Morphologie mathématique 2

Ecole d'été STIC – Sousse 2008

Antoine MANZANERA – ENSTA/LEI

Chapitre 2 : Géométrie discrète & Aspects algorithmiques

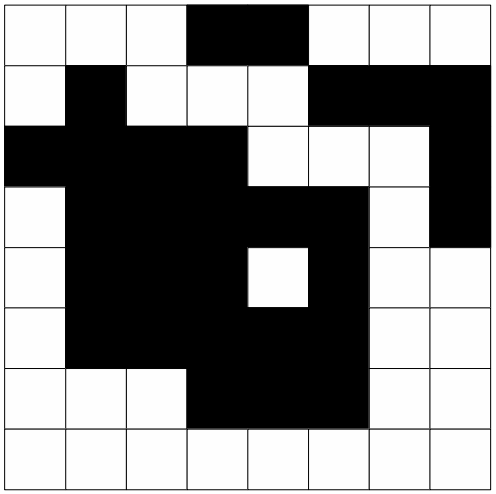
- Introduction à la géométrie discrète
- Topologies et distances discrètes
- Transformées en distances discrètes
- Calcul des opérateurs de base
- Files d'attente et opérateurs géodésiques

Images discrètes : modélisation

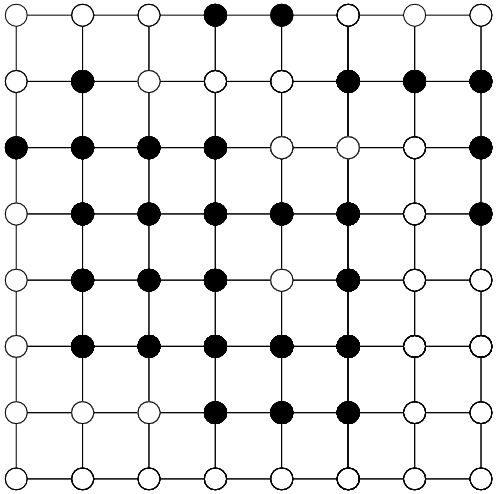
- Le *plan discret* est représenté par $\mathbf{Z}^2$.
- Une *image discrète binaire* est un sous-ensemble de $\mathbf{Z}^2$.
- Une *image discrète en niveau de gris* est une fonction de $\mathbf{Z}^2$ dans $\mathbf{N}$.

$$X \subset \mathbf{Z}^2$$

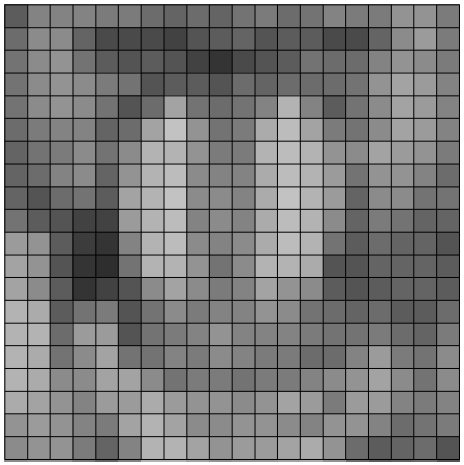
$$I : \mathbf{Z}^2 \rightarrow \mathbf{N}$$



Une image binaire (représentation « pavage »)



Une image binaire (représentation « maillage »)

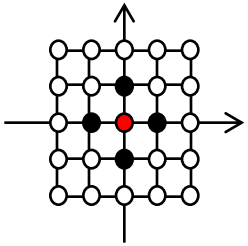


Une image en niveaux de gris

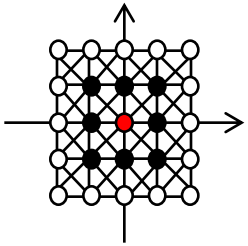
Topologies dans la maille carrée

Dans la maille carrée, on peut définir 2 types de *relations d'adjacence*, donc de topologie :

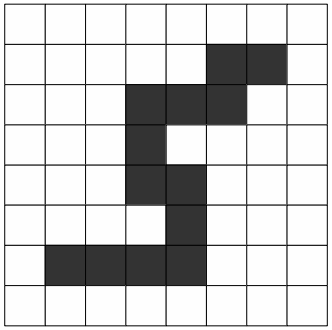
4-connexité :



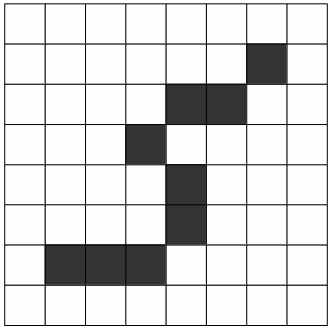
8-connexité :



Par clôture transitive, on définit la *relation de connexion*, qui est une relation d'équivalence :



Un chemin 4-connexe

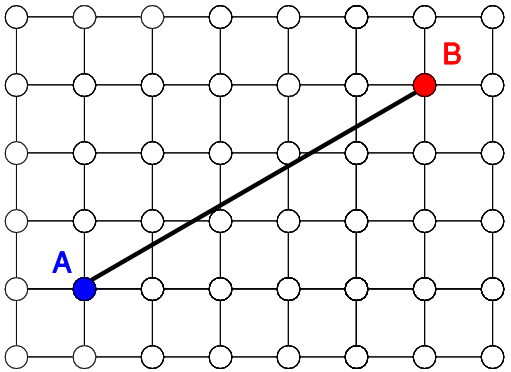


Un chemin 8-connexe

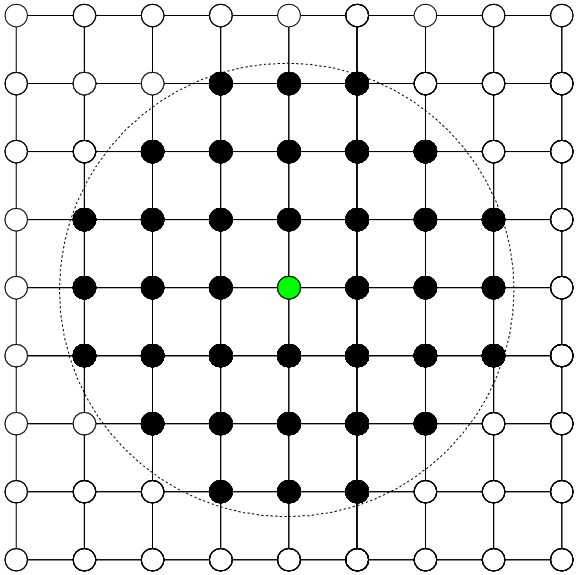
Les classes d'équivalence de la relation de connexion sont les *composantes connexes*.

Distances dans la maille carrée

La distance euclidienne dans la maille carrée se calcule facilement pour 2 points donnés, mais est difficile à manipuler d'un point de vue algorithmique pour calculer la carte de distance à un ensemble donné (transformée en distance).



$$d_E(A,B) = \sqrt{5^2 + 3^2} = \sqrt{34}$$

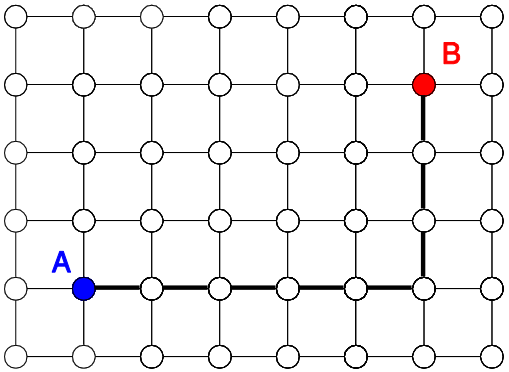


$$B^{d_E}_{\sqrt{10}}(C) = \left\{ z \in \mathbf{Z}^2 ; d_E(z,c) \leq \sqrt{10} \right\}$$

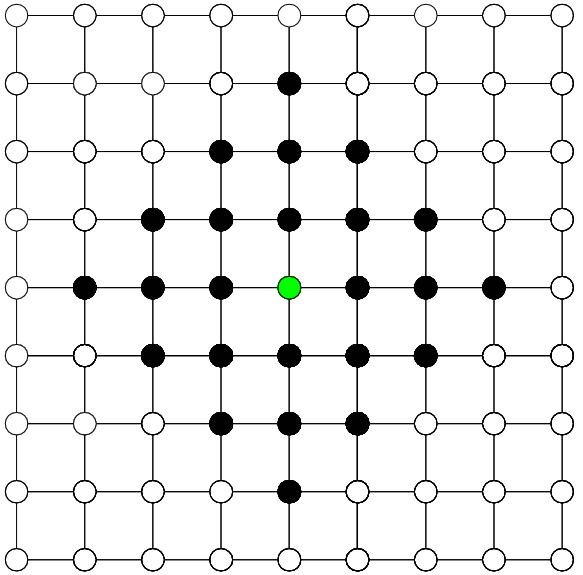
Distances dans la maille carrée

Les distances discrètes plus faciles à manipuler d'un point de vue algorithmique sont celles qui sont induites par la topologie : étant donnée une relation d'adjacence la distance entre A et B est alors définie comme le nombre minimum d'arêtes que compte un chemin qui relie A à B. Par exemple la distance d_4, ou distance de la 4-connexité :

$$d_4(A,B) = |x_A - x_B| + |y_A - y_B|$$



$$d_4(A,B) = 5 + 3 = 8$$

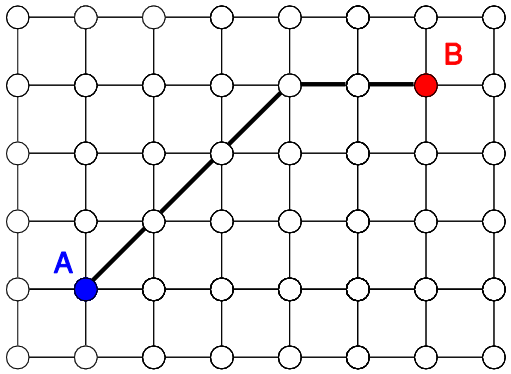


$$B_3^{d_4}(C) = \{z \in \mathbf{Z}^2; d_4(z,c) \leq 3\}$$

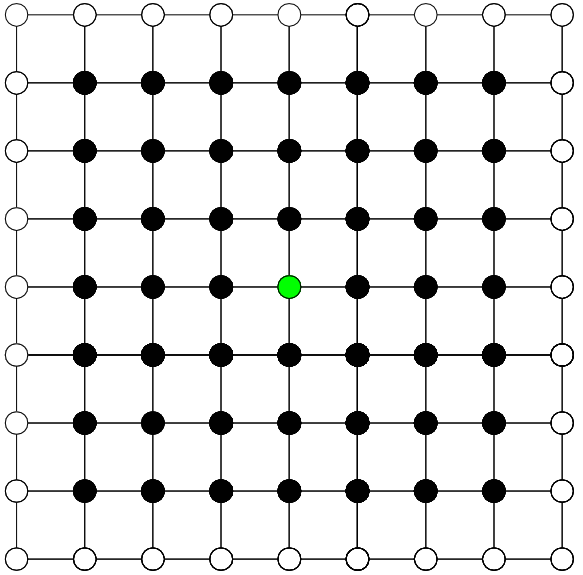
Distances dans la maille carrée

De la même façon, la distance d_8, ou distance de la 8-connexité :

$$d_8(A,B) = \max\left(|x_A - x_B|, |y_A - y_B|\right)$$



$$d_8(A,B) = \max(5,3) = 5$$

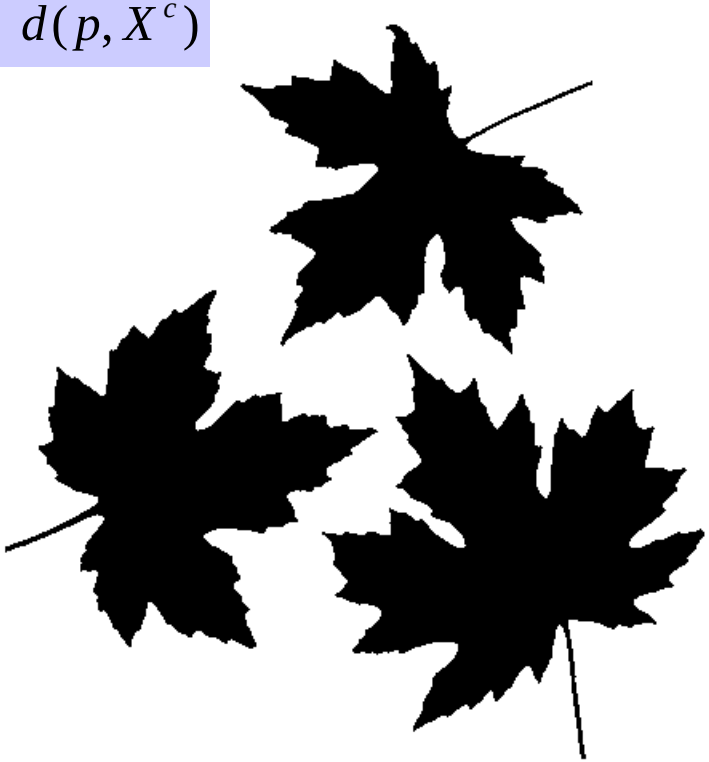


$$B_3^{d_8}(C) = \left\{z \in \mathbf{Z}^2; d_8(z,c) \leq 3\right\}$$

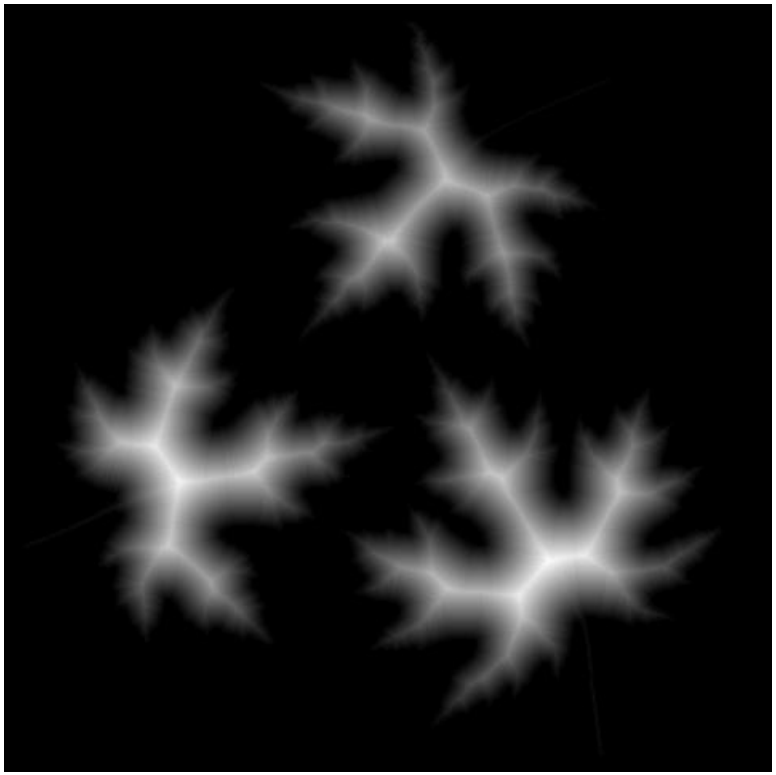
Transformée en distance

La transformée en distance d'une image binaire X est une fonction qui associe à chaque pixel de X sa distance au complémentaire X^c. Cette fonction est très utile en analyse d'images, par exemple pour le calcul des opérateurs morphologiques :

$$\begin{array}{rccc} F_X^d : & \mathbf{Z^2} & \rightarrow & \mathbf{N} \\ & p & \mapsto & d(p, X^c) \end{array}$$



X



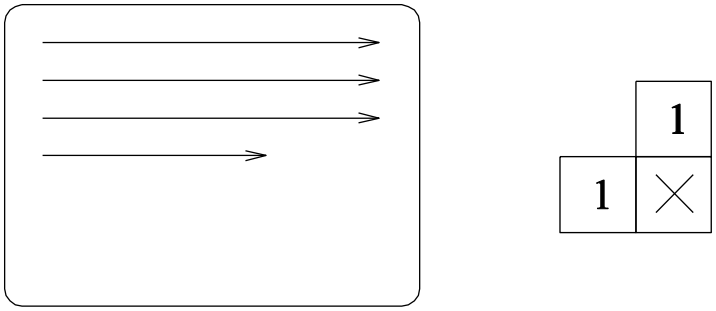
$F_X^{d_E}$

Transformée en distance : Algorithmes

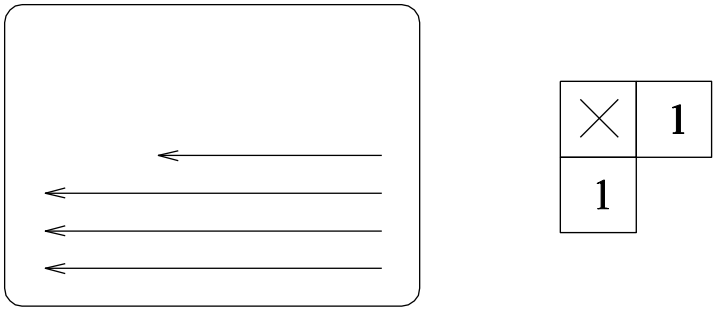
Pour les distances d_4 et d_8, la transformée en distance d'une image binaire X se calcule facilement par un algorithme récursif, basé sur 2 balayages d'image : 1 direct, 1 rétrograde :

```
% Balayage direct
for i = 1:w
  for j = 1:h
    if (i,j)∉X F(i,j)=0;
    else F(i,j) = min(F(i-1,j)+1,F(i,j-1)+1);
  end
end
% Balayage rétrograde
for i = w:-1:1
  for j = h:-1:1
    F(i,j) = min(F(i,j),F(i+1,j)+1,F(i,j+1)+1);
  end
end
```

Algorithme de calcul de la transformée en distance d_4



Balayage direct et son masque de calcul



Balayage rétrograde et son masque de calcul

Transformée en distance : Algorithmes

Illustration du calcul de la transformée en distance d_4 en 2 balayages, sur un exemple :

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

Etat « initial »

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

Après le premier balayage

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

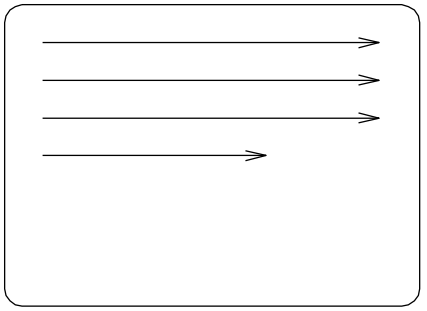
Après les 2 balayages

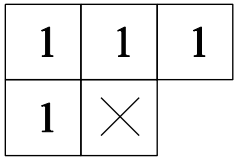
Transformée en distance : Algorithmes

La transformée en distance d_8 se calcule de façon similaire :

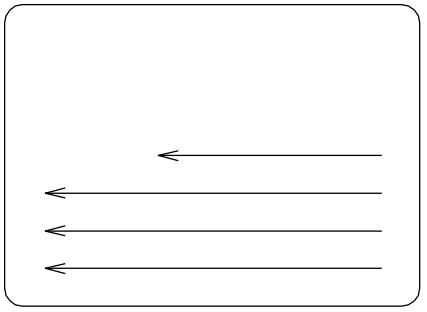
```
% Balayage direct
for i = 1:w
   for j = 1:h
      if (i,j)∉X F(i,j)=0;
      else F(i,j) = min(F(i-1,j)+1,F(i,j-1)+1,F(i-1,j-1)+1);
   end
end
% Balayage rétrograde
for i = w:-1:1
   for j = h:-1:1
      F(i,j) = min(F(i,j),F(i+1,j)+1,F(i,j+1)+1,F(i+1,j+1)+1);
   end
end
```

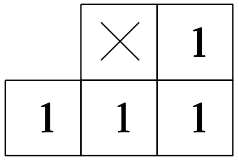
Algorithme de calcul de la transformée en distance d_8





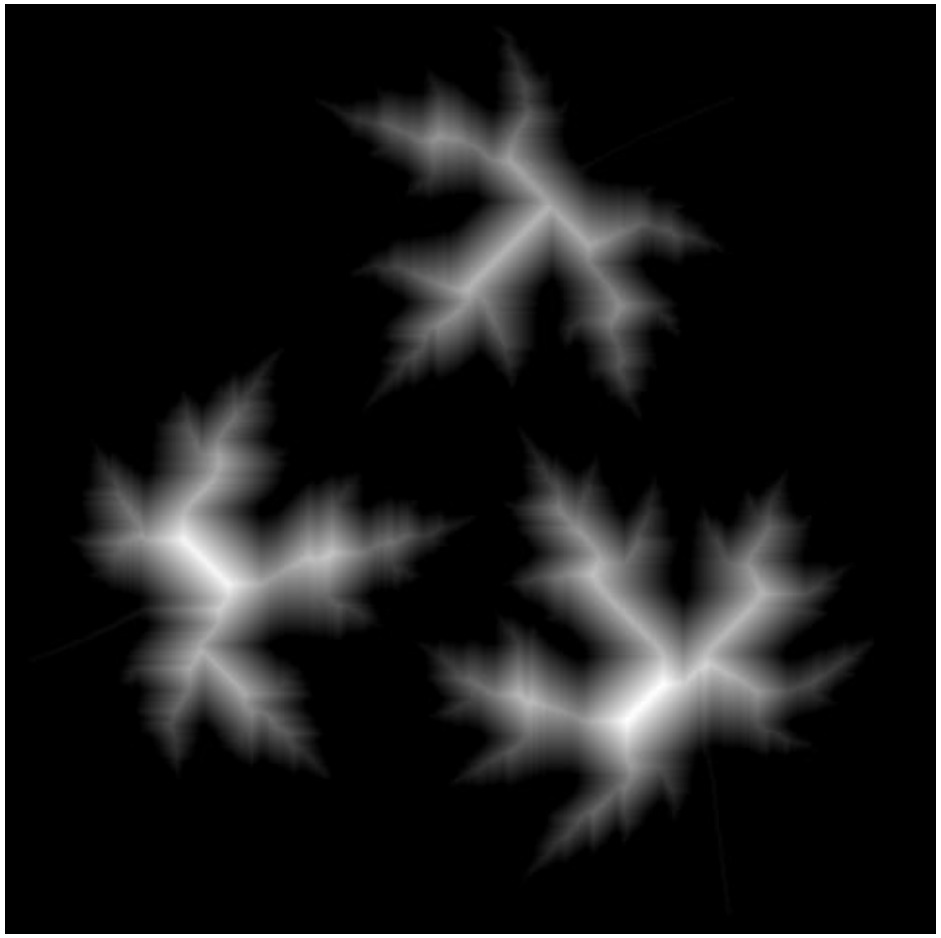
Balayage direct et son masque de calcul



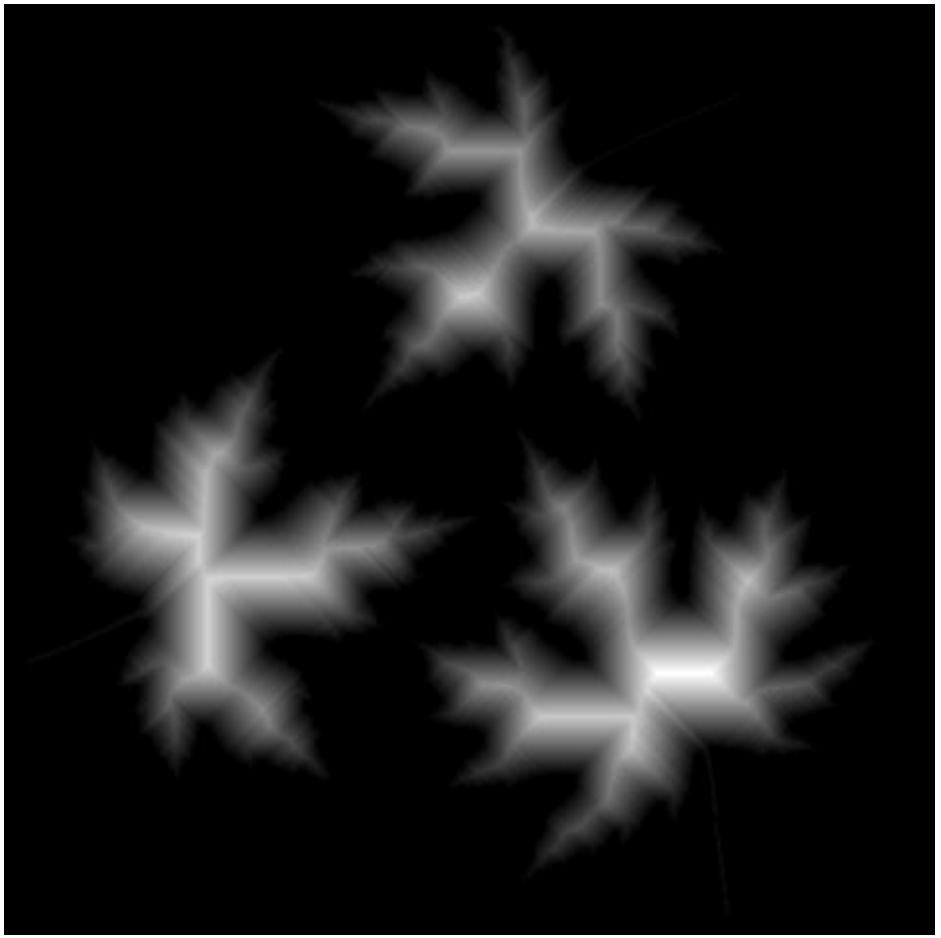


Balayage rétrograde et son masque de calcul

Comparaison distances d_4 et d_8



Transformée en distance d_4

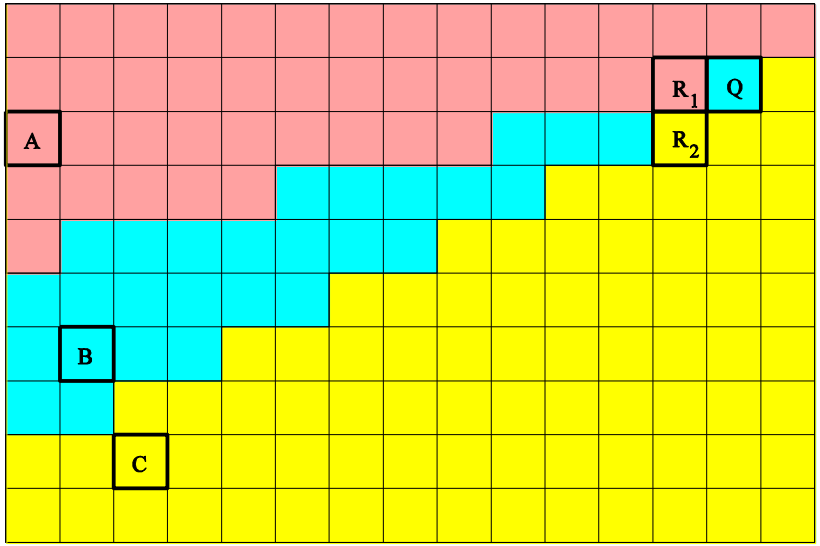


Transformée en distance d_8

Distance euclidienne : algorithmes

On ne peut pas calculer de transformée en distance euclidienne *exacte* en utilisant un algorithme similaire, car la valeur de la transformée en distance en un point ne peut pas toujours être décidée en fonction de la valeur de la transformée en distance de ses 8 plus proches voisins :

Sur la figure ci-contre, le pixel Q est plus proches de B que de A ou de C. Mais tous ses 8 plus proches voisins sont soit plus proches de A, soit plus proches de C, que de B



Distance euclidienne : algorithmes

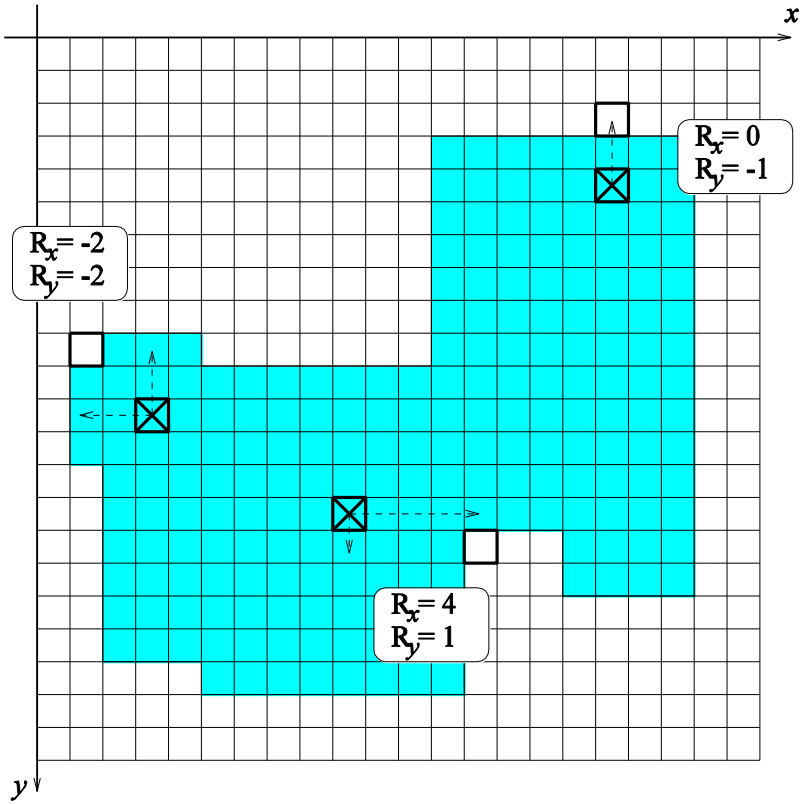
Néanmoins, on peut calculer une très bonne approximation de la transformée en distance euclidienne sur la maille carrée, grâce à l'algorithme de Danielsson-Leymarie (DL). Cet algorithme consiste à calculer récursivement les coordonnées relatives des pixels les plus proches du complémentaire :

L'algorithme consiste à calculer, pour chaque pixel p de X, les coordonnées $(R_x(p),R_y(p))$ *tels que le point de* X^c *le plus proche de p a pour coordonnées :*

$$(x_p + Rx(p), y_p + Ry(p))$$

La valeur de la transformée en distance au point p est donc :

$$F_X^E(p) = \sqrt{\left(R_x(p)\right)^2 + \left(R_y(p)\right)^2}$$



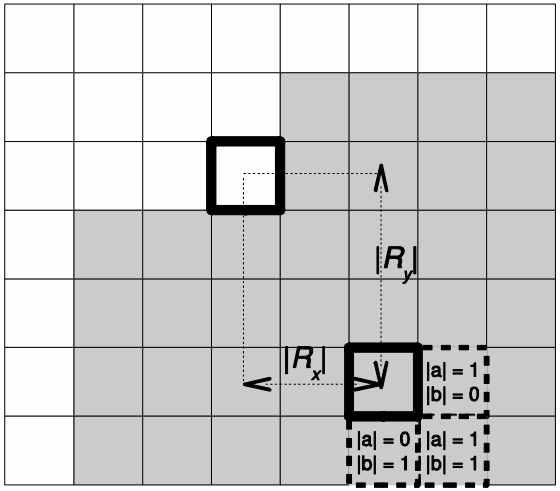
Distance euclidienne : algorithmes

Le carré de la distance euclidienne est calculé par sommation marginale : quand un nombre *n* augmente de 1, son carré augmente de 2*n*+1 :

$$(|R_x|+|a|)^2+(|R_y|+|b|)^2 = {R_x}^2+{R_y}^2+2|R_x a|+2|R_y b|+a^2+b^2$$

et donc :

$$F_X^E(x+a,y+b)^2 = F_X^E(x,y)^2+2|R_x a|+2|R_y b|+a^2+b^2$$



Notations pour l'algorithme :

$V^- = \{(-1,-1),(0,-1),(+1,-1),(-1,0)\}$ le voisinage causal

$V^+ = \{(+1,+1),(0,+1),(-1,+1),(+1,0)\}$ le voisinage anticausal

Enfin, on note : $DF^{(a,b)}(x,y) = 2|aR_x(x+a,y+b)|+2|bR_y(x+a,y+b)|+a^2+b^2$

l'augmentation marginale du carré de la transformée en distance, lorsqu'on passe du point (x+a,y+b) au point (x,y).

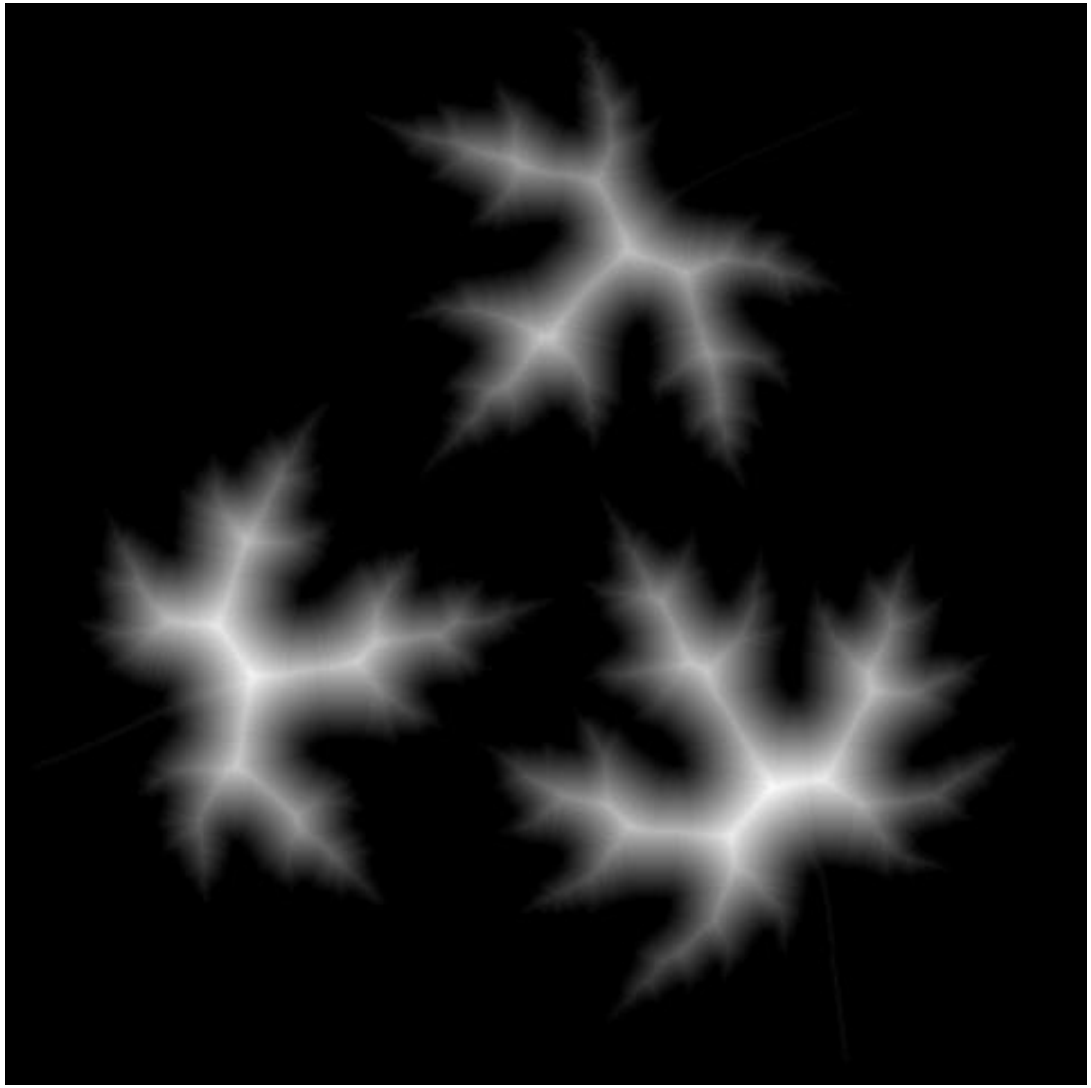
Algorithme de Danielsson-Leymarie

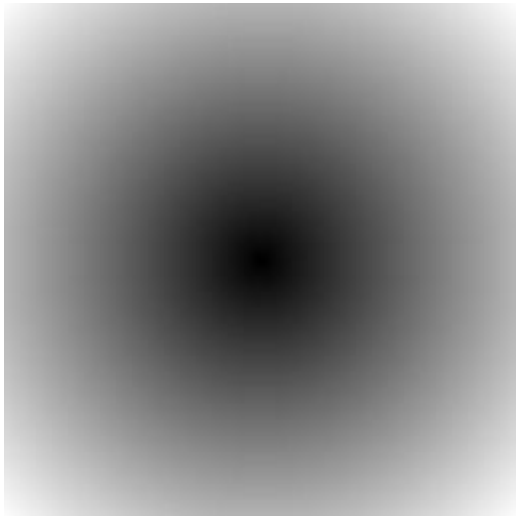
```
for i = 1:w   % Initialisation
  for j = 1:h
    if (i,j)∉X {F(i,j)=0;Rx(i,j)=0;Ry(i,j)=0;}
    else {F(i,j)=∞;Rx(i,j)=0;Ry(i,j)=0;}
  end
end
for i = 1:w   % Balayage direct
  for j = 1:h
    (1) (a,b) = Arg Min { F(i+u,j+v)+DF(u,v)(i,j);(u,v) ∈V- };
    (2) Rx(i,j)=Rx(i+a,j+b)+a ; Ry(i,j)=Ry(i+a,j+b)+b;
    (3) F(i,j) = F(i+a,j+b)+DF(a,b)(i,j);
  end
end
for i = w:-1:1  % Balayage rétrograde
  for j = h:-1:1
    (1) (a,b) = Arg Min { F(i+u,j+v)+DF(u,v)(i,j);(u,v) ∈V+ };
    (2) Rx(i,j)=Rx(i+a,j+b)+a ; Ry(i,j)=Ry(i+a,j+b)+b;
    (3) F(i,j) = F(i+a,j+b)+DF(a,b)(i,j);
end
end
```

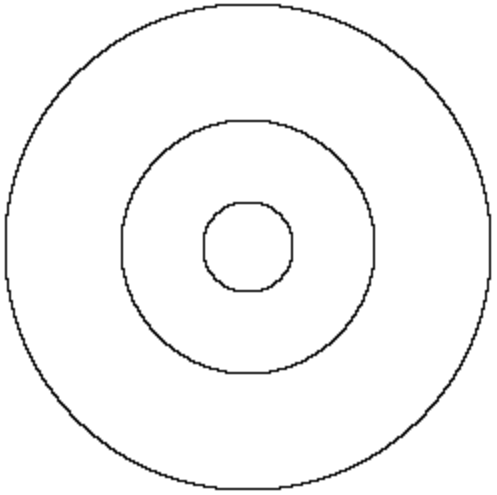
Algorithme de calcul de la transformée en distance quasi-euclidienne par l'algorithme DL en 2 passes

L'algorithme DL a une complexité constante par pixel. L'algorithme ci-contre, en 2 passes, nécessite 8 décalages (multiplication par 2), 12 sommes et 6 comparaisons par pixel. En réalité, l'algorithme en 2 passes produit des erreurs qui peuvent être corrigées par des balayages supplémentaires (utilisant des masques plus petits). L'algorithme DL complet a donc une complexité de : 8 décalages, 14 sommes et 8 comparaisons par pixel.

Algorithme de DL : résultats

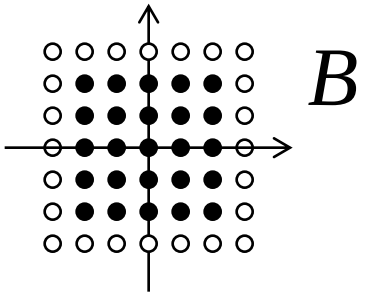






Implantation des opérateurs de base

Ex : élément structurant carré de coté *c*.



Méthode triviale :

```
DILATE (Image_IN X, Image_OUT Y,Elt_struct B) {
Pour tout pixel p ∈X {
  Y(p) = 0;
  Pour tout b ∈B {
     Y(p) = Y(p) OU X(p-b);
     }
  }
}
```

X

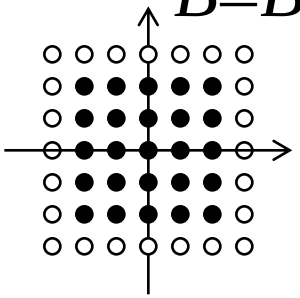
$\delta_B(X)$

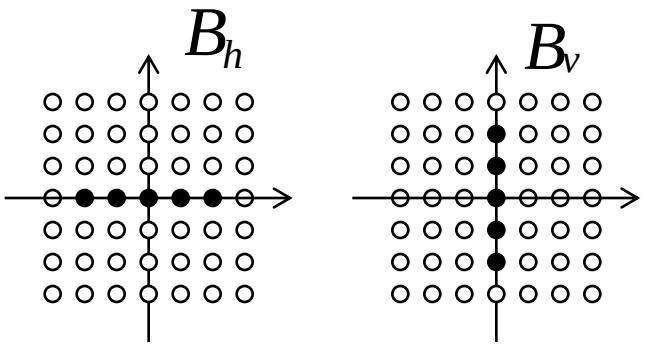
Complexité du calcul par pixel : c^2

Implantation des opérateurs de base

Elément structurant carré de coté *c*.

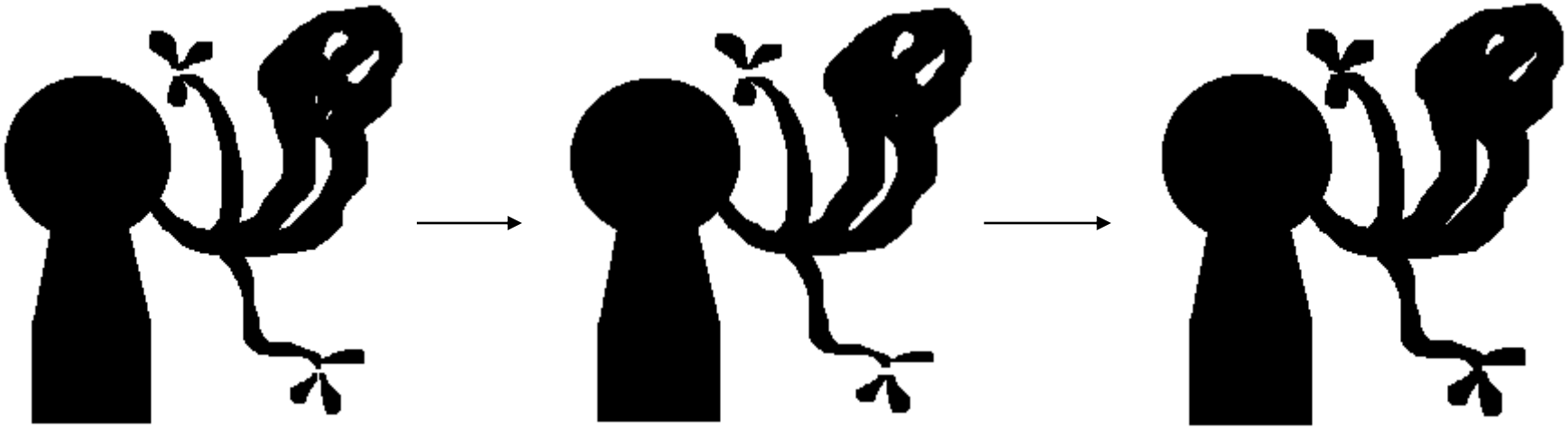
$B = B_h \oplus B_v$





(décomposition des polyèdres de Steiner)

Complexité du calcul par pixel : *2c*



X

$\delta_{B_h}(X)$

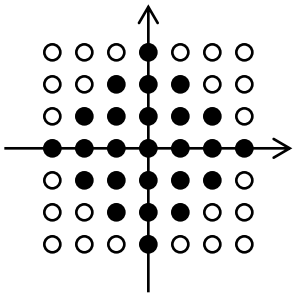
$\delta_{B_v}\left(\delta_{B_h}(X)\right) = \delta_{\delta_{B_v}(B_h)}(X) = \delta_B(X)$

Erosions binaires et distances discrètes

Pour les ensembles (images binaires), dans le cas où l'élément structurant est une boule d'une distance discrète, on calculera l'érodé *par seuillage de la transformée en distance* :

ex :

distance de la 4-connexité

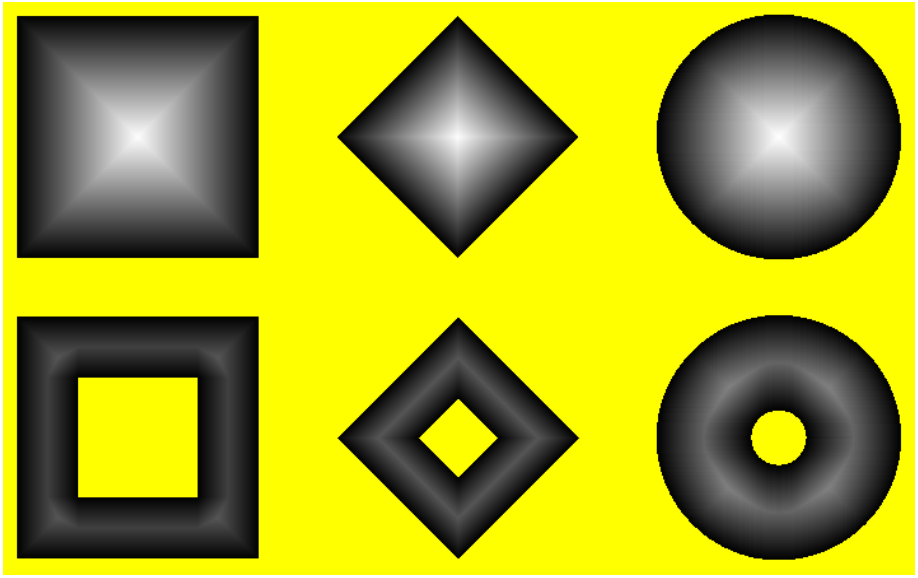


$$d_4(a,b) = |x_a - x_b| + |y_a - y_b|$$

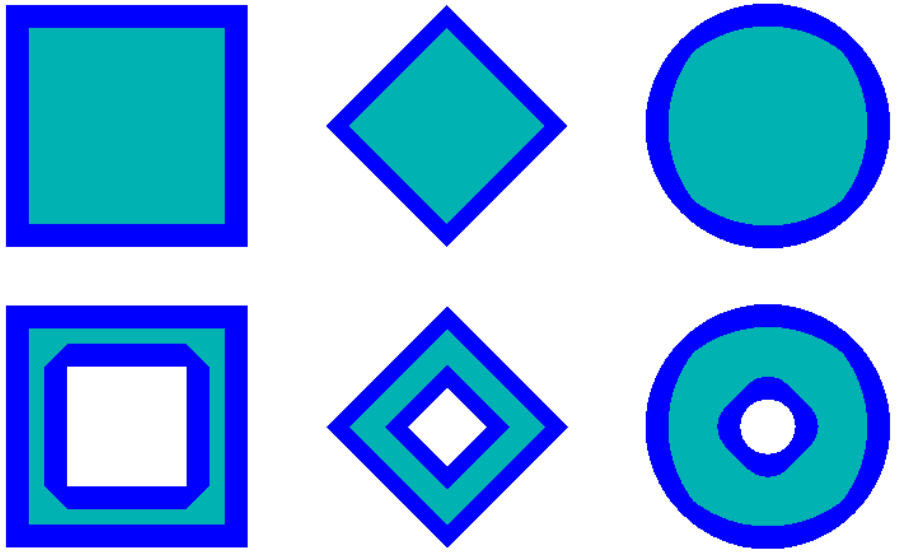
en effet : $p \in \varepsilon_{B_\lambda}(X) \Leftrightarrow F_X^d(p) \geq \lambda$

transformée en distance d de l'ensemble X :

$$F_X^d : \begin{array}{ccc} \mathbf{Z}^2 & \rightarrow & \mathbf{N} \\ p & \mapsto & d(p, X^c) \end{array}$$



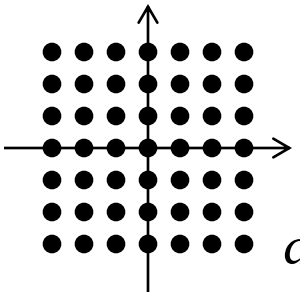
Transformée en distance d_4



Erosion par une boule de d_4

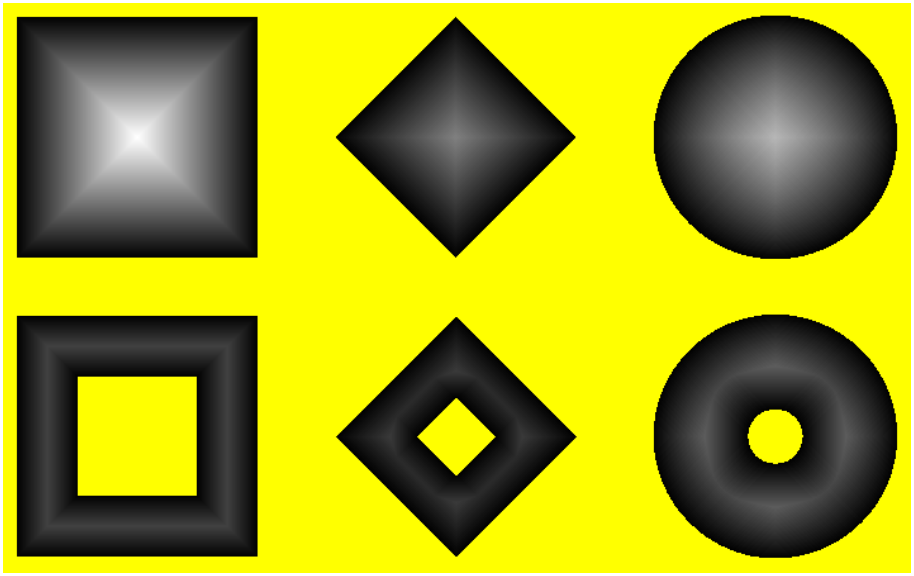
Erosions binaires et distances discrètes

ex :

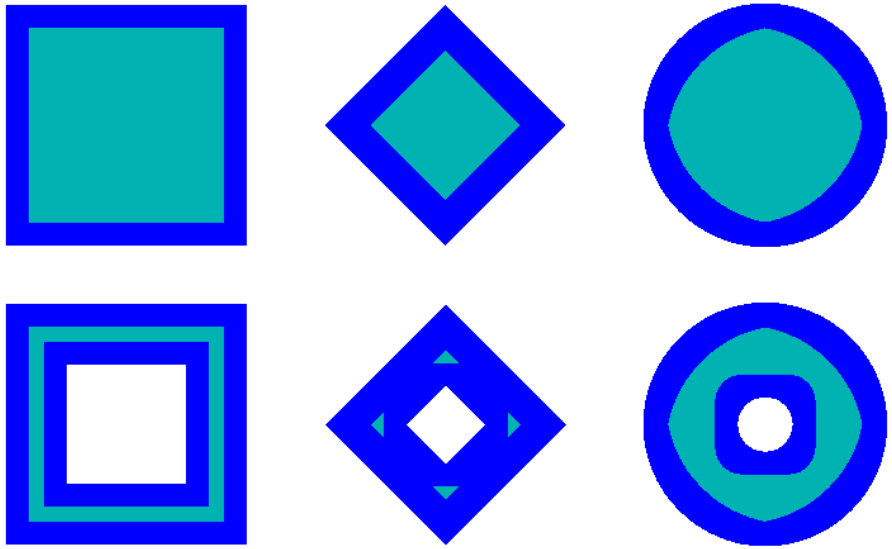


distance de la 8-connexité

$$d_8(a,b) = max(|x_a - x_b|, |y_a - y_b|)$$



Transformée en distance d_8

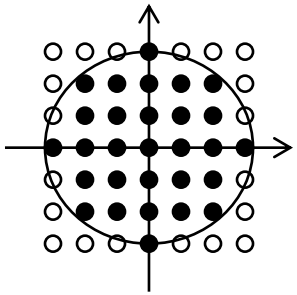


Erosion par une boule de d_8

Erosions binaires et distances discrètes

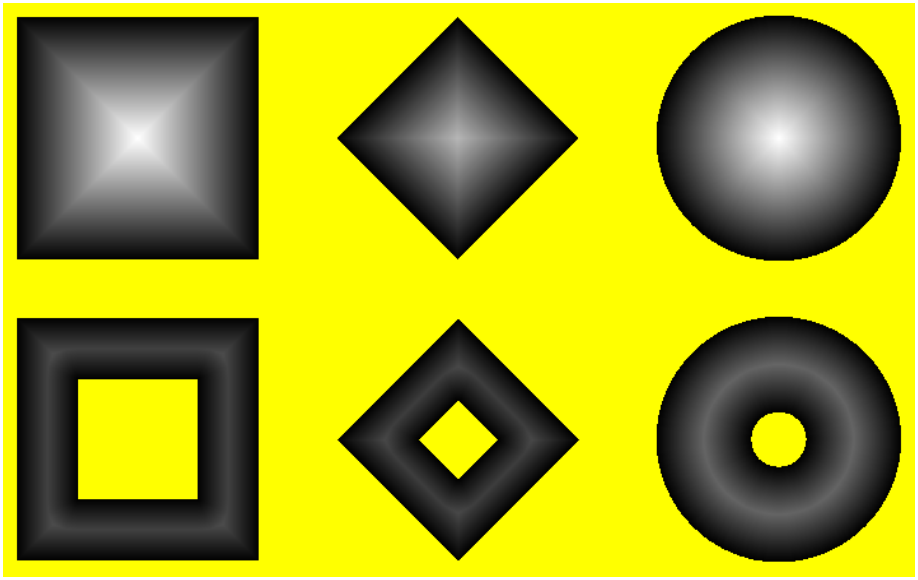
ex :

distance euclidienne

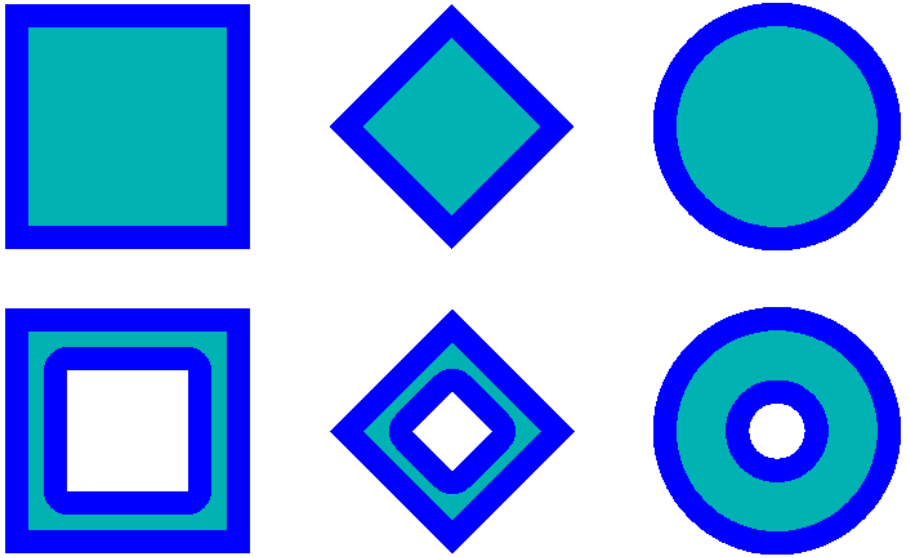


$$d_e(a,b) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}$$

Grâce aux techniques de calcul récursif de la transformée en distance, la complexité du calcul par pixel devient constante : *(O(1))*



Transformée en distance *quasi-euclidienne*



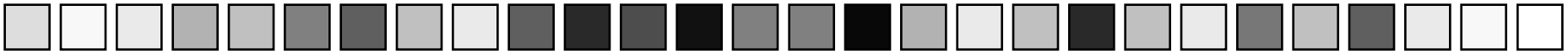
Erosion par une boule *quasi-euclidienne*

Implantation des opérateurs en niveaux de gris

L'implantation de l'érosion par calcul de la fonction distance n'est valable que pour les opérateurs ensemblistes. Existe-t-il des algorithmes pour le calcul de l'érosion en niveaux de gris, dont la complexité soit indépendante de la taille de l'élément structurant ?

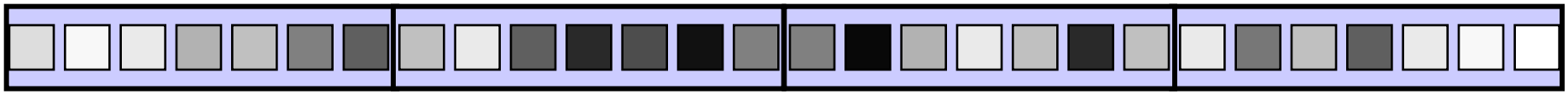
OUI ! Dans le cas d'élément structurant 1D (segment), nous détaillons ci-dessous l'algorithme de Van Herk :

Soit X une image 1D à valeurs numériques :



Soit B un segment de taille K ($K = 2p + 1$). Supposons qu'on souhaite calculer l'érosion de X par B.

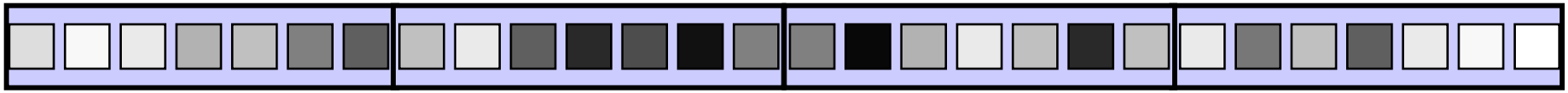
On « partitionne » X en segment de taille K :



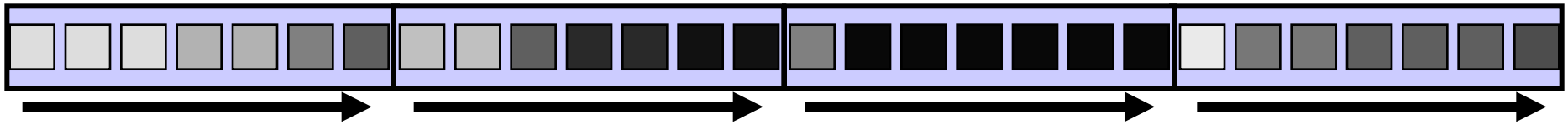
L'algorithme de Van Herk comprend 3 phases :

Van Herk / extrema récursifs par blocs

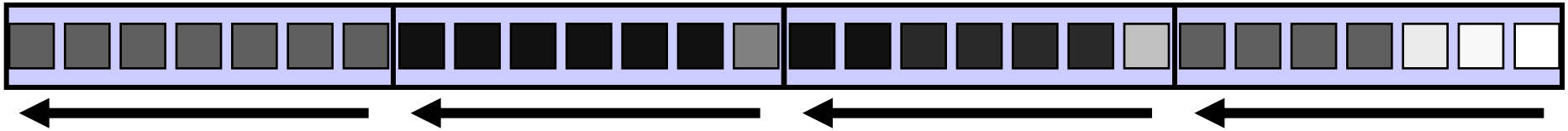
$X =$



$E_1 =$



$E_2 =$



Phase (1) :

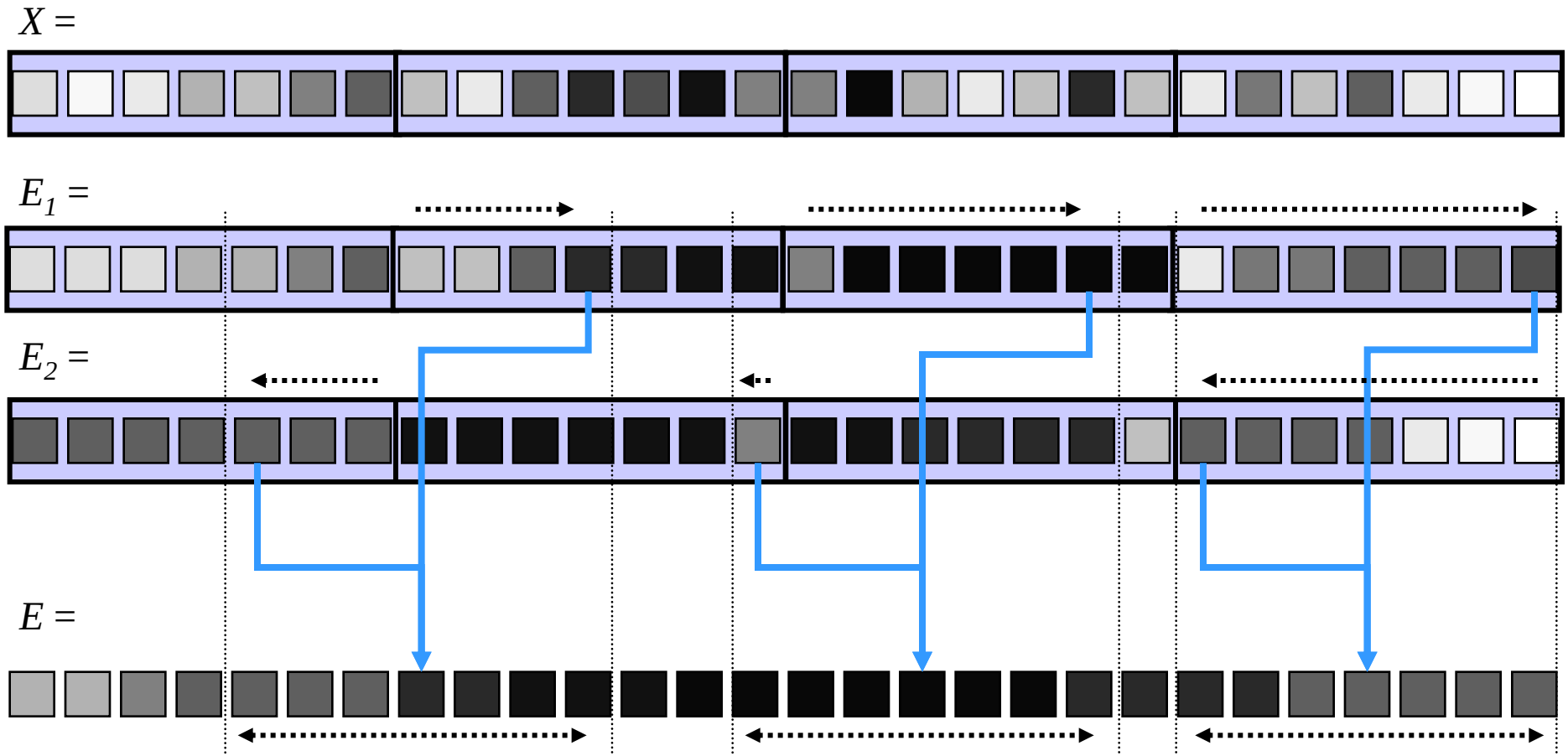
```
for (i = 0; i < W ; i++)
  if (i % K == 0)
    E1[i] = X[i];
  else
    E1[i] = min(E1[i-1],X[i]);
```

Phase (2) :

```
for (i = W-1; i > 0 ; i--)
  if (i % K == 0)
    E2[i] = X[i];
  else
    E2[i] = min(E2[i+1],X[i]);
```

Rq : les calculs de E_1 et de E_2 sont indépendants et peuvent être réalisés en parallèle.

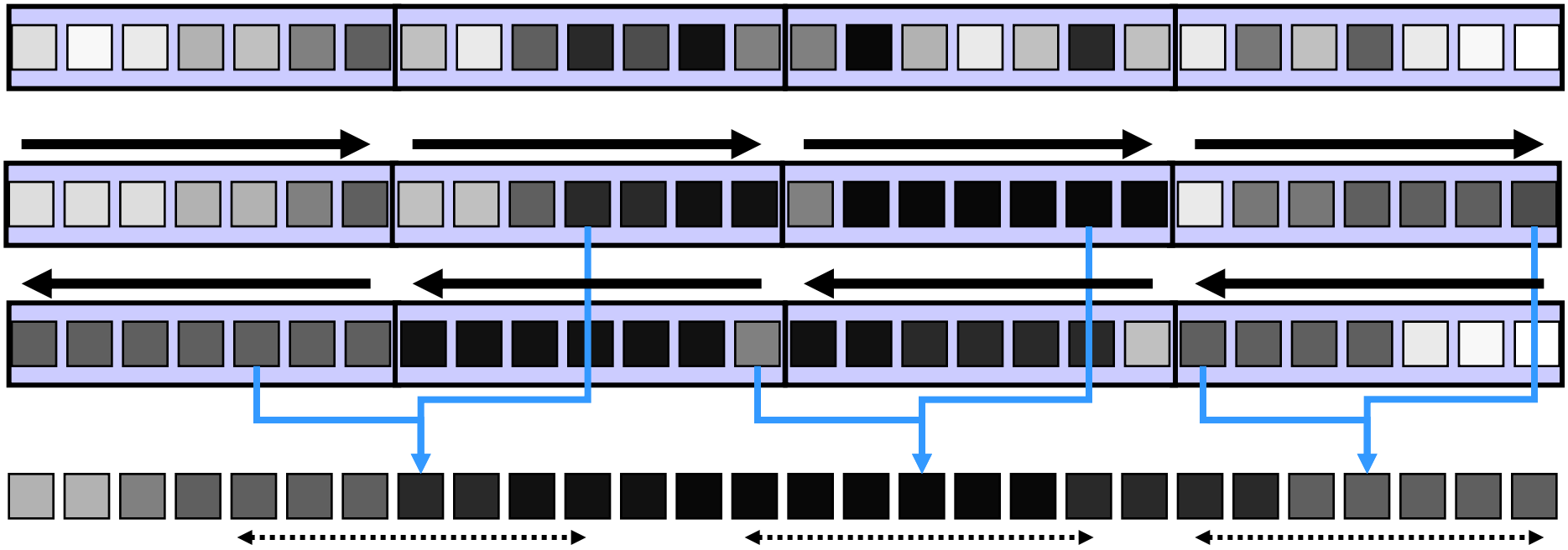
Van Herk / Calcul érosion/dilatation



Phase (3) :

```
for (i = 0; i < W ; i++)
    E[i] = min(E1[i+K/2],E2[i-K/2]);
```

Van Herk / Conclusion



- Complexité : 3 min/max quelque soit la longueur de l'élément structurant.
- Adapté à un calcul séquentiel, mais compatible avec un parallélisme de données.
- Adaptable à des éléments structurants rectilignes de n'importe quelle orientation.

[Van Herk 92]

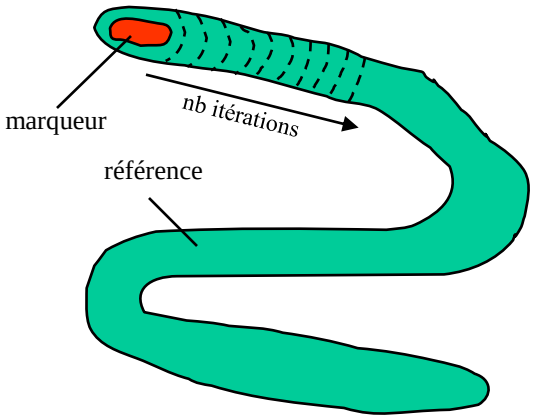
Reconstruction : algorithmique

RECONSTRUCTION NAÏVE

Sur une architecture séquentielle, l'implantation « naïve » de la reconstruction, *i.e.* basée sur la définition :

$$\begin{cases} \delta_g^r(f) = \delta_g(f) \wedge r \\ E_g^r(f) = \sup_{n \geq 0} \left\{ (\delta_g^r)^n(f) \right\} \end{cases}$$

conduit à un coût de calcul tout à fait prohibitif, puisque le nombre d'itérations de dilatation géodésique peut être égal au diamètre géodésique des plus grandes composantes connexes :

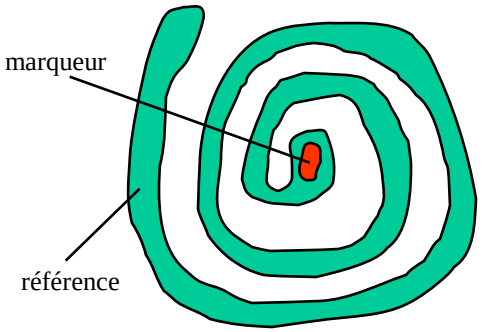


RECONSTRUCTION SEQUENTIELLE

Une implantation sensiblement plus efficace consiste à « propager » le marqueur au cours d'un balayage séquentiel, direct puis rétrograde :

```
RECONSTRUIT (Marqueur M, Référence R) {
  Répéter jusqu'à stabilité {
    // Balayage direct
    Pour j de 0 à h {
      Pour i de 0 à w {
        M(i,j) = MIN(R(i,j),MAX(M(i-1,j),M(i,j-1),M(i,j)));
        }
      }
    // Balayage rétrograde
    Pour j de h à 0 {
      Pour i de w à 0 {
        M(i,j) = MIN(R(i,j),MAX(M(i+1,j),M(i,j+1),M(i,j));
        }
      }
  }
```

Néanmoins, le nombre d'itérations de double balayage peut parfois être important dans le cas de composantes connexes enroulées, par exemple :

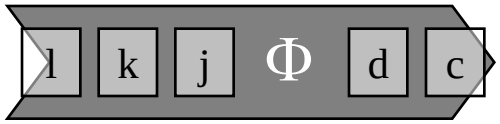


Algorithmique des files d'attente

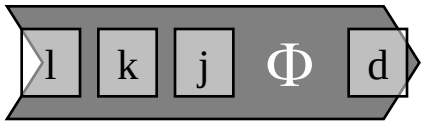
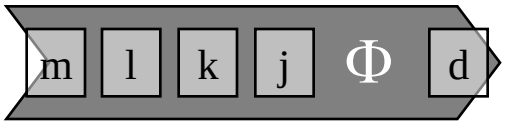
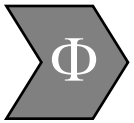
La file d'attente (FIFO) est une structure de donnée particulièrement utile dans les algorithmes morphologiques à base de reconstruction géodésique. Son intérêt est multiple :

• On restreint les calculs aux pixels susceptibles de changer : on examine les pixels qui sont dans la file d'attente, et pas tous les pixels de l'image.

• La terminaison d'un algorithme de relaxation est rendue visible par le fait que la file d'attente est vide. On n'a donc plus besoin de garder une trace explicite des changements pour détecter la convergence.



Φ est la file d'attente.
c est la valeur de l'élément de tête.
l celle de l'élément de queue.

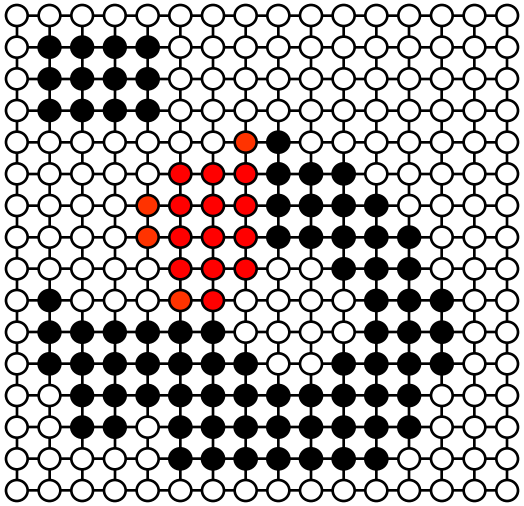
x = pop(Φ)		La fonction **POP(Φ)** supprime l'élément de tête et renvoie sa valeur, soit **x = c**.
push(Φ,y)		La procédure **PUSH(y,Φ)** ajoute en queue de **Φ** un nouvel élément de valeur **y**, soit **m = y**.
empty(Φ) == TRUE		La fonction **empty(Φ)** est une fonction booléenne qui renvoie **1** si et seulement si **Φ** est vide.

La structure de donnée* File d'attente *et ses fonctions associées.

Reconstruction binaire à base de files d'attente

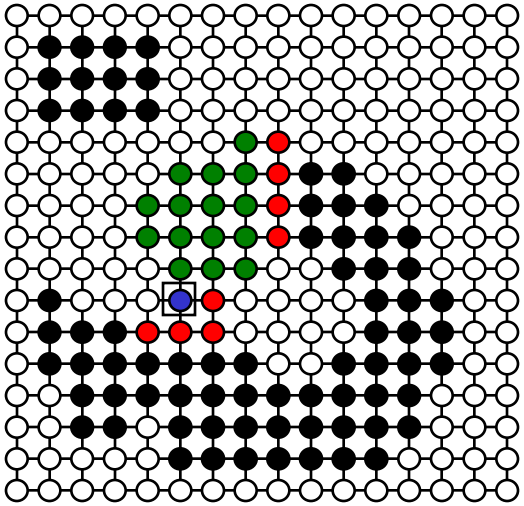
La reconstruction par file d'attente consiste à initialiser la FIFO avec le marqueur, puis pour chaque élément de la FIFO extrait, rajouter ses voisins dans l'image, ainsi jusqu'à convergence (FIFO vide). Le nombre d'opération est proportionnel au nombre de pixels « ajoutés » au marqueur…

Initialisation



- Image
- Marqueur
- Complémentaire

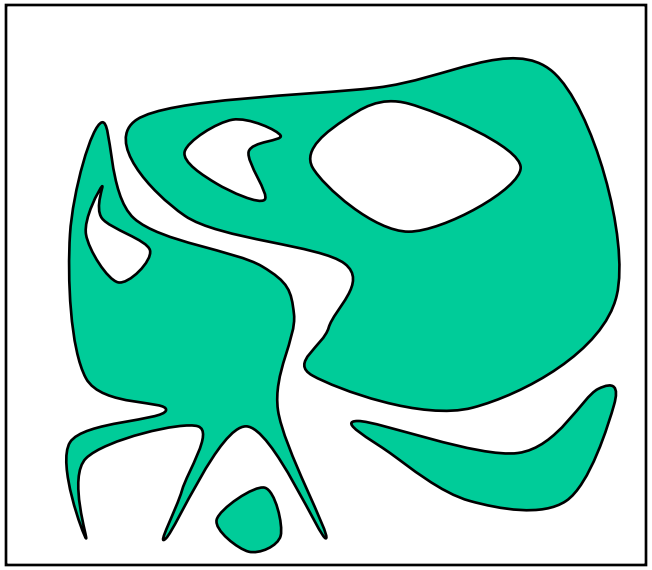
Parcours



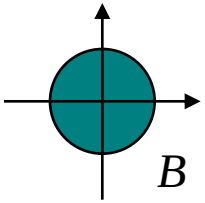
- pixel traité
- pixel à traiter
- pixel en cours

Ouvertures et fermetures par reconstruction

L'ouverture par reconstruction élimine les composantes connexes qui n'appartiennent pas à l'ouvert sans modifier les autres :

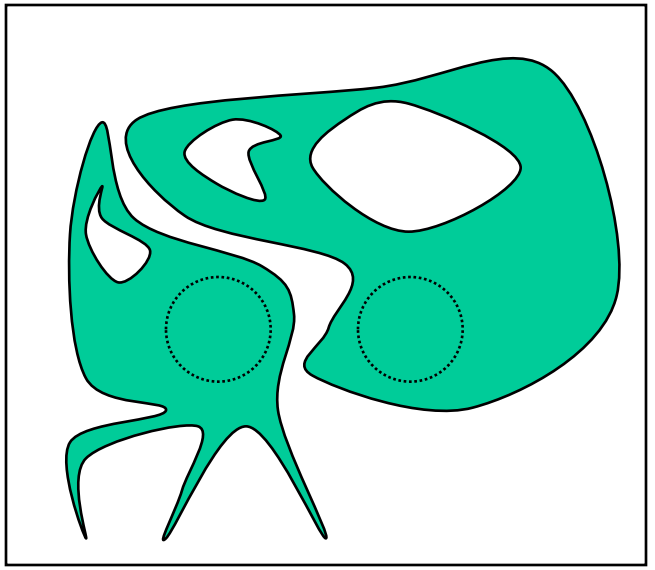


X



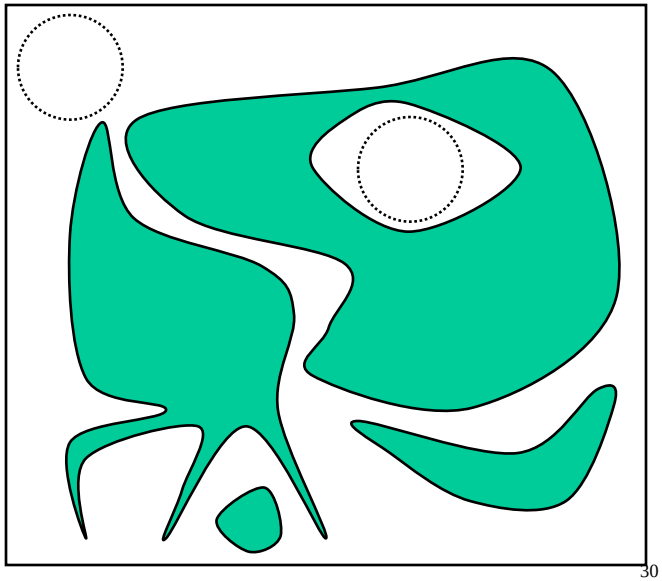
ouverture par reconstruction

$$E^{X}\left(\gamma_{B}(X)\right)$$



fermeture par reconstruction

$$\left(E^{X^{c}}\left(\left(\varphi_{B}(X)\right)^{c}\right)\right)^{c}$$

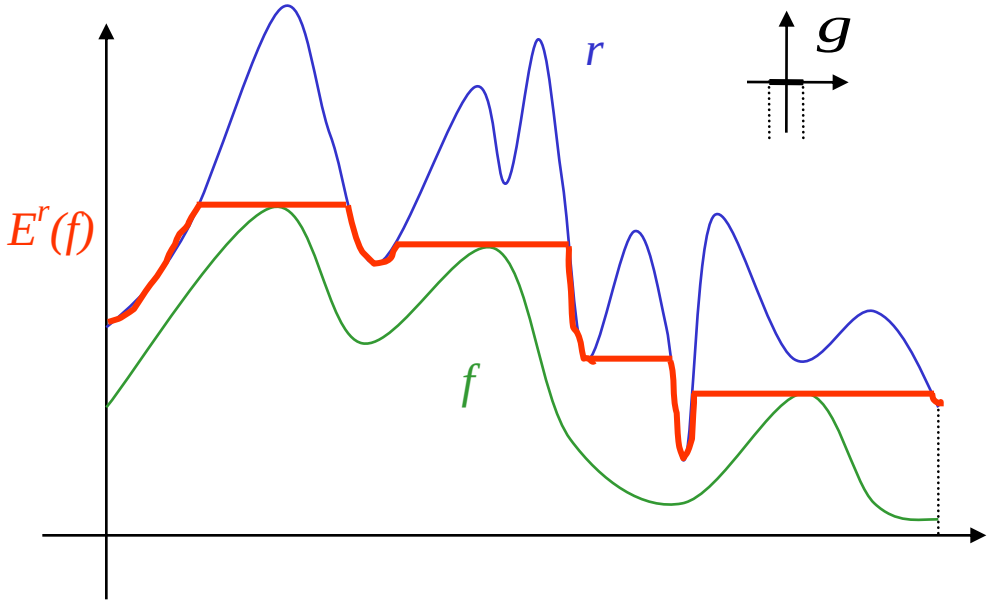


La fermeture par reconstruction est définie par dualité :

Reconstruction fonctionnelle

La dilatation géodésique de f dans r :

$$\delta_g^r(f) = \delta_g(f) \wedge r$$



La reconstruction géodésique de f dans r :

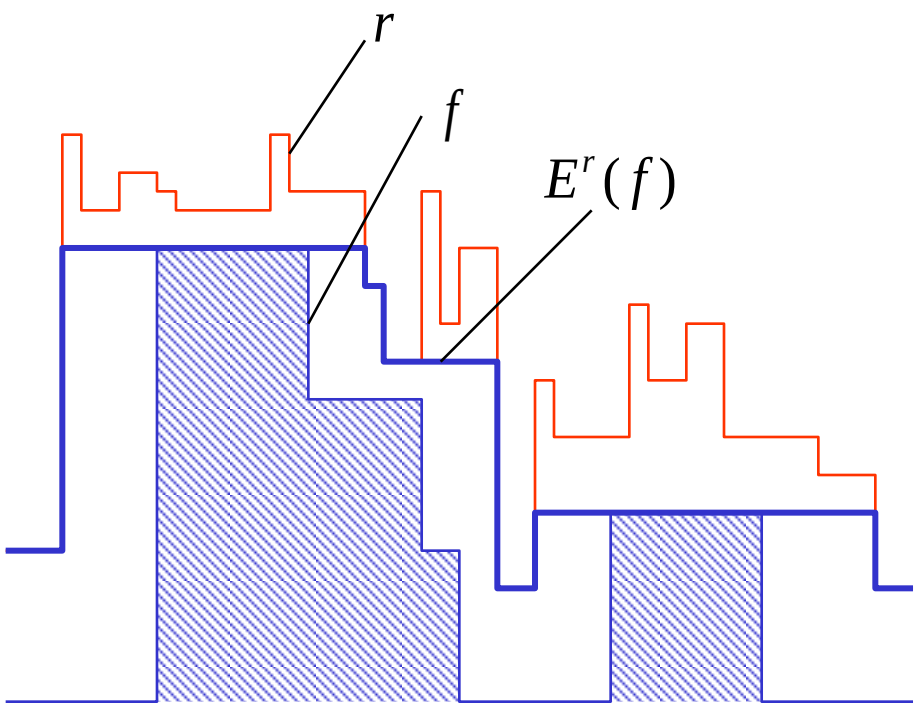
$$E_g^r(f) = \sup_{n \geq 0} \left\{ (\delta_g^r)^n (f) \right\}$$

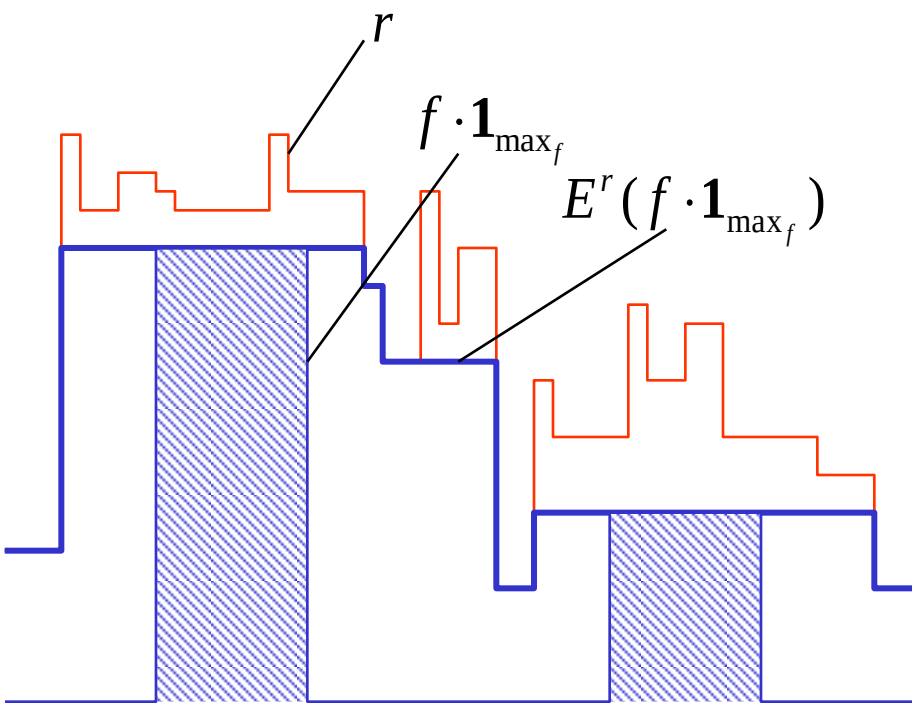
Reconstruction numérique à base de FIFO

Dans le cas de la reconstruction numérique (fonctionnelle), l'utilisation des FIFO est moins immédiate car il faut déterminer le domaine de stabilité (ensemble des points fixes) de la fonction marqueur *f*, au bord duquel la propagation va être initialisée. Ce domaine de stabilité est en fait l'ensemble des *maxima régionaux* de *f*. On utilise alors la propriété suivante :

La reconstruction de *f* est la même que la reconstruction de la restriction de *f* à ses maxima locaux :

$$E^r(f) = E^r(f \cdot \mathbf{1}_{\max_f})$$

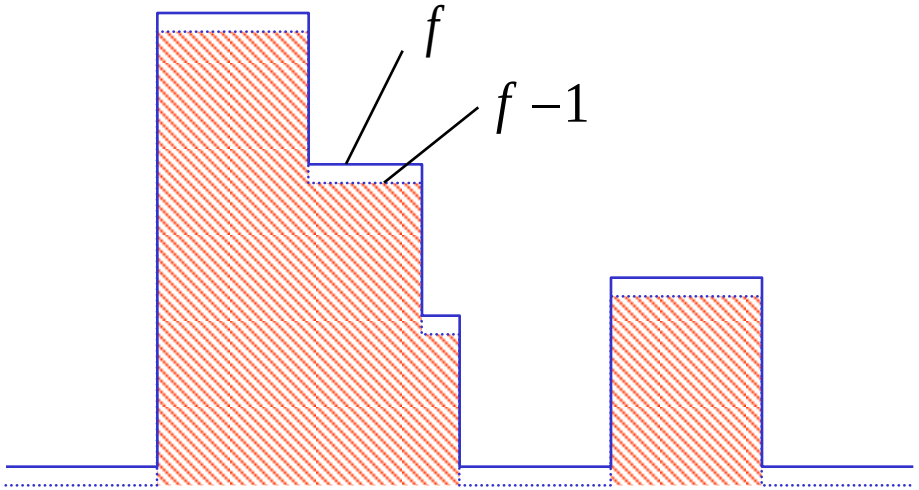




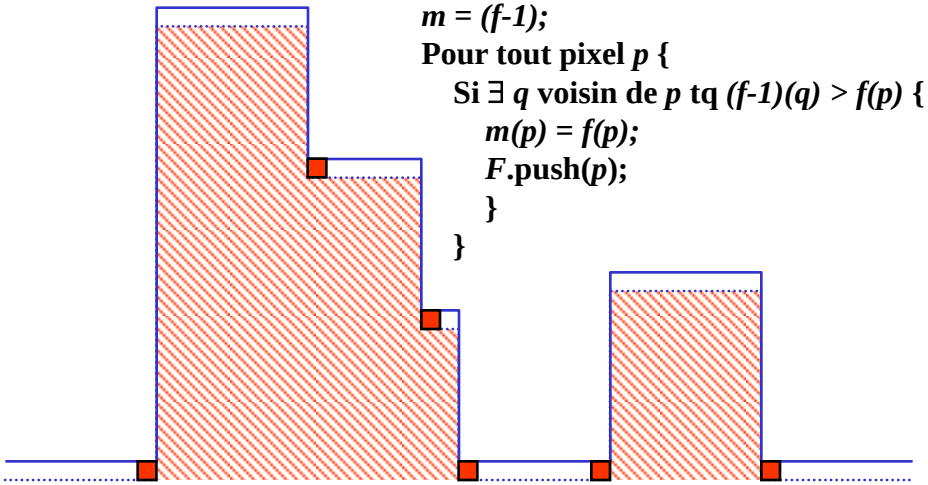
Reconstruction numérique à base de FIFO

(1) La première étape consiste donc à calculer les maxima régionaux de *f* :

Pour cela, on reconstruit *f-1* sous *f* :

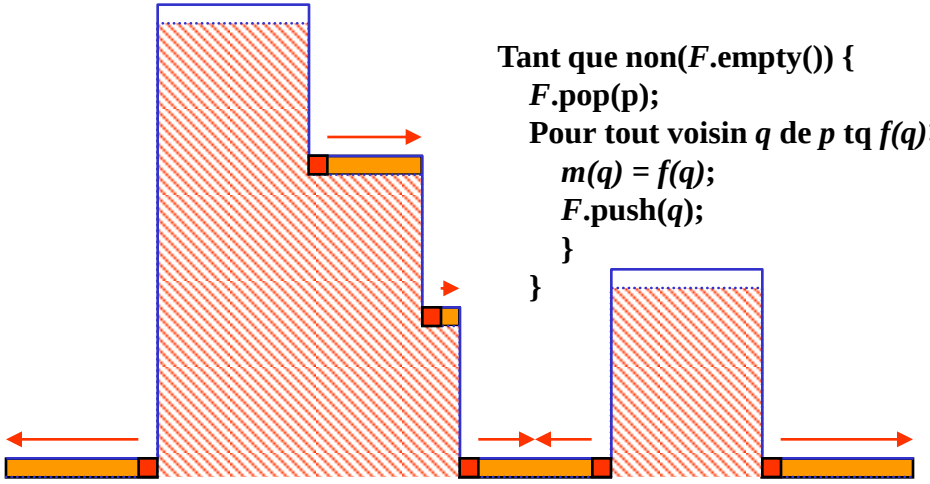


Initialisation de la FIFO :



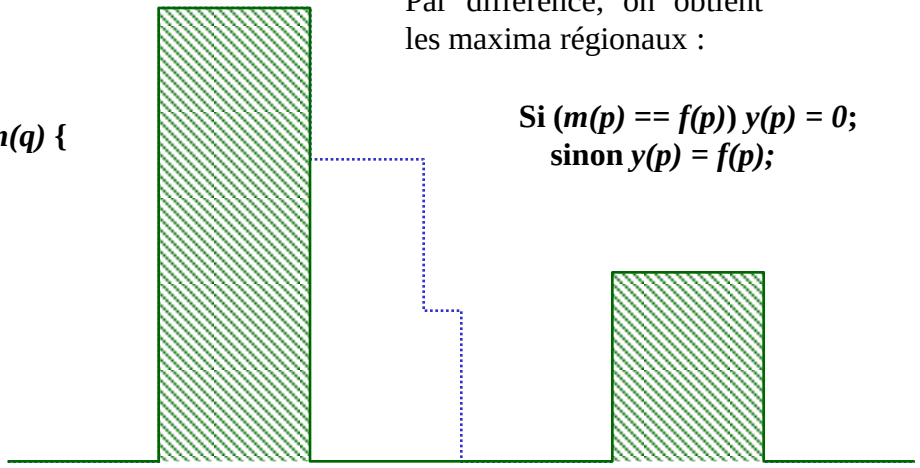
```
m = (f-1);
Pour tout pixel p {
   Si ∃ q voisin de p tq (f-1)(q) > f(p) {
      m(p) = f(p);
      F.push(p);
      }
}
```

Propagation de la FIFO :



```
Tant que non(F.empty()) {
   F.pop(p);
   Pour tout voisin q de p tq f(q)>m(q) {
      m(q) = f(q);
      F.push(q);
      }
}
```

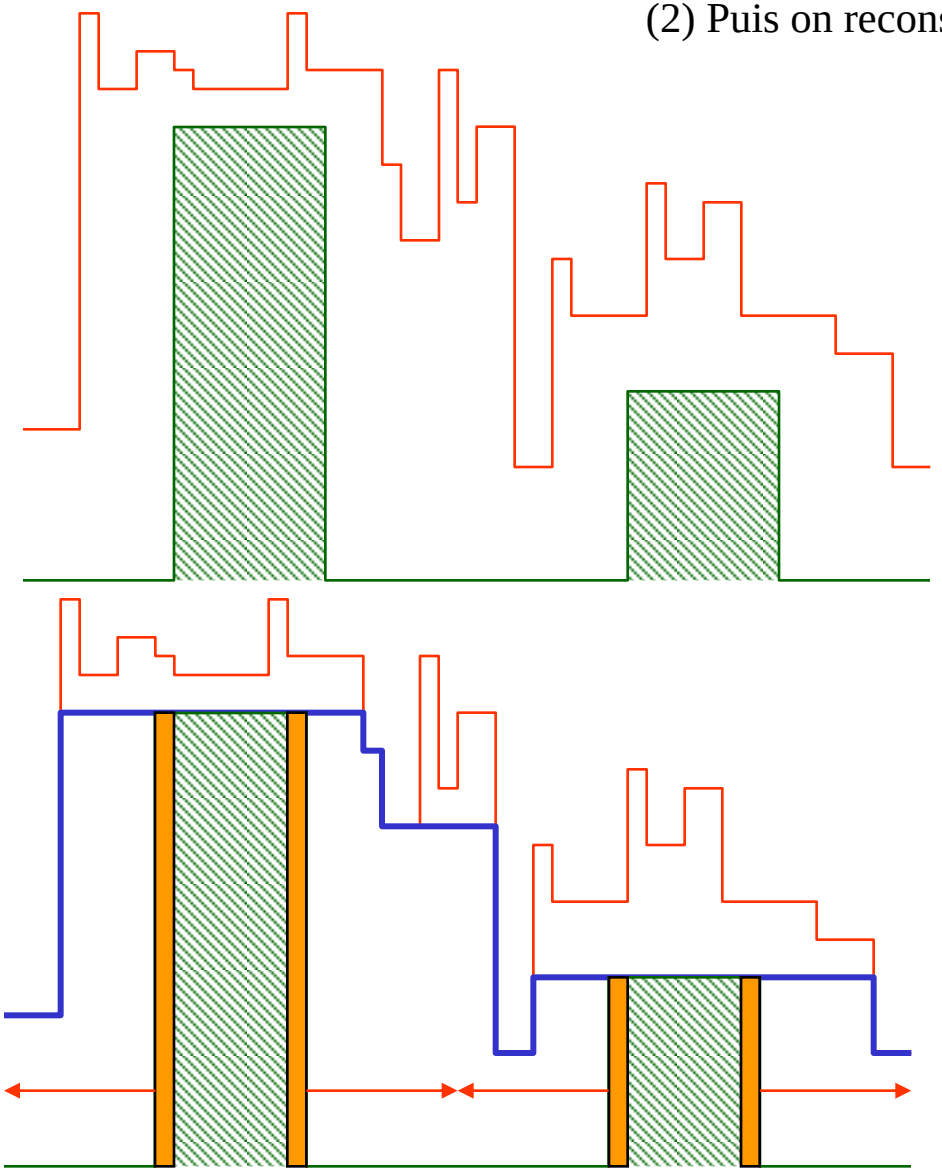
Par différence, on obtient les maxima régionaux :



```
Si (m(p) == f(p)) y(p) = 0;
   sinon y(p) = f(p);
```

Reconstruction numérique à base de FIFO

(2) Puis on reconstruit $\max_f$ sous r :



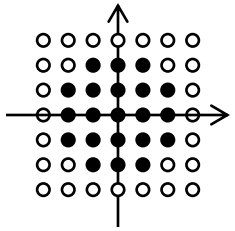
```
// Initialisation FIFO
m = max_f;
Pour tout pixel p tq
  (m(p) ≠ 0) et (∃ q voisin de p tq m(q) = 0) {
    F.push(p);
    }
// Propagation FIFO
Tant que non(F.empty()) {
   F.pop(p);
   Pour tout voisin q de p tq m(p)>m(q) {
      m(q) = MIN(f(q),m(p));
      F.push(q);
      }
   }
```

Le coût de calcul de la reconstruction numérique par FIFO est donc obtenue par un nombre constant de parcours d'images : 2 balayages complets pour les initialisations de FIFO, et 2 parcours de FIFO où les points ne sont examinés qu'une fois en général (2 ou 3 dans des cas extrêmes où 2 ou plusieurs maxima régionaux sont très proches).

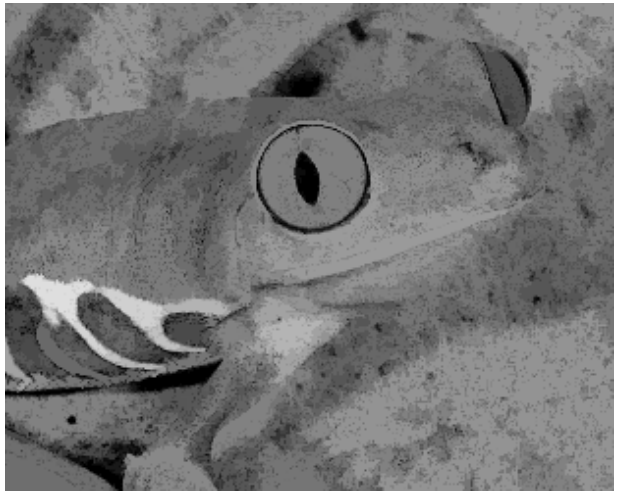
Ouvertures et fermetures par reconstruction

Par extension, les ouvertures et fermetures par reconstruction élimine les structures en préservant les contours des images numériques :

élément structurant de l'ouverture morphologique :



original



ouverture par reconstruction

fermeture par reconstruction

Transformées en distance et FIFO

Les files d'attente peuvent également être utilisées pour calculer la transformée en distance de manière efficace par propagation du contour. L'algorithme correspond alors à une application à la maille carrée de l'algorithme de Dijkstra. Tous les algorithmes récursifs présentés précédemment s'adaptent facilement à ce cadre. Par exemple, pour la distance d_4 :

```
% Initialisation FIFO
for i = 1:w
  for j = 1:h
    if (i,j)∈X
      if ∃(i',j'),d4((i,j),(i',j'))==1)&(i',j')∉X : {F(i,j) = 1; Q.push(i,j);}
      else : F(i,j) = ∞ ; endif
    endif
  endfor
endfor
% Parcours de la FIFO et étiquetage
while ~(Q.empty)
  Q.pop(i,j);
  forall (i',j'), (i',j')∈X,d4((i,j),(i',j'))==1) and F(i',j')==∞ : {F(i',j') = F(i,j)+1 ; Q.push(i',j');}
  endfor
endwhile
```

Algorithme de calcul de la transformée en distance d_4 par l'algorithme de Dijkstra avec FIFO.