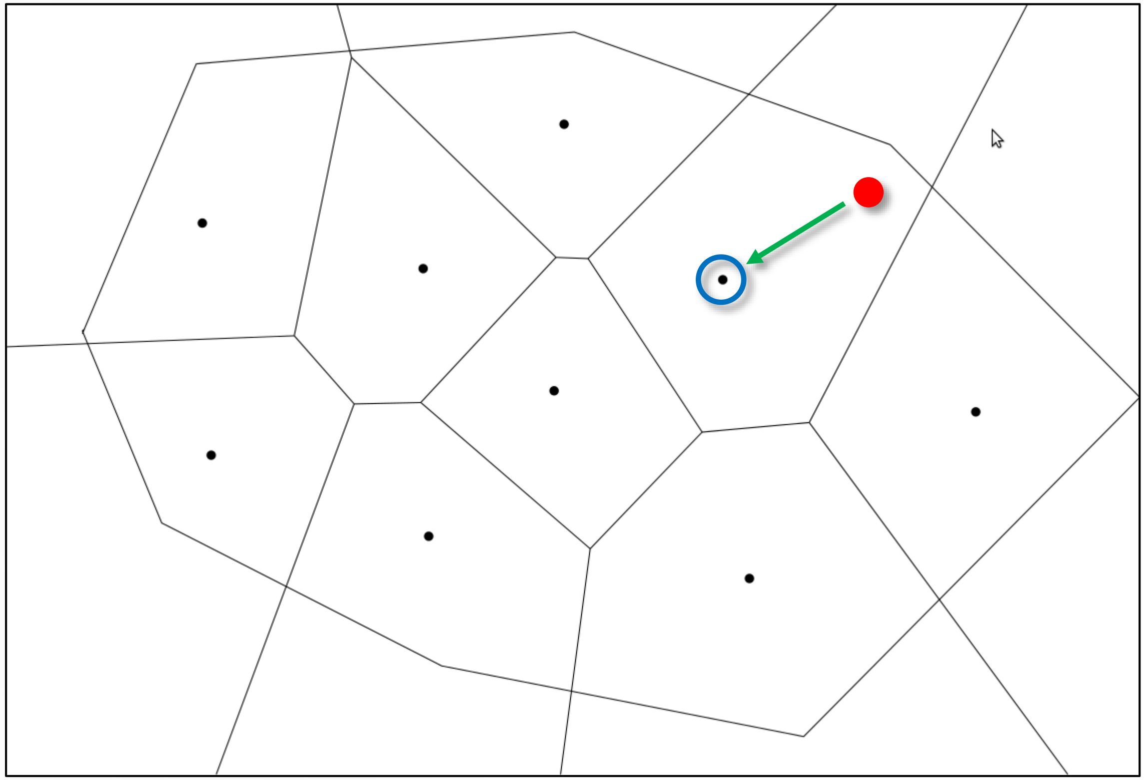
- A linear time algorithm for computing exact Euclidean distance transform.
  - Idea : *Voronoi diagram, dimensionality reduction*
- Direct calculation of Euclidean distance via Voronoi diagram.



Euclidean distance to closest feature point ?

3D Euclidean Distance Transformation – Version 4.

- A linear time algorithm for computing exact Euclidean distance transform.
  - Idea : *Voronoi diagram, dimensionality reduction*
- Direct calculation of Euclidean distance via Voronoi diagram.

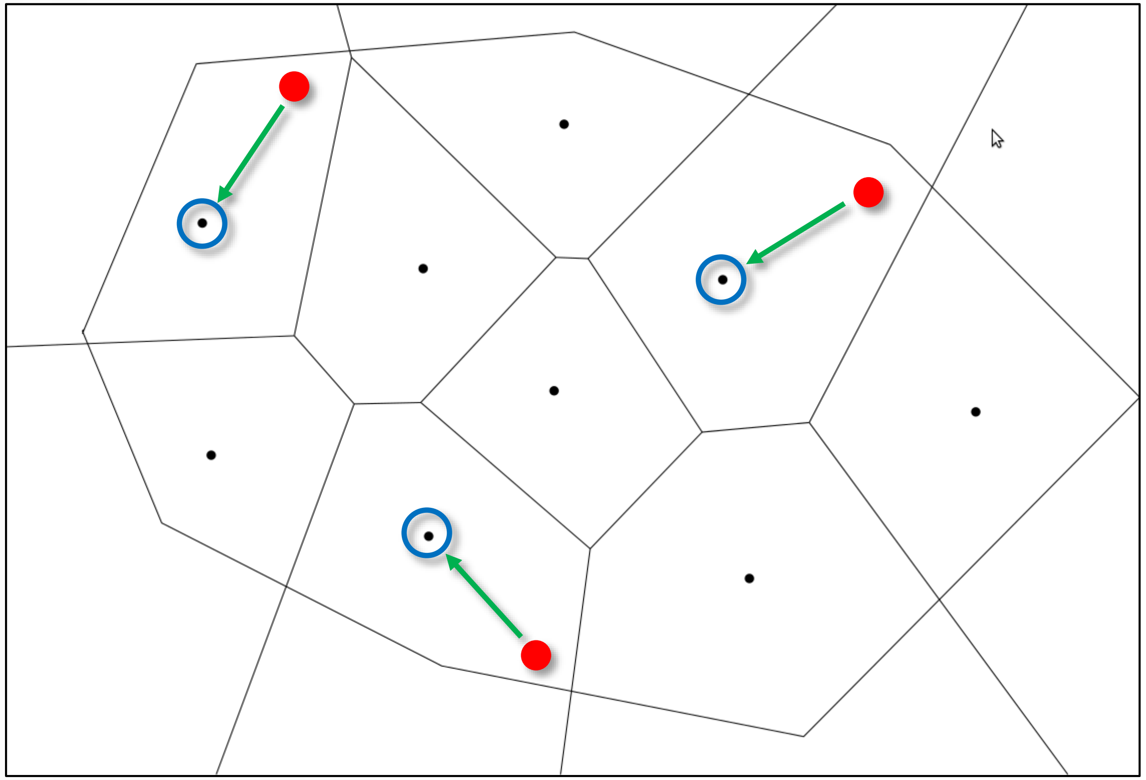


Euclidean distance to closest feature point ?

= Feature Transform (FT)

3D Euclidean Distance Transformation – Version 4.

- A linear time algorithm for computing exact Euclidean distance transform.
  - Idea : *Voronoi diagram, dimensionality reduction*
- Direct calculation of Euclidean distance via Voronoi diagram.



Goal :
for every voxel, calculate feature transform.

A linear time algorithm for computing exact Euclidean distance transform

- Definitions & abbreviations
  - Feature Voxel (FV)
  - Distance Transform (DT)
  - Closest Feature Voxel (CFV)
  - Closest Feature Transform (FT)
  - Euclidean DT (EDT)

  - Voronoi sites : $S = \{f_i\}$
    - Feature points (f)
  - Voronoi cell : C_f
    - The set of all points whose closest point is f
    - f is <u>center</u> of C_f (feature point)
  - Voronoi cells : $V_s = \{C_{f_i}\}$

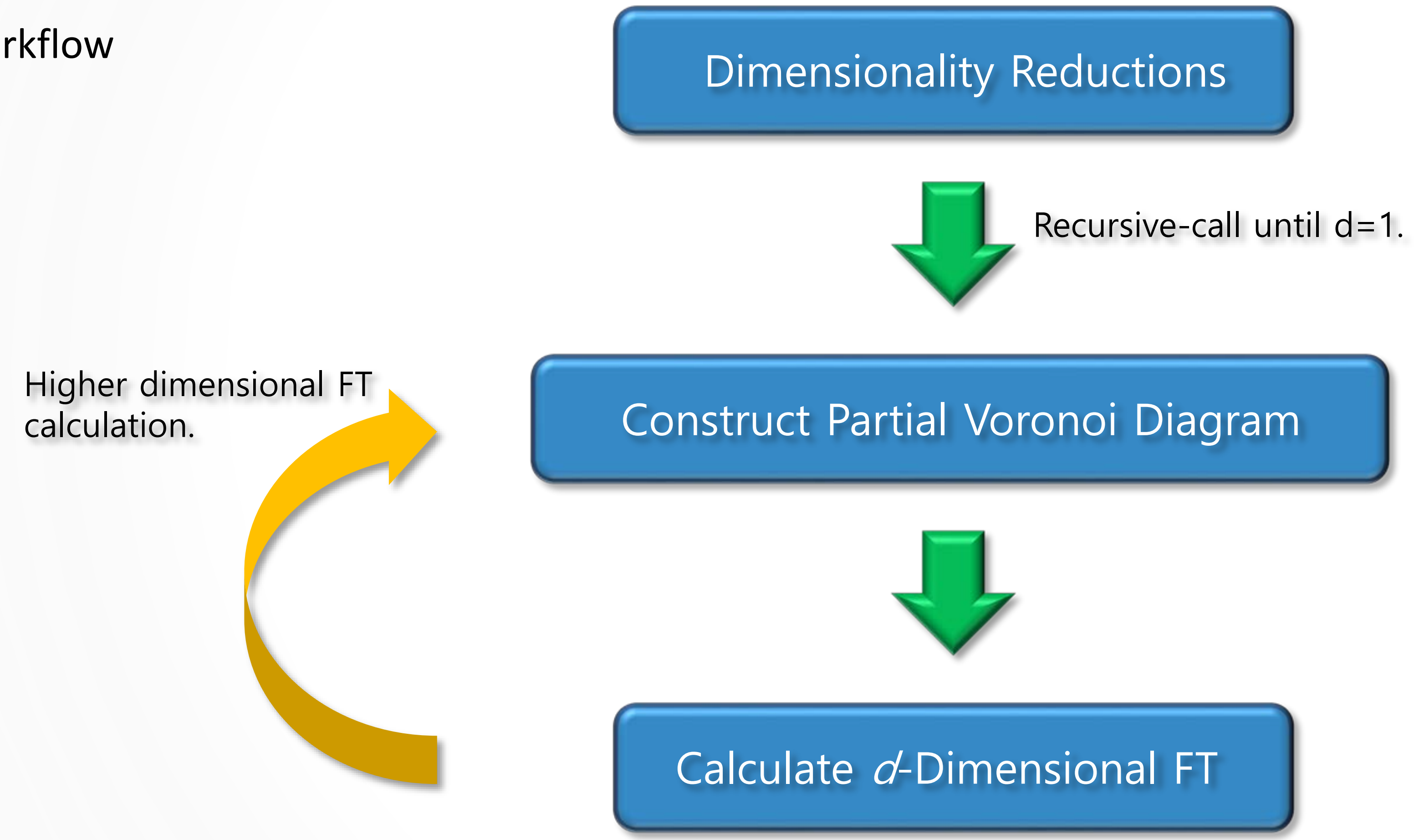
- The FT of a binary image :
  - discretized version of the voronoi diagram where Voronoi sites are the FVs of the image
- For each voxel x in I, $F(x) = CFV\ in\ I$.
  - DT can be computed easily from F.
- $I_{d,x}$: d-dimensional subimage.
  - restriction of I to the subspace whose last $k - d$ coordinates are identical to the corresponding coordinates of x. (k : dimension of input image)
- F_d : FT at the dth dimension level.
  - $F_d(x) = CFV\ in\ I_{d,x}$
- $I_{k,x} = I$ and $F_k = F$.
- $F_0(x) = x\ if\ I(x) = 1,$
- $F_0(x) = \emptyset\ otherwise.$
  - $\emptyset$: undefined.

A linear time algorithm for computing exact Euclidean distance transform

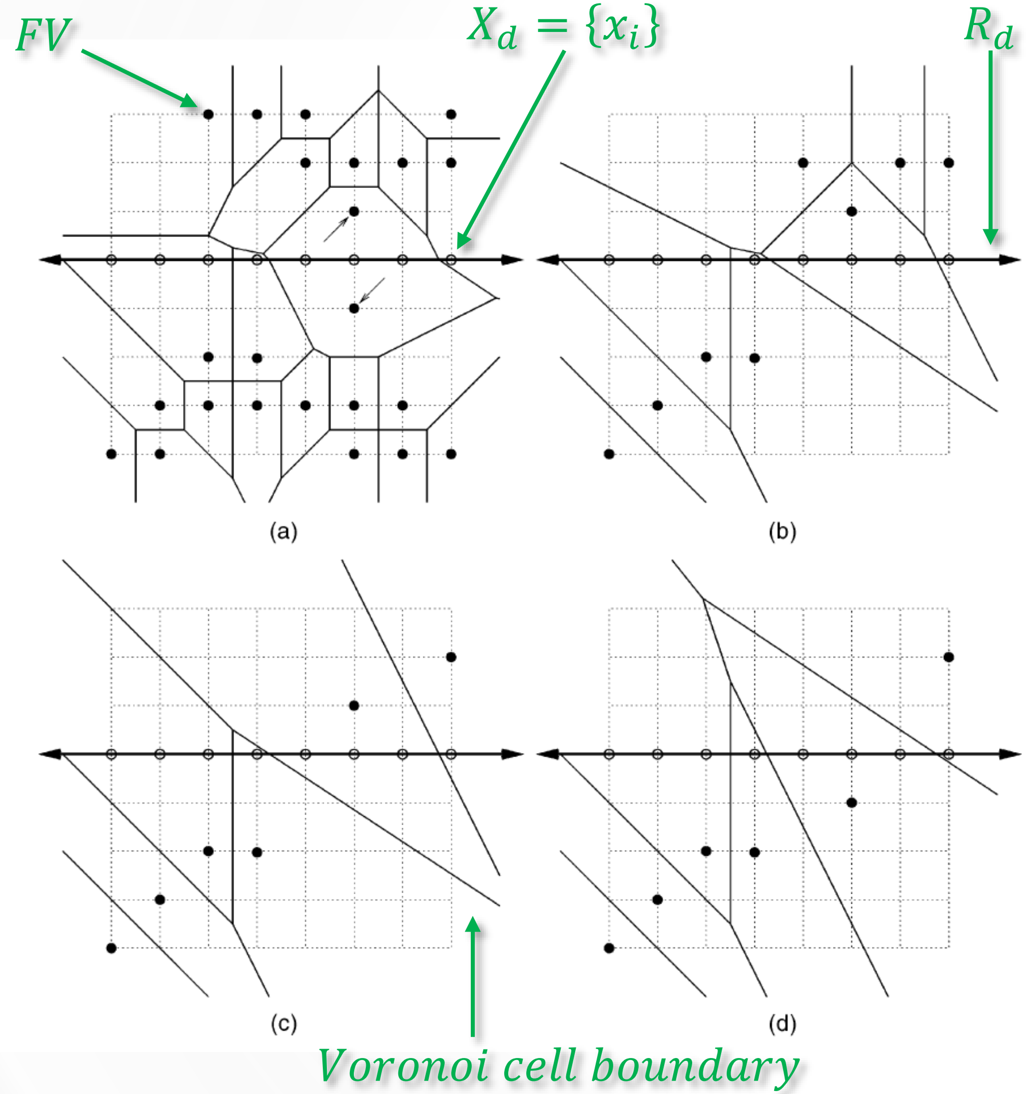
- $X_d = \{x_i\}\ for\ i = 1, \ldots, n_d$
  - Set of n_d voxels in I formed by varying the dth coordinate from 1 to n_d and fixing all other coordinates.
- R_d
  - "row" (continuous line) running through the set of voxels X_d.
- S_d
  - Set of FVs in the binary subimage $I_{d,x}$.
  - All voxels x_i in the set X_d belong to the same subimage.
- $V^*{}_d = V_{S_d} \cap R_d$
  - Intersection of the Voronoi diagram V_{S_d} whose Voronoi sites are the set of FVs S_d with the row R_d.
- $S'{}_d = \{F_{d-1}(x_i)\}$
  - Set of CFVs in the next lower dimension for the set of voxels $X_d = \{x_i\}$ on the row R_d.
  - $S'{}_d$ has at most one FV for each voxel x_i. (arbitrarily chosen if equidistant)
  - $S'{}_d \subseteq S_d$

A linear time algorithm for computing exact Euclidean distance transform

- Workflow

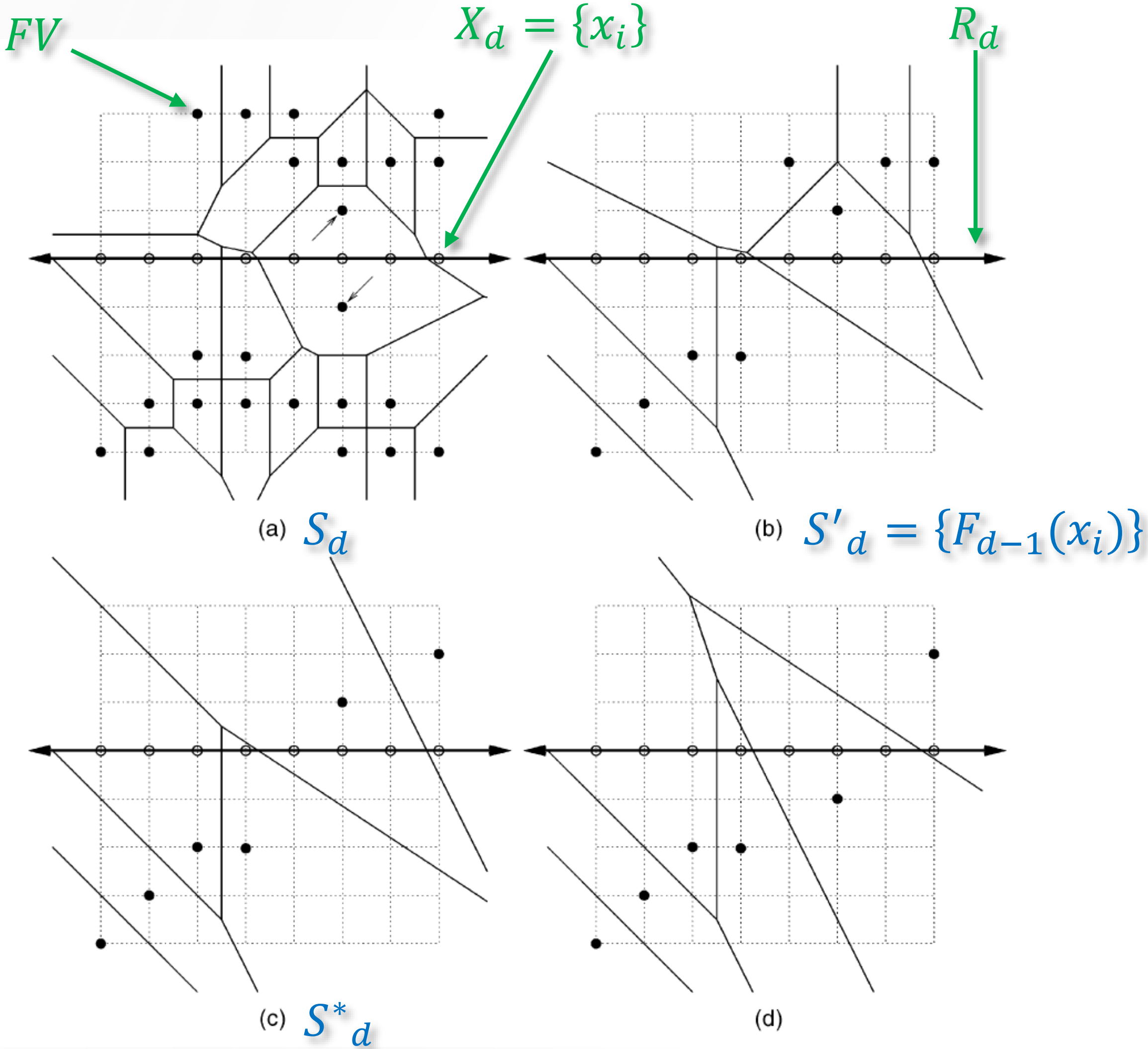


A linear time algorithm for computing exact Euclidean distance transform



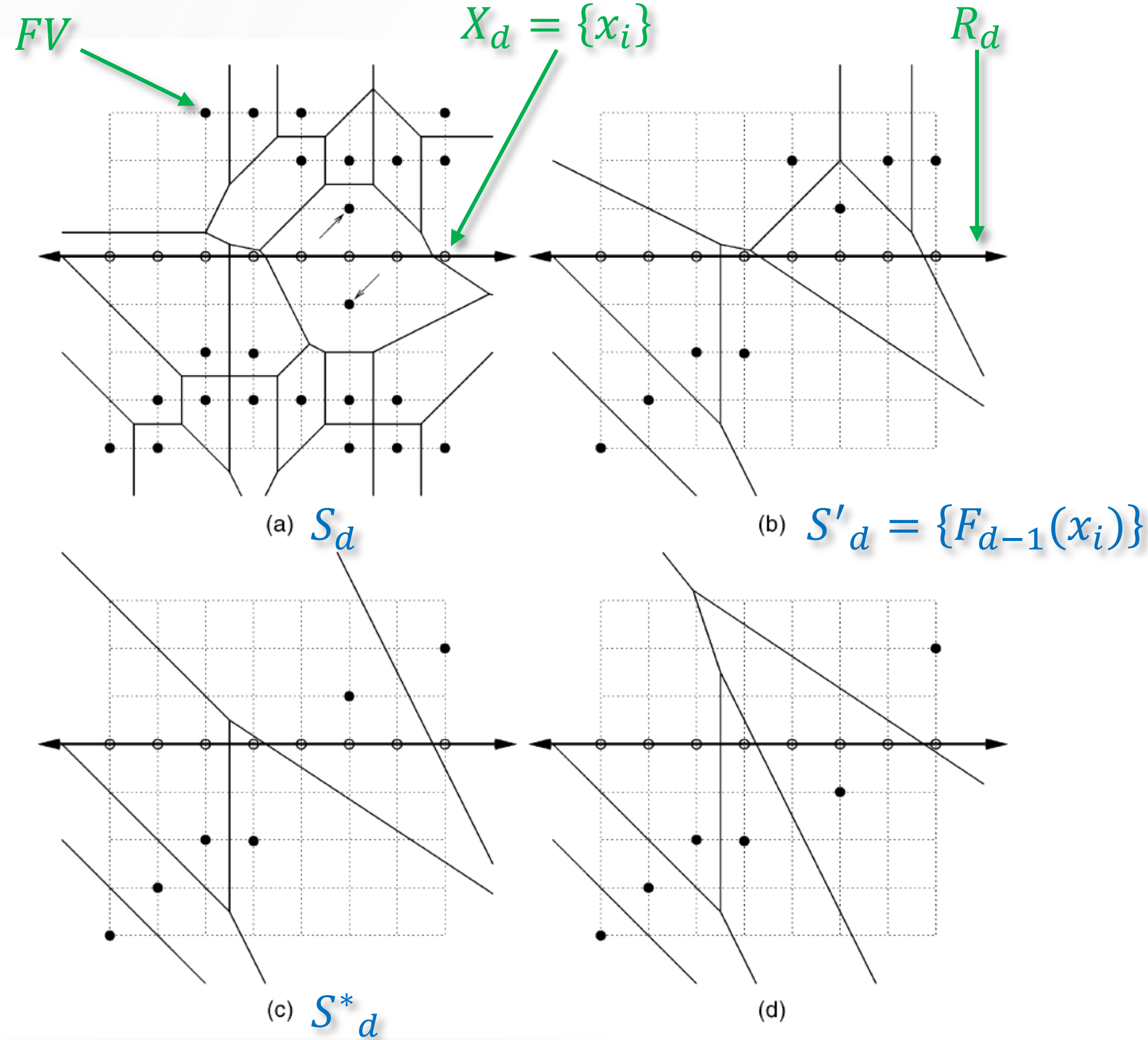
- (a) The FVs shown are the set $\boldsymbol{S_d}$.
- (b) The FVs shown are the set $\boldsymbol{S'_d} = \{F_{d-1}(x_i)\}$
  - the set of CFVs in the next lower dimension for the set of voxels $\{x_i\}$ on the row R_d.
  - $F_{d-1}(x_i)$ is the CFV in the same column as x_i.
- (c) The FVs shown are the set $\boldsymbol{S^*_d}$.
- (d) The FVs shown are the set $\boldsymbol{S^*_d}$.
  - choosing the bottom of the two equidistant FVs in (a).
- S^*_d
  - subset of S_d that are the Voronoi sites(FVs) of Voronoi cells in V^*_d.
  - Voronoi sites of Voronoi cells in V_{S_d} that intersect R_d.
- $S^*_d \subseteq S'_d \subseteq S_d$
- $V^*_d = V_{S_d} \cap R_d = V_{S'_d} \cap R_d = V_{S^*_d} \cap R_d$.

A linear time algorithm for computing exact Euclidean distance transform



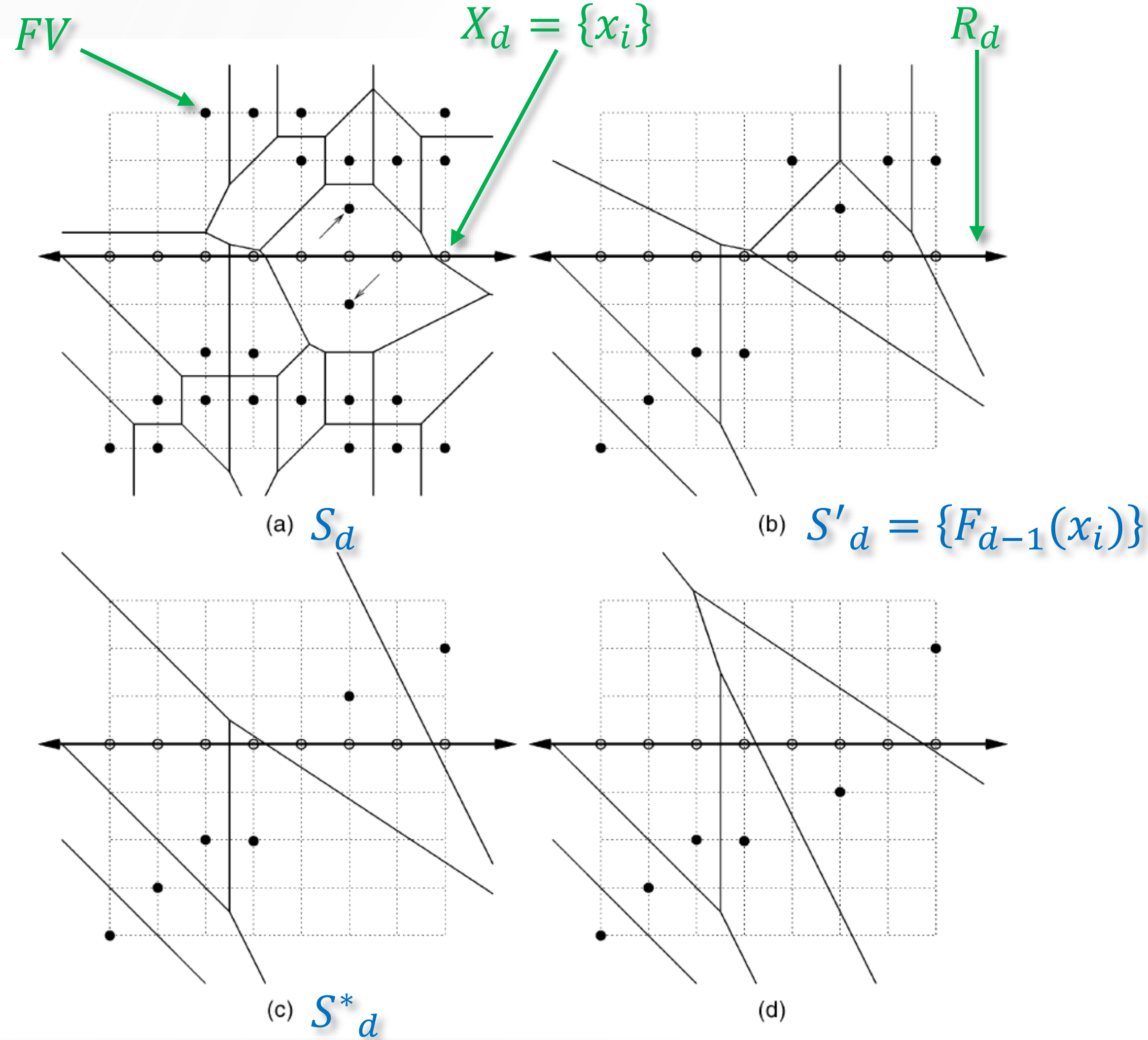
- Constructing V^*_d.
- To construct V^*_d, it is sufficient to consider the set S'_d. (rather than the larger set S_d)
  - from the definition of F_d, $\Delta(x, f) \leq \Delta(x, g)$ where $f = F_{d-1}(x)$ and g is any other FV in S_d.
  - $\Delta(y, f) \leq \Delta(y, g)$ where y is any other point on the row R_d.
  - all points on the row R_d are at least as close to f as they are to g.
  - → Either the Voronoi cell for site g does not intersect R_d, or that the Voronoi cells for site f and g both intersect R_d along a common boundary ($\Delta(x, f) = \Delta(x, g)$).
- $V^*_d = V_{S_d} \cap R_d = V_{S'_d} \cap R_d$.

A linear time algorithm for computing exact Euclidean distance transform

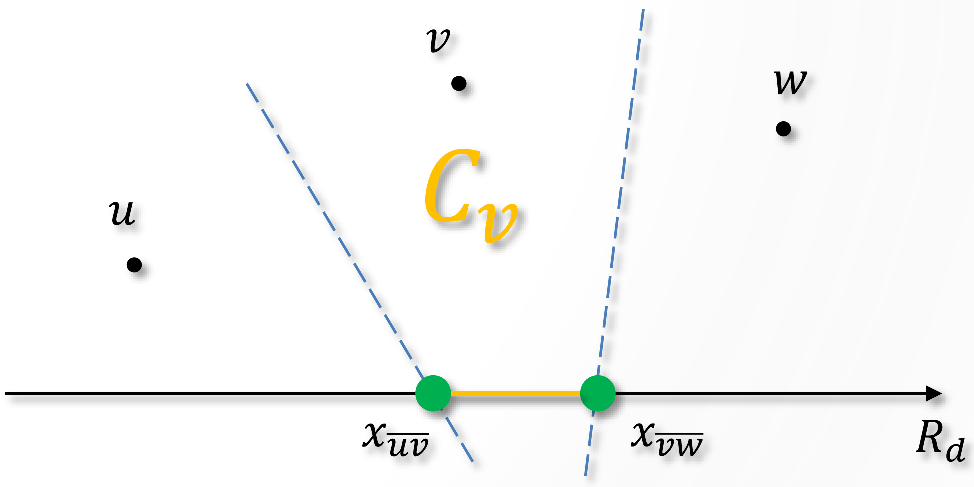


- Construct $V^*_d = \{C^*_{f_i}\}$?
  - → Determine the ordered set S^*_d. Visit each voxel by traversing the row in dth coordinate order.
- Proof :
  - FVs $f, g \in S^*_d$.
    - Voronoi cells C_f, C_g repectively.
  - Voxels $x, y \in R_d$.
  - $x_d < y_d \rightarrow f_d < g_d$.
  - $f_d < g_d \rightarrow x_d < y_d$.
  - Voronoi sites (FVs) S^*_d are sorted by the dth coordinate → associated Voronoi cells are similarly ordered.

A linear time algorithm for computing exact Euclidean distance transform

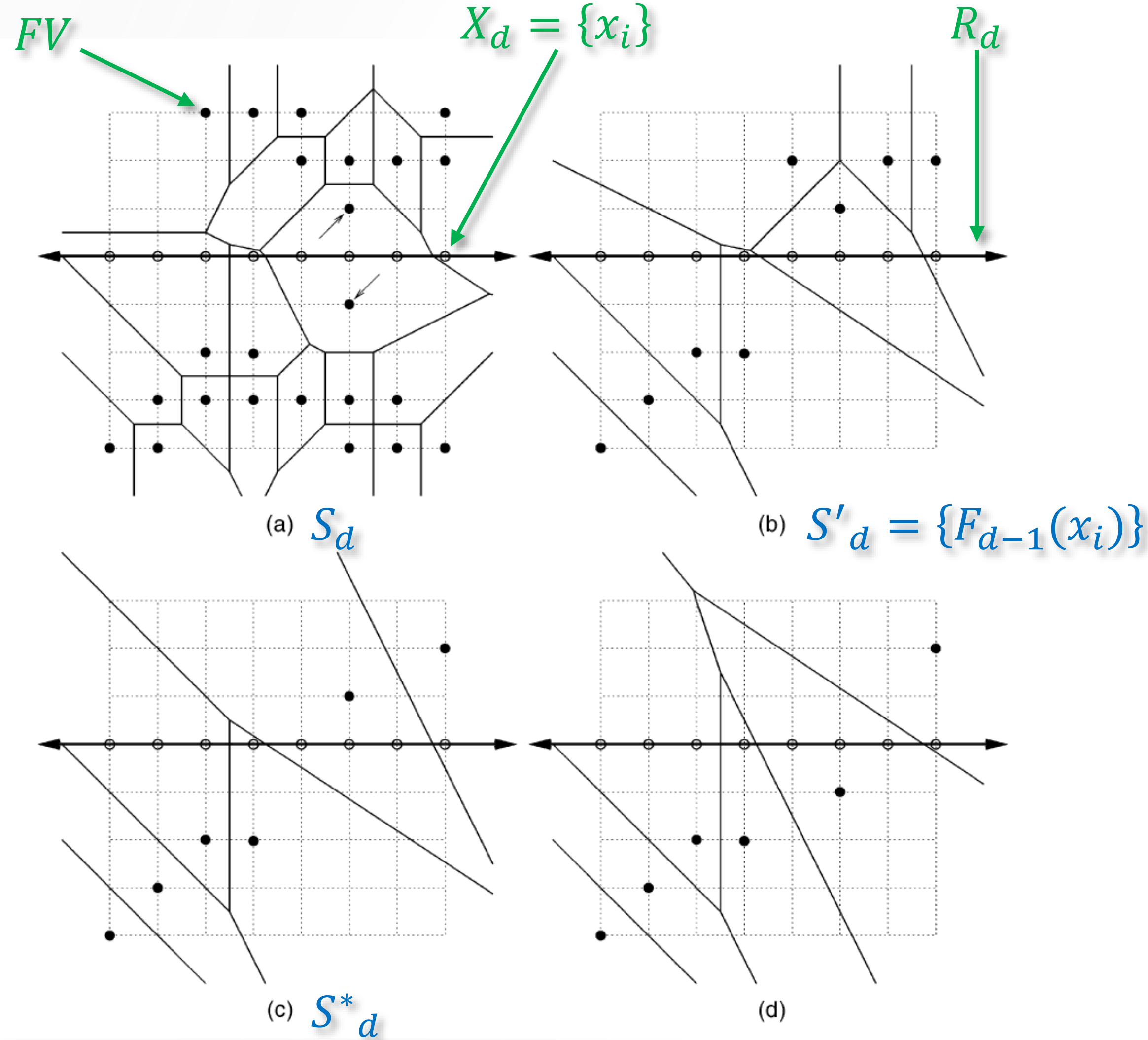


- Cell intersection checking algorithm.
  - Three FVs $u, v, w \in S'_d$.
  - $u_d < v_d < w_d$.
  - $x_{\overline{uv}}$: point on the line R_d that is equidistant from u and v. ($\Delta(u, x_{\overline{uv}}) = \Delta(v, x_{\overline{uv}})$)
  - $(x_{\overline{uv}})_d$: dth coordinate of this point.
  - C_v does not intersect R_d if $(x_{\overline{uv}})_d > (x_{\overline{vw}})_d$.

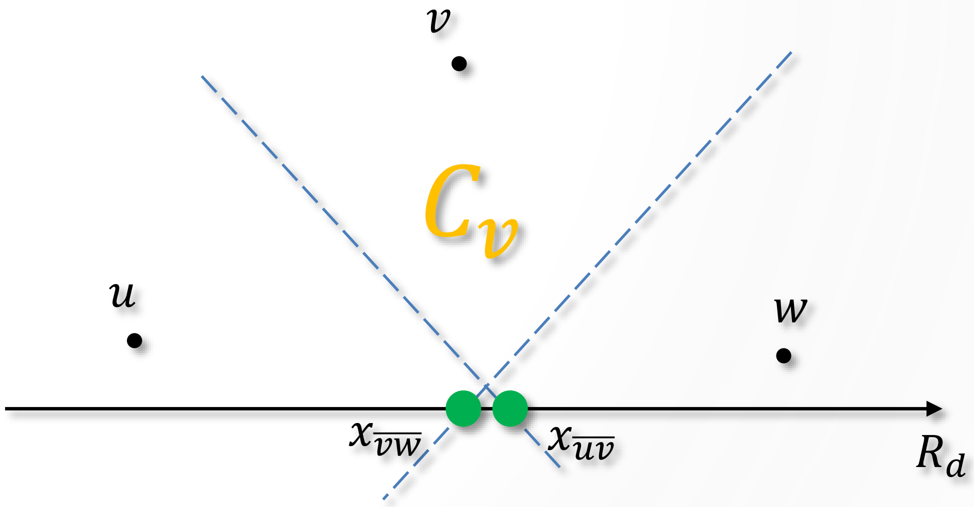


$(x_{\overline{uv}})_d < (x_{\overline{vw}})_d$ → C_v intersects.

A linear time algorithm for computing exact Euclidean distance transform



- Cell intersection checking algorithm.
  - Three FVs $u, v, w \in S'_d$.
  - $u_d < v_d < w_d$.
  - $x_{\overline{uv}}$: point on the line R_d that is equidistant from u and v. $(\Delta(u, x_{\overline{uv}}) = \Delta(v, x_{\overline{uv}}))$
  - $(x_{\overline{uv}})_d$: dth coordinate of this point.
  - C_v does not intersect R_d if $(x_{\overline{uv}})_d > (x_{\overline{vw}})_d$.



$(x_{\overline{uv}})_d > (x_{\overline{vw}})_d$ → C_v does not intersects.

A linear time algorithm for computing exact Euclidean distance transform

- ComputeFT

```
if d = 1 then        /* Compute F_{d-1} */
  for i_1 ← 1 to n_1 do
    if I(i_1, j_2, ..., j_k) = 1 then
      F(i_1, j_2, ..., j_k) ← (i_1, j_2, ..., j_k)
    else
      F(i_1, j_2, ..., j_k) ← φ
    endif
  endfor
else
  for i_d ← 1 to n_d do
    COMPUTEFT(d − 1, i_d, j_{d+1}, ..., j_k)
  endfor
endif
for i_1 ← 1 to n_1 do        /* Compute F_d */
  ...
    for i_{d-1} ← 1 to n_{d-1} do
      VORONOIFT(d, i_1, ..., i_{d-1}, j_{d+1}, ..., j_k)
    endfor
  ...
endfor
return
```

Recursive call for dimensionality reduction.

- VoronoiFT

```
l ← 0        /* Construct partial Voronoi diagram */
for i ← 1 to n_d do
  x_i ← (j_1, ..., j_{d-1}, i, j_{d+1}, ..., j_k)
  if (f_i ← F(x_i)) ≠ φ then
    if l < 2 then
      l ← l + 1, g_l ← f_i
    else
      while l ≥ 2 and REMOVEFT(g_{l-1}, g_l, f_i, R_d) do
        l ← l − 1
      endwhile
      l ← l + 1, g_l ← f_i
    endif
  endif
endfor
if (n_S ← l) = 0 then
  return
endif
l ← 1        /* Query partial Voronoi diagram */
for i ← 1 to n_d do
  while l < n_S and Δ(x_i, g_l) > Δ(x_i, g_{l+1}) do
    l ← l + 1
  endwhile
  F(x_i) ← g_l
endfor
return
```

$V^*{}_d$

A linear time algorithm for computing exact Euclidean distance transform

- ComputeFT

```
if d = 1 then        /* Compute F_{d-1} */
  for i_1 ← 1 to n_1 do
    if I(i_1, j_2, ..., j_k) = 1 then
      F(i_1, j_2, ..., j_k) ← (i_1, j_2, ..., j_k)
    else
      F(i_1, j_2, ..., j_k) ← φ
    endif
  endfor
else
  for i_d ← 1 to n_d do
    COMPUTEFT(d − 1, i_d, j_{d+1}, ..., j_k)
  endfor
endif
for i_1 ← 1 to n_1 do        /* Compute F_d */
  ...
    for i_{d-1} ← 1 to n_{d-1} do
      VORONOIFT(d, i_1, ..., i_{d-1}, j_{d+1}, ..., j_k)
    endfor
  ...
endfor
return
```

Construct F_0
(FT in 0-dimension)

F_{d-1}

F_d

- VoronoiFT

```
l ← 0        /* Construct partial Voronoi diagram */
for i ← 1 to n_d do
  x_i ← (j_1, ..., j_{d-1}, i, j_{d+1}, ..., j_k)
  if (f_i ← F(x_i)) ≠ φ then
    if l < 2 then
      l ← l + 1, g_l ← f_i
    else
      while l ≥ 2 and REMOVEFT(g_{l-1}, g_l, f_i, R_d) do
        l ← l − 1
      endwhile
      l ← l + 1, g_l ← f_i
    endif
  endif
endfor
if (n_S ← l) = 0 then
  return
endif
l ← 1        /* Query partial Voronoi diagram */
for i ← 1 to n_d do
  while l < n_S and Δ(x_i, g_l) > Δ(x_i, g_{l+1}) do
    l ← l + 1
  endwhile
  F(x_i) ← g_l
endfor
return
```

$S'_d = \{f_i\}$

removing.

$S^*_d = \{g_l\}$

determines the ordered set S^*_d

construct FT in d − dimension.

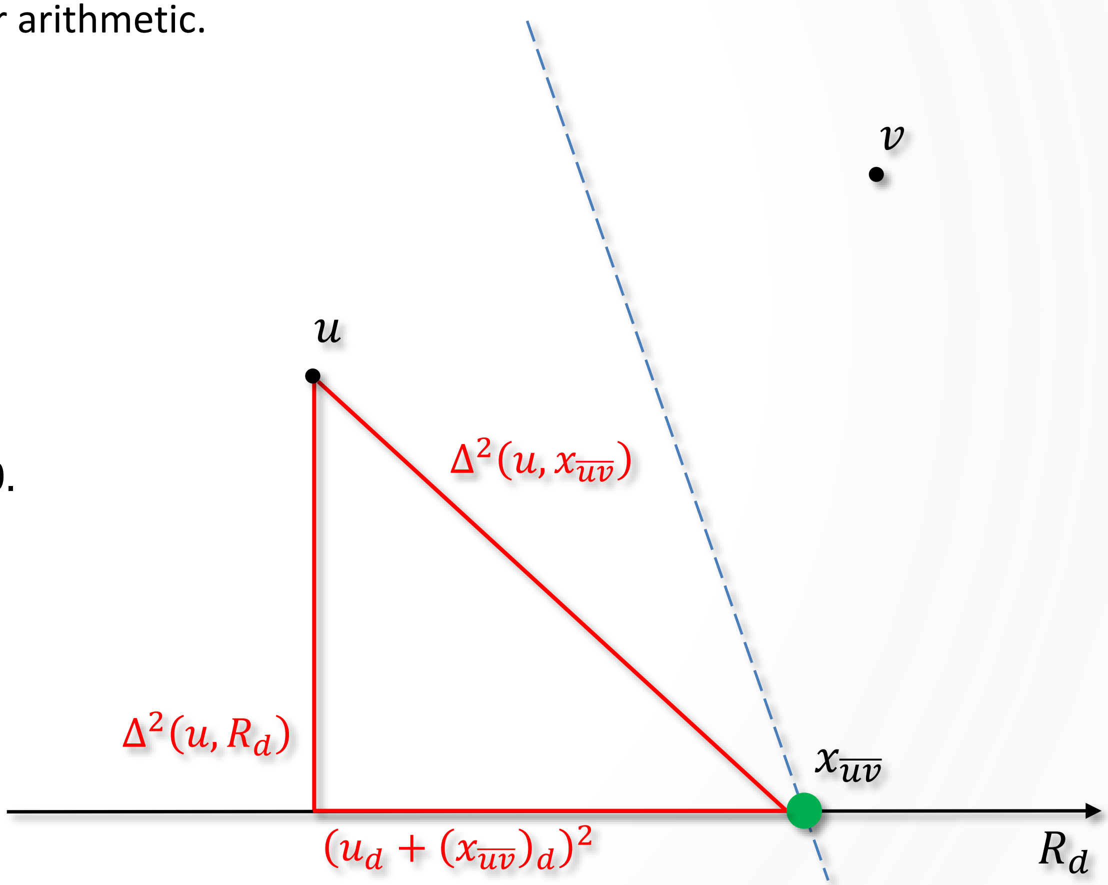
A linear time algorithm for computing exact Euclidean distance transform

- Performance evaluation.
  - Initialization of F_0 takes $O(N)$ time.
  - At each dimension d, VoronoiFT is executed for each of the N/n_d rows.
    - For each row, construction of $S^*{}_d$ takes $O(n_d)$ time, since there are n_d FVs in $S'{}_d$, and each FV is added to and removed from $S^*{}_d$ at most once.
    - Calculating $x_{\overline{uv}}$ requires $O(1)$ time.
  - Querying simply requires $O(n_d)$ time.
  - Thus, at each dimension, the time complexity is $O(n_d \times N/n_d) = O(N)$, and the algorithm for computing the FT of I runs in $O(N)$ time.
    - DT of I can be computed from the FT in $O(N)$ time.

- $\therefore \underline{O(N)}$

A linear time algorithm for computing exact Euclidean distance transform

- Applying Euclidean distance.
  - RemoveFT can be implemented simply using only integer arithmetic.
  - $\Delta^2(u, x_{\overline{uv}}) = \Delta^2(u, R_d) + (u_d + (x_{\overline{uv}})_d)^2$
    - $\Delta^2(u, R_d) = \sum_{i \neq d}(u_i - r_i)^2$
  - $\Delta^2(u, x_{\overline{uv}}) = \Delta^2(v, x_{\overline{uv}})$
    - definition of $x_{\overline{uv}}$.
  - $(x_{\overline{uv}})_d = \frac{\Delta^2(v,R_d) - \Delta^2(u,R_d) + {v_d}^2 - {u_d}^2}{2(v_d - u_d)}$
  - Verifying inequality $(x_{\overline{uv}})_d > (x_{\overline{vw}})_d$ is equivalent to
  - $c \cdot \Delta^2(v, R_d) - b \cdot \Delta^2(u, R_d) - a \cdot \Delta^2(w, R_d) - abc > 0.$
    - $a = v_d - u_d$
    - $b = w_d - v_d$
    - $c = w_d - u_d = a + b$
- $D_d(x) = \Delta^2(x, F_d(x))$
  - $D(x) = \Delta^2(x, F(x))$
  - $u = F_{d-1}(x)$
    - $\Delta^2(u, R_d) = \Delta^2(x, u) = \Delta^2(x, F_{d-1}(x)) = D_{d-1}(x)$



A linear time algorithm for computing exact Euclidean distance transform

- ComputeEDT

```
if $d = 1$ then        /* Compute $D_{d-1}$ */
  for $i_1 \leftarrow 1$ to $n_1$ do
    if $I(i_1, j_2, \ldots, j_k) = 1$ then
      $D(i_1, j_2, \ldots, j_k) \leftarrow 0$
    else
      $D(i_1, j_2, \ldots, j_k) \leftarrow \infty$
    endif
  endfor
else
  for $i_d \leftarrow 1$ to $n_d$ do
    ComputeEDT($d-1, i_d, j_{d+1}, \ldots, j_k$)
  endfor
endif
for $i_1 \leftarrow 1$ to $n_1$ do        /* Compute $D_d$ */
  ...
    for $i_{d-1} \leftarrow 1$ to $n_{d-1}$ do
      VoronoiEDT($d, i_1, \ldots, i_{d-1}, j_{d+1}, \ldots, j_k$)
    endfor
  ...
endfor
return
```

- VoronoiEDT

```
$l \leftarrow 0$        /* Construct partial Voronoi diagram */
for $i \leftarrow 1$ to $n_d$ do
  $\mathbf{x}_i \leftarrow (j_1, \ldots, j_{d-1}, i, j_{d+1}, \ldots, j_k)$
  if $(f_i \leftarrow D(\mathbf{x}_i)) \neq \infty$ then
    if $l < 2$ then
      $l \leftarrow l+1$, $g_l \leftarrow f_i$, $h_l \leftarrow i$
    else
      while $l \geq 2$ and RemoveEDT($g_{l-1}, g_l, f_i, h_{l-1}, h_l, i$) do
        $l \leftarrow l-1$
      endwhile
      $l \leftarrow l+1$, $g_l \leftarrow f_i$, $h_l \leftarrow i$
    endif
  endif
endfor
if $(n_S \leftarrow l) = 0$ then
  return
endif
$l \leftarrow 1$        /* Query partial Voronoi diagram */
for $i \leftarrow 1$ to $n_d$ do
  while $l < n_S$ and $g_l + (h_l - i)^2 > g_{l+1} + (h_{l+1} - i)^2$ do
    $l \leftarrow l+1$
  endwhile
  $D(\mathbf{x}_i) \leftarrow g_l + (h_l - i)^2$
endfor
return
```

A linear time algorithm for computing exact Euclidean distance transform

- ComputeEDT

```
if d = 1 then        /* Compute D_{d-1} */
  for i_1 ← 1 to n_1 do
    if I(i_1, j_2, ..., j_k) = 1 then
      D(i_1, j_2, ..., j_k) ← 0
    else
      D(i_1, j_2, ..., j_k) ← ∞
    endif
  endfor
else
  for i_d ← 1 to n_d do
    COMPUTEEDT(d − 1, i_d, j_{d+1}, ..., j_k)
  endfor
endif
for i_1 ← 1 to n_1 do        /* Compute D_d */
  ...
    for i_{d-1} ← 1 to n_{d-1} do
      VORONOIEDT(d, i_1, ..., i_{d-1}, j_{d+1}, ..., j_k)
    endfor
  ...
endfor
return
```

$f_i = D_{d-1}(x) = \Delta^2(f_i, R_d)$

$h_l : d - th\ coordinate\ of\ g_l$

$g_l = \Delta^2(g_l, R_d)$

- VoronoiEDT

```
l ← 0        /* Construct partial Voronoi diagram */
for i ← 1 to n_d do
  x_i ← (j_1, ..., j_{d-1}, i, j_{d+1}, ..., j_k)
  if (f_i ← D(x_i)) ≠ ∞ then
    if l < 2 then
      l ← l + 1, g_l ← f_i, h_l ← i
    else
      while l ≥ 2 and REMOVEEDT(g_{l-1}, g_l, f_i, h_{l-1}, h_l, i) do
        l ← l − 1
      endwhile
      l ← l + 1, g_l ← f_i, h_l ← i
    endif
  endif
endfor
if (n_S ← l) = 0 then
  return
endif
l ← 1        /* Query partial Voronoi diagram */
for i ← 1 to n_d do
  while l < n_S and g_l + (h_l − i)^2 > g_{l+1} + (h_{l+1} − i)^2 do
    l ← l + 1
  endwhile
  D(x_i) ← g_l + (h_l − i)^2
endfor
return
```

A linear time algorithm for computing exact Euclidean distance transform

- Practical implementations :
  - for medical 3D images w/ anisotropic voxel dimensions, multiply weight (w_i) when calculating distance.
  - all of the arithmetic operations occur in VoronoiEDT, lines 8 and 20.
    - $while\ l \geq 2\ and\ RemoveEDT(\ldots)do$
    - $while\ l < n_s\ and\ g_l + (h_l - i)^2 > g_{l+1} + (h_{l+1} - i)^2\ do$
      - take advantage of similarities in successive computations.
  - for fixed number of dimensions (e.g., k=3), compute D_i consecutively rather than recursion.
  - computation of D_1 can be implemented more efficiently by *forward-and-reverse propagation*.

References

- Maurer Jr, Calvin R., Rensheng Qi, and Vijay Raghavan. "A linear time algorithm for computing exact Euclidean distance transforms of binary images in arbitrary dimensions." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*25.2 (2003): 265-270.

- Saito, Toyofumi, and Jun-Ichiro Toriwaki. "New algorithms for Euclidean distance transformation of an n-dimensional digitized picture with applications."*Pattern recognition* 27.11 (1994): 1551-1565.

Thank you!