



# Etude d'algorithmes d'inpainting

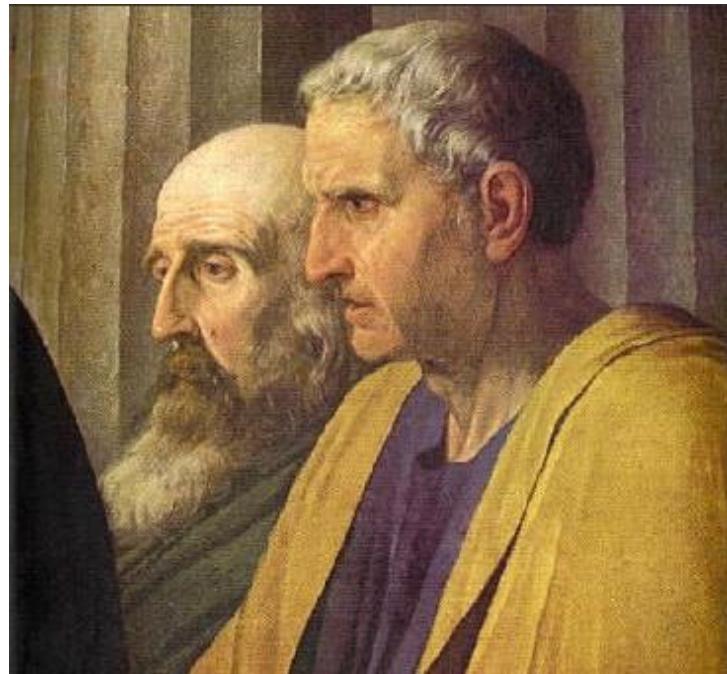
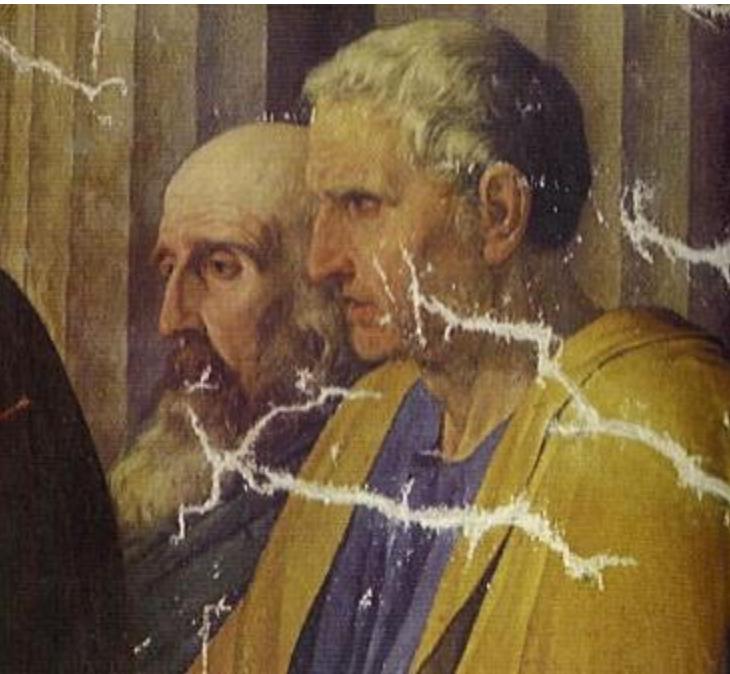
Nicolas Klarsfeld, Approfondissement Signal et Images

Stage réalisé sous la supervision d'Adrian Nachman, professeur de  
mathématiques à l'université de Toronto

# Plan

- Qu'est-ce que l'inpainting ?
- 1) Inpainting par diffusion hypoelliptique (Boscain et al., 2014)
  - La diffusion hypoelliptique
  - La restauration statique
  - Avantages et inconvénients de la méthode
- 2) Inpainting par « Diffusion maps » et relaxation spectrale (Gepshtein and Keller, 2013)
  - Diffusion maps
  - Algorithmes nécessaires pour la diffusion maps
    - Local Binary Patterns
    - Heat equation inpainting
    - Best exemplar inpainting
    - Patch Match
  - Relaxation spectrale
- Conclusion

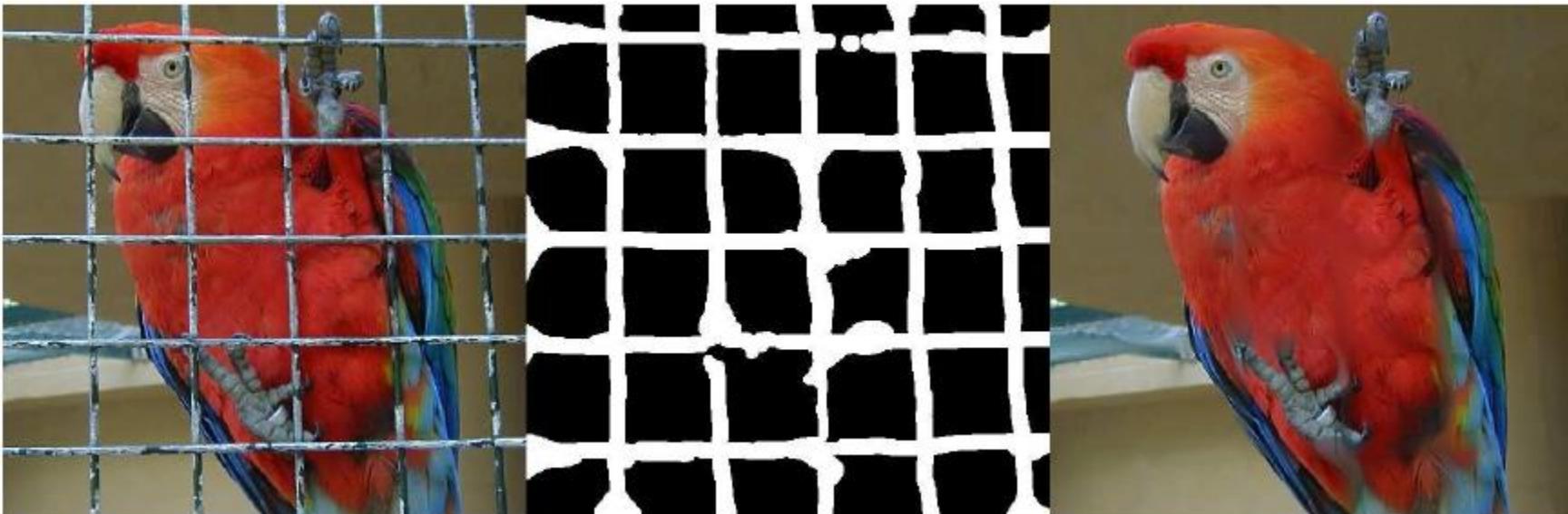
# Qu'est-ce que l'inpainting ?



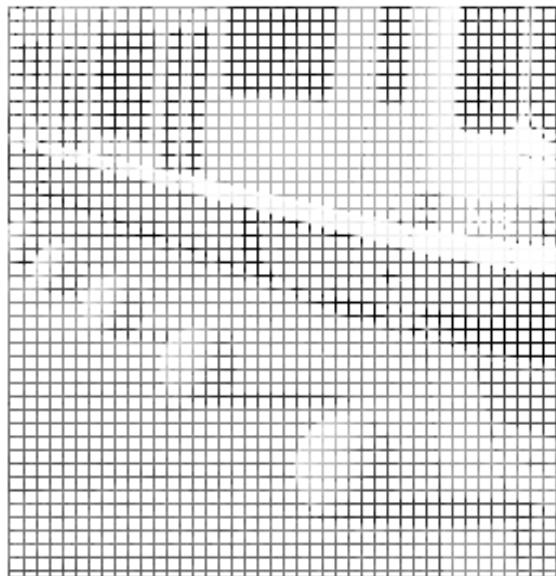
# 2 types de méthodes

- Les méthodes géométriques (aussi appelées non locales)
- Les méthodes non locales (utilisant des patches)
- Nous allons faire une rapide revue de littérature des deux types de méthodes, puis nous étudierons une méthode appartenant à chacune des classes de méthodes : une méthode locale, puis une méthode non locale.

# Méthodes géométriques, 1 : Tschumperlé



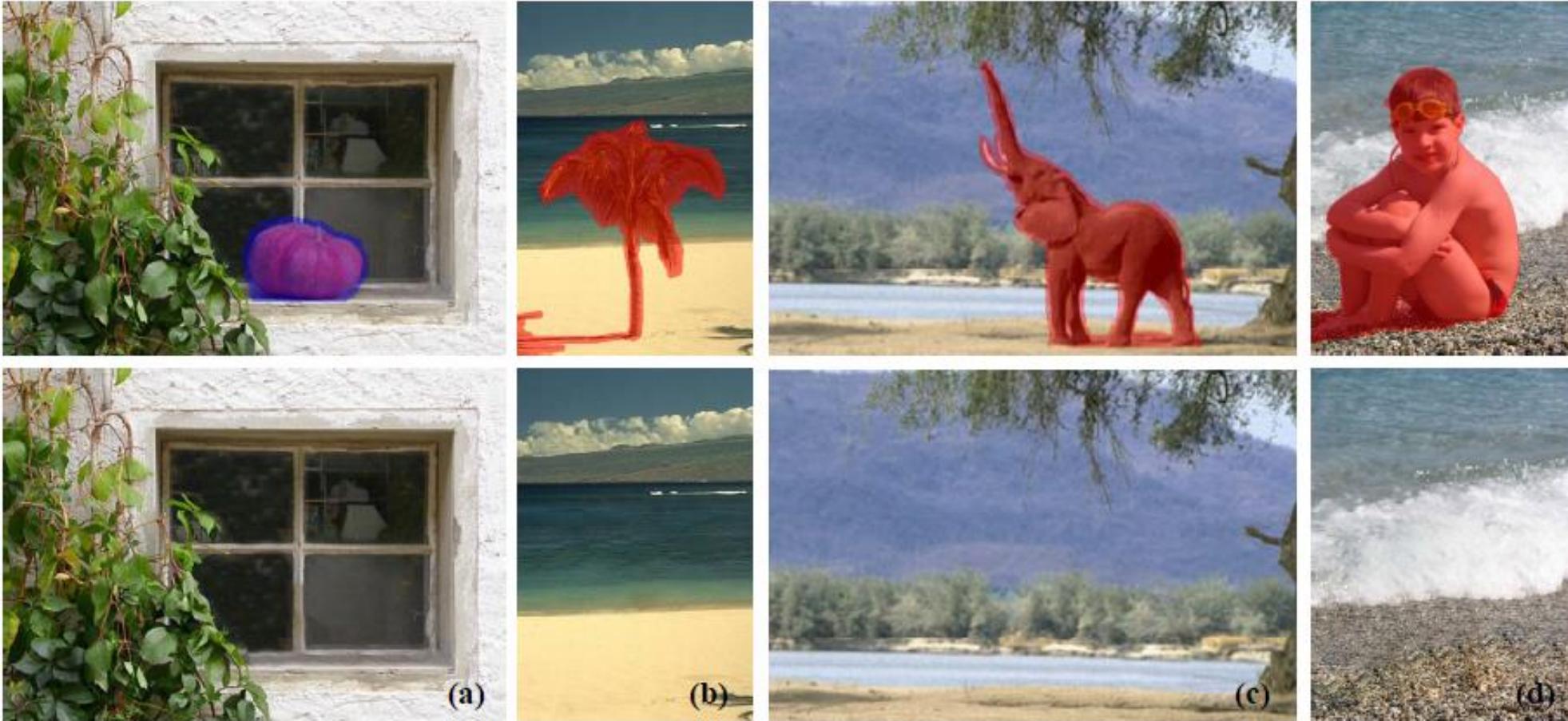
# Méthodes géométriques, 2 : Masnou



# Méthodes non locales, 1 : Komodakis



# Méthodes non locales, 2 : He and Sun



# Méthodes non locales, 3 : Drori et al.

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought Alice 'without pictures or conversation?' So she was considering in her own mind (well as she could, for the hot day made her feel very sleepy and stupid), whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her. There was nothing so VERY remarkable in that, nor did Alice think it so VERY much out of the way to hear the Rabbit say to itself, 'Oh dear! Oh dear! I shall be late!' (when she thought it over afterwards, it occurred to her that she ought to have won at this, but at the time it all seemed quite natural); but when the Rabbit actually TOOK A WATCH OUT OF ITS WAISTCOAT-POCKET, and looked at it, and then started to run away, for it flashed across her mind that she had never before seen a rabbit with either a waistcoat-pocket, or a watch to take out of it, and burning with curiosity, she ran across the field after it, and fortunately was just in time to see it pop down a large rabbit-hole under the hedge. In another moment down went Alice after it, never once considering how in the world she was to get out again. The rabbit-hole went straight on like a tunnel for some way, and then dipped suddenly down, so suddenly that Alice had not a moment to think about stopping herself before she found herself falling down a very deep well. Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next. First, she tried to look down and make out what she was coming to, but it was too dark to see anything; then she looked at the sides of the well, and noticed that they were filled with cupboards and book-shelves.



# Différences entre méthodes locales et non locales

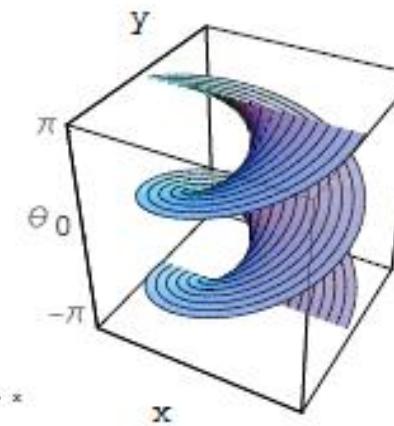
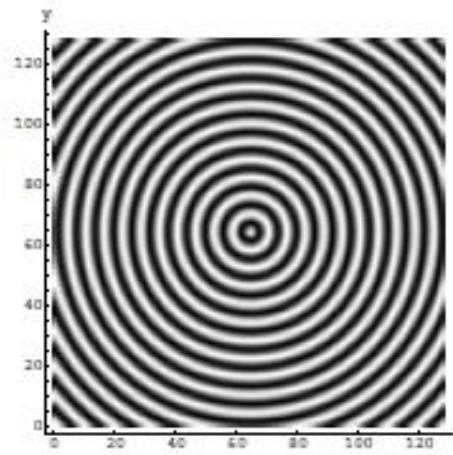
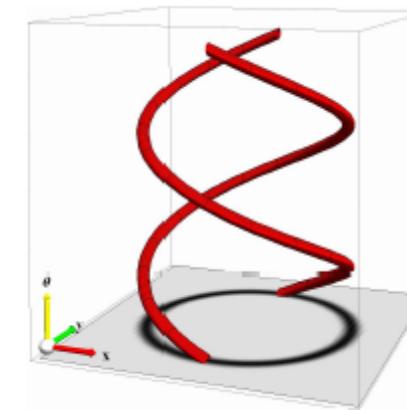
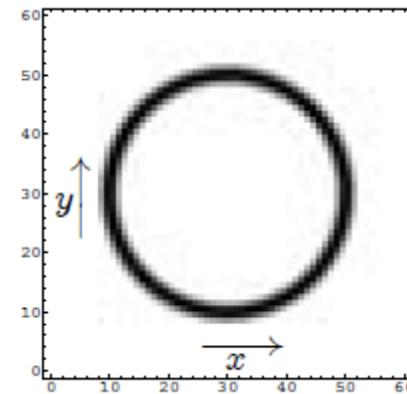
- Méthodes locales :
  - Plus rapides
  - Ne permettent pas de reconstituer les textures
  - Ont parfois du mal à inpainter les trous de grande taille
- Méthodes non locales :
  - Plus longues
  - Permettent de reconstituer les régions texturées de l'image
  - Certaines de ces méthodes permettent de reconstituer des trous de grande taille.

# Partie 1: Inpainting par diffusion hypoelliptique (Boscain et al., 2014)

- Diffusion hypoelliptique : 3 étapes :
  - 1) Lifter l'image dans un tableau à 3 dimensions (dimension supplémentaire en plus de x et y → angle local de l'image, discrétisé en 30 valeurs réparties entre 0° et 180° par pas de 6°).
  - 2) Faire évoluer les intensités suivant l'EDP suivante :
- 3) Projection des valeurs du tenseur 3D sur les axes de l'image

$$\frac{df}{dt} = \frac{1}{2} \left( a \cdot \frac{\partial^2 f}{\partial \theta^2} + b \cdot \left( \cos(\theta) \cdot \frac{\partial}{\partial x} + \sin(\theta) \cdot \frac{\partial}{\partial y} \right)^2 \circ f \right)$$

# Lift et projection : intuition

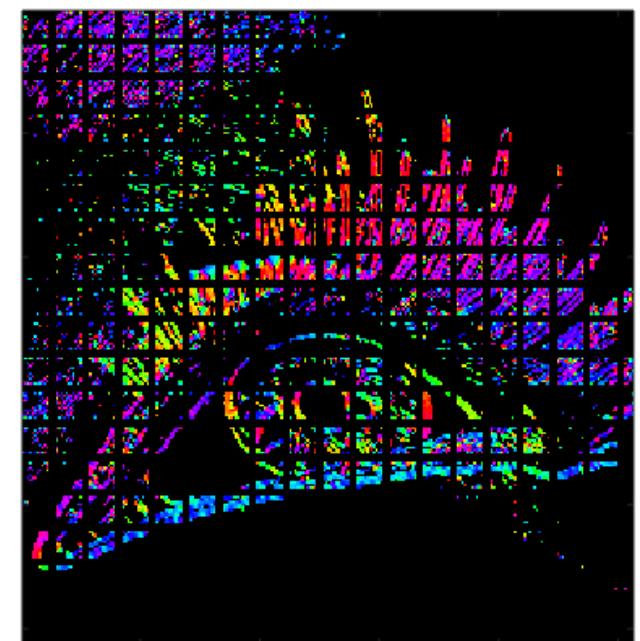
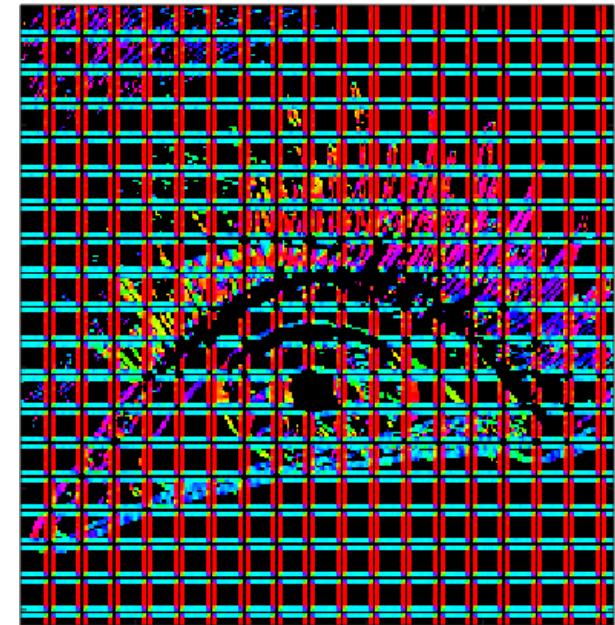
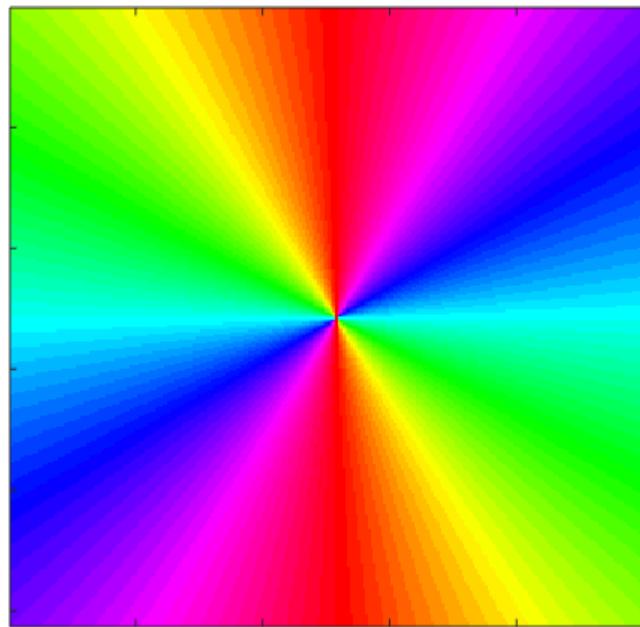
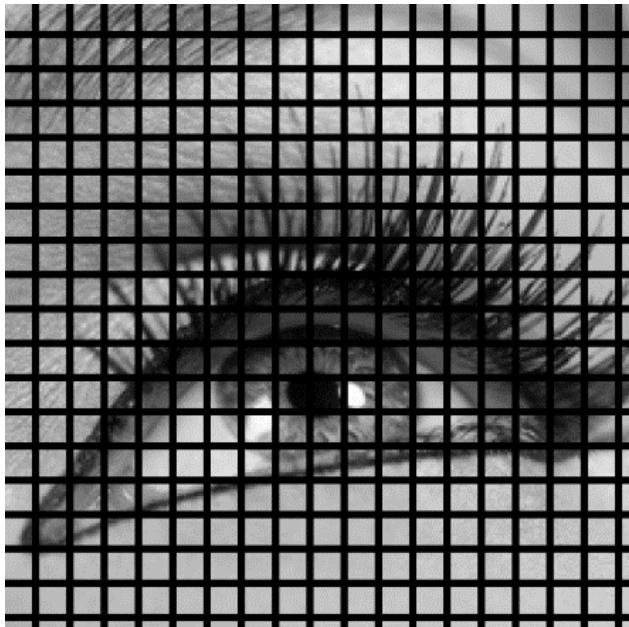


# Hypoelliptic diffusion : intuition



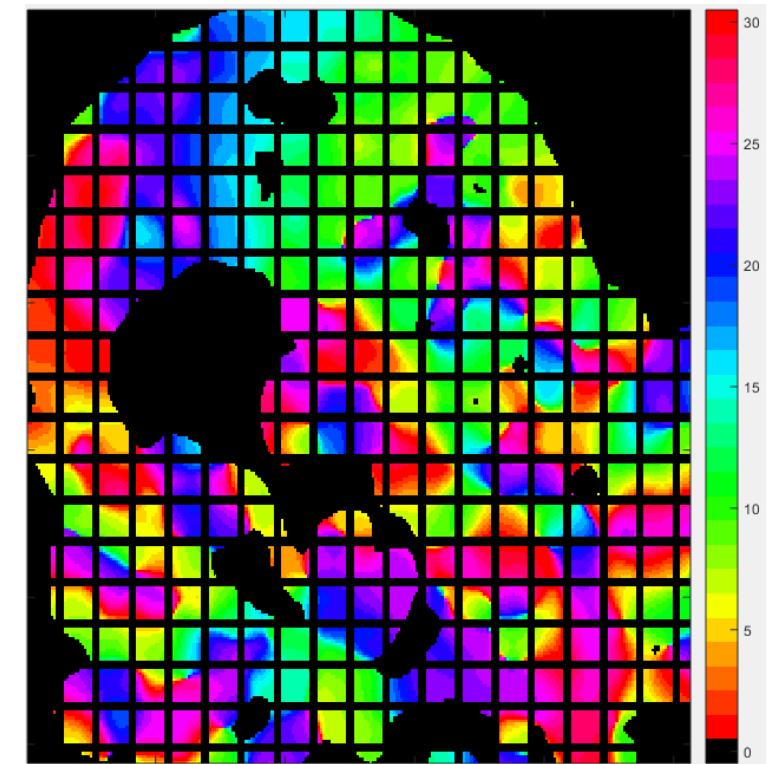
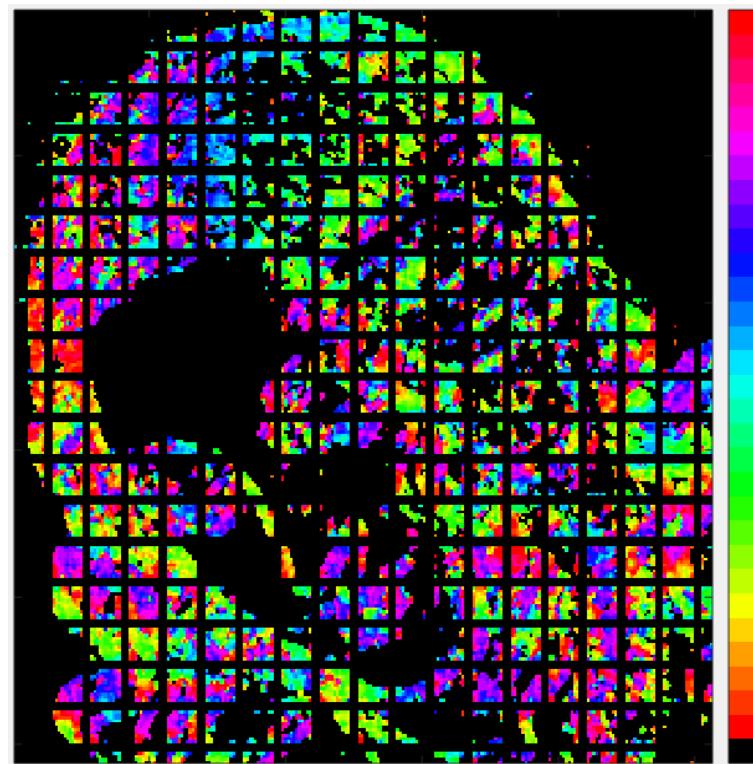
# Lift : calcul des angles locaux (1)

Importance d'éviter les bords dans la discréétisation du gradient



# Lift : calcul des angles locaux (2)

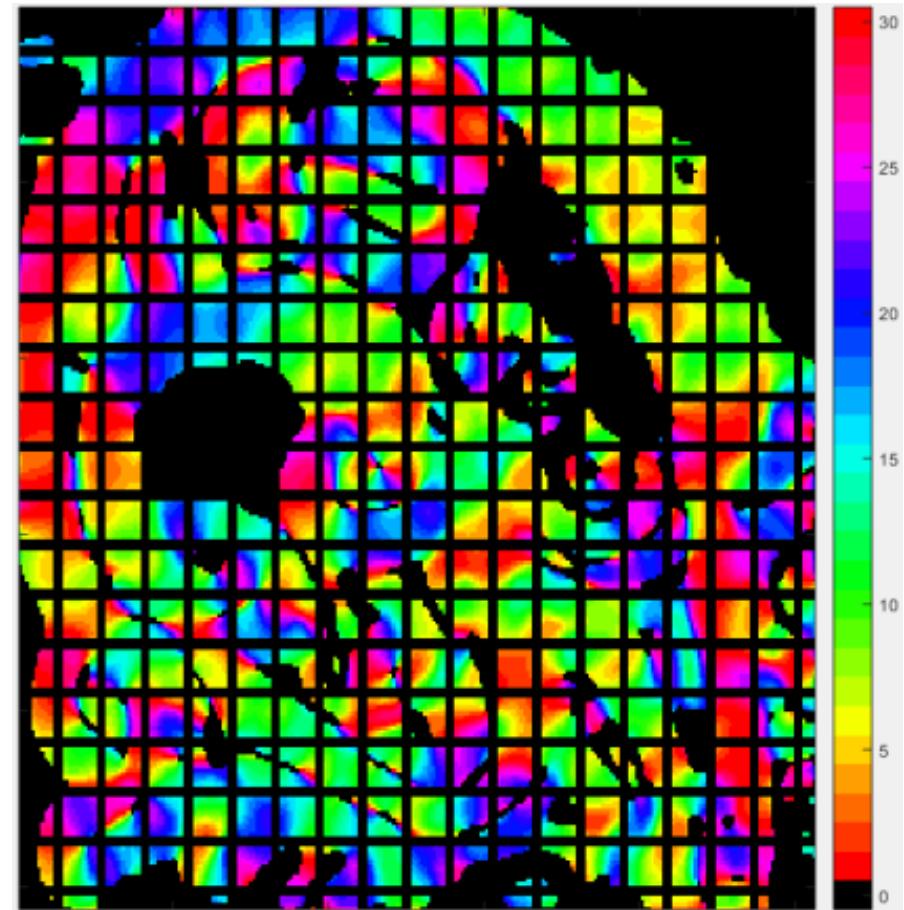
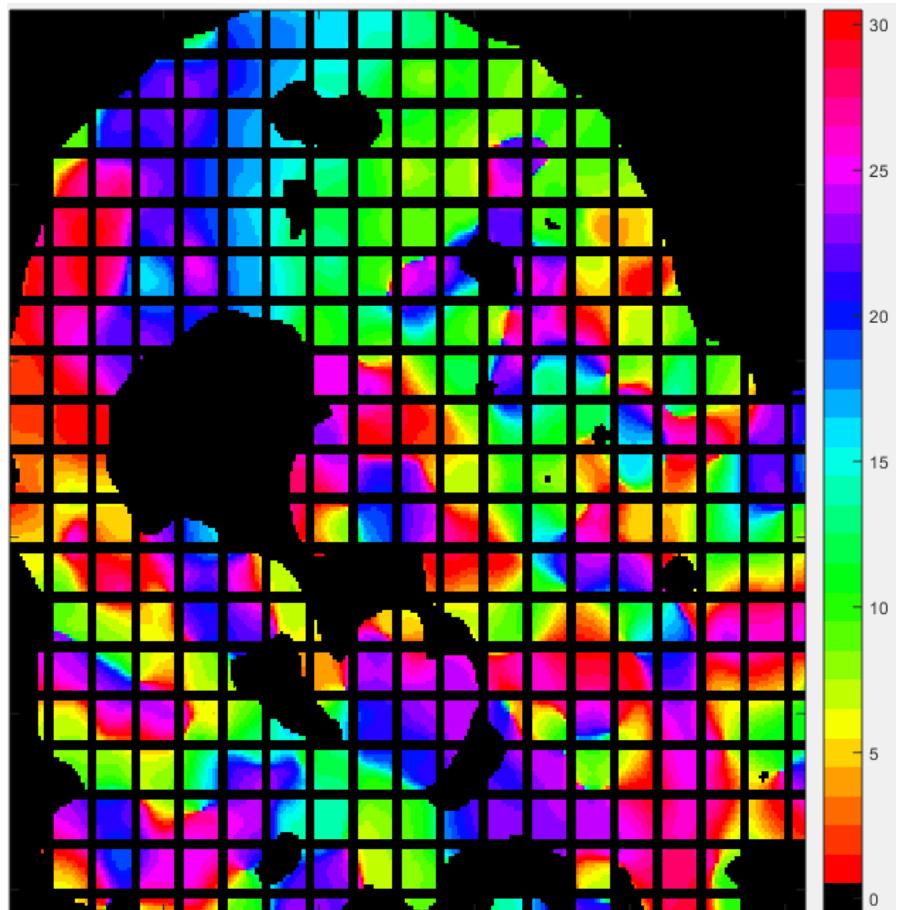
Calcul de l'angle local : utilisation du tenseur de structure (différentes échelles possibles)



Slider interactif pour régler le lift (le montrer  
dans Matlab)

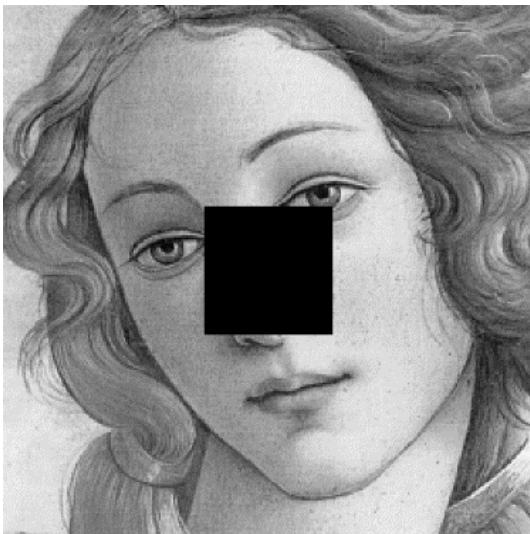
# Lift : calcul des angles locaux (3)

Robustesse du tenseur de structure par rapport au gradient de l'image floutée



# Lift : calcul des angles locaux (4)

- Importance de compenser le flou près des bords du trou (sinon création d'un gradient du trou vers l'image)



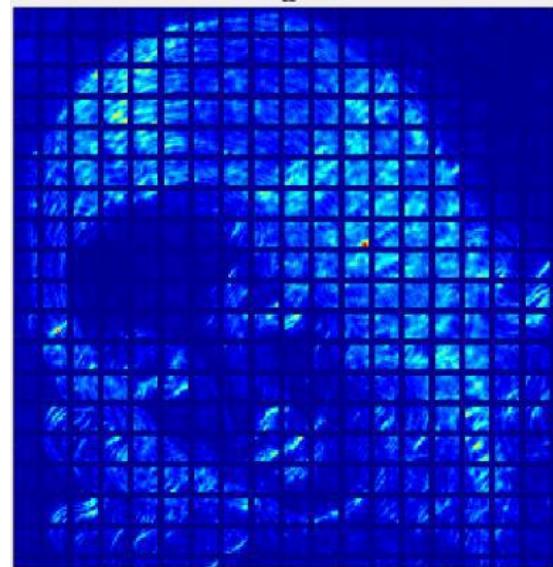
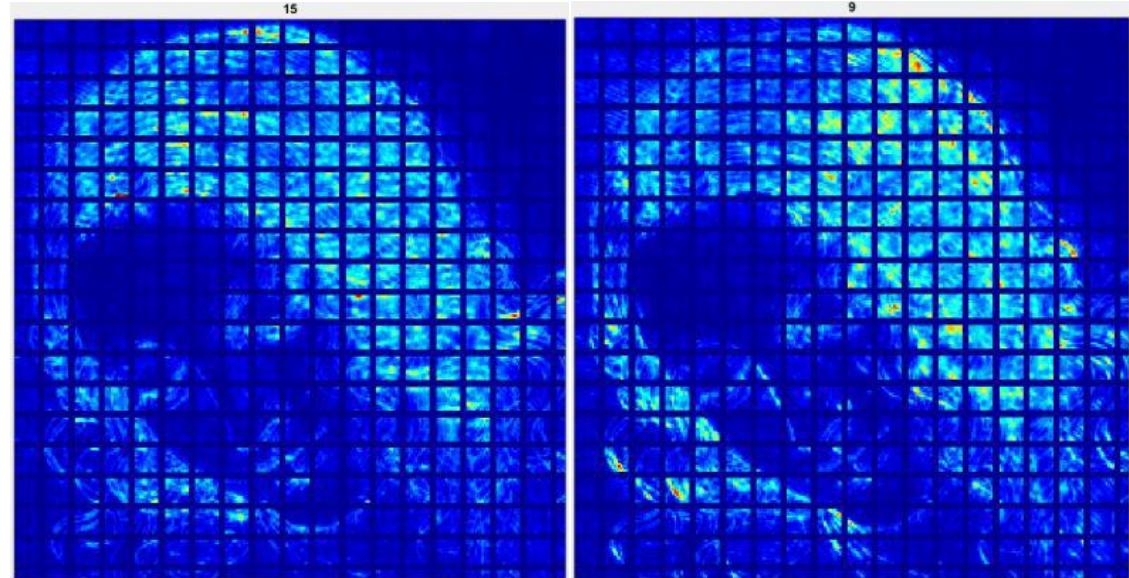
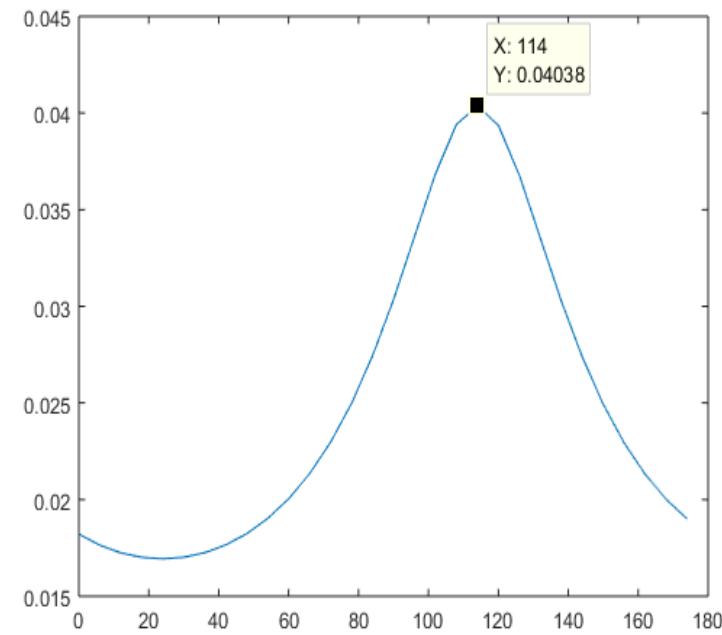
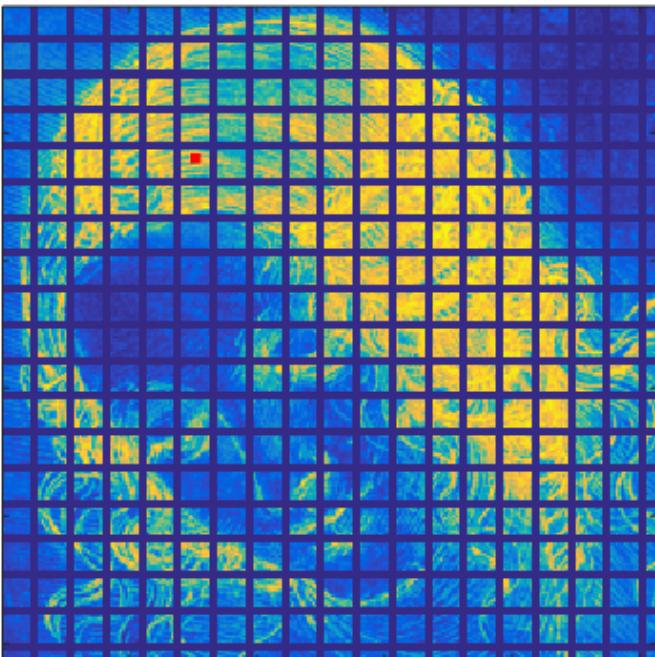
# Lift : formule du lift

- Une fois que l'angle est calculé, on peut utiliser plusieurs formules pour le lift
- Lift « Tout ou rien » (plus proche angle parmi les 30 angles possibles)



# Lift : formule du lift

- Lift plus progressif :
  - $(\cos(\theta - \theta_0))^n$  avec n croissant avec la directionnalité de l'image, qu'on peut mesurer avec la norme du gradient par exemple.

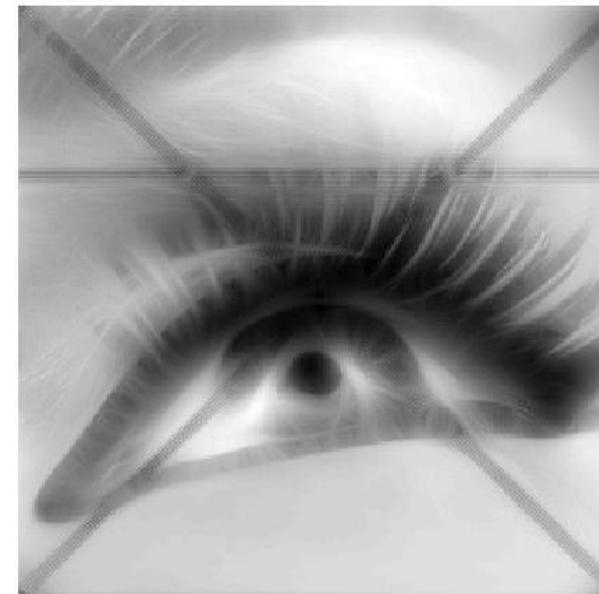


# EDP de diffusion hypoelliptique

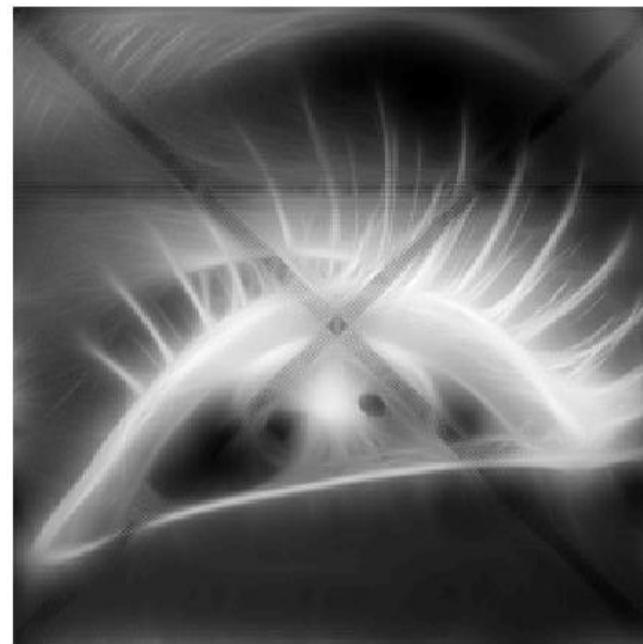
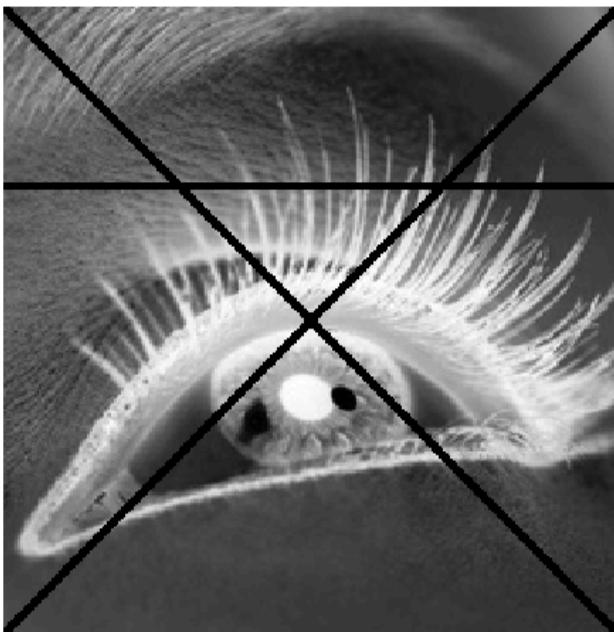
- $\frac{df}{dt} = \frac{1}{2} \left( a \cdot \frac{\partial^2 f}{\partial \theta^2} + b \cdot \left( \cos(\theta) \cdot \frac{\partial}{\partial x} + \sin(\theta) \cdot \frac{\partial}{\partial y} \right)^2 \circ f \right)$ 
  - 1 équation à  $Mx * My * 30$  inconnues (forward Euler)
- $Fourier \left( \frac{df}{dx} \right)(k) = Fourier \left( \frac{f(x+dx) - f(x-dx)}{2 \cdot dx} \right)(k) = \frac{1}{dx} \cdot i \cdot \sin \left( \frac{2\pi(k-1)}{M} \right) \cdot Fourier(f)$
- $Mx * My$  équations à 30 inconnues :
  - Cranck Nicholson (schéma implicite et inconditionnellement stable)
  - Ou résolution exacte (exponentielles de matrices)

# Projection

- Importance de la renormalisation, de sorte à avoir un lift tel que  $\text{projection}(\text{lift})=\text{identité}$  (c'est la forme du lift qui compte)
- Proposé par les auteurs : moyenne ( $\approx \text{sum}$ ) ou max.
- Inconvénient de max : ne fait apparaître que les structures lumineuses



- Prendre le minimum envoie tout vers zéro.
- Solution : inversion de la luminosité. Les structures propagées doivent être lumineuses (limitation).

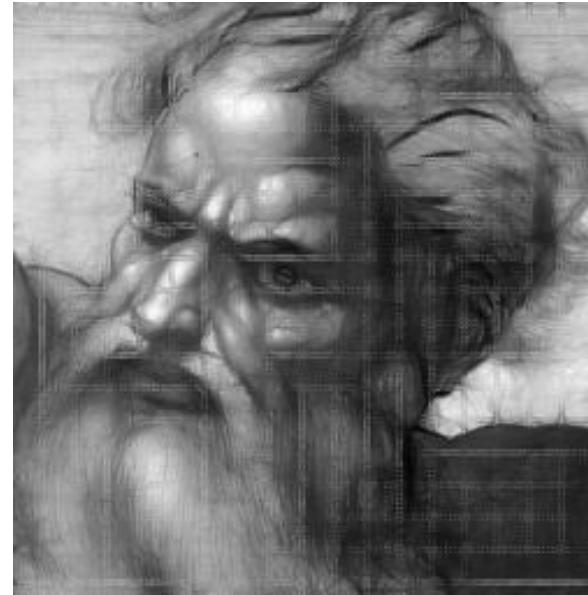


# Diffusion statique

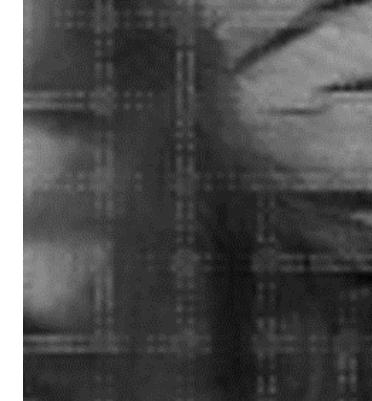
- Problème de la diffusion hypoelliptique simple : modifie aussi les pixels connus. Solution : diffusion statique.
- Pendant un petit temps  $dt$ , on utilise la diffusion hypoelliptique. Puis  $\forall(x, y)$  connu, on multiplie la colonne  $f(x, y, \theta)$  de façon à ce avoir  $\text{projection}(f(x, y, \theta)) = f_0(x, y, \theta)$ . Puis on réitère la séquence diffusion/réinitialisation des colonnes au dessus des pixels connus.

# Résultats

- Problème :  
Artefacts



# Enlever les artefacts



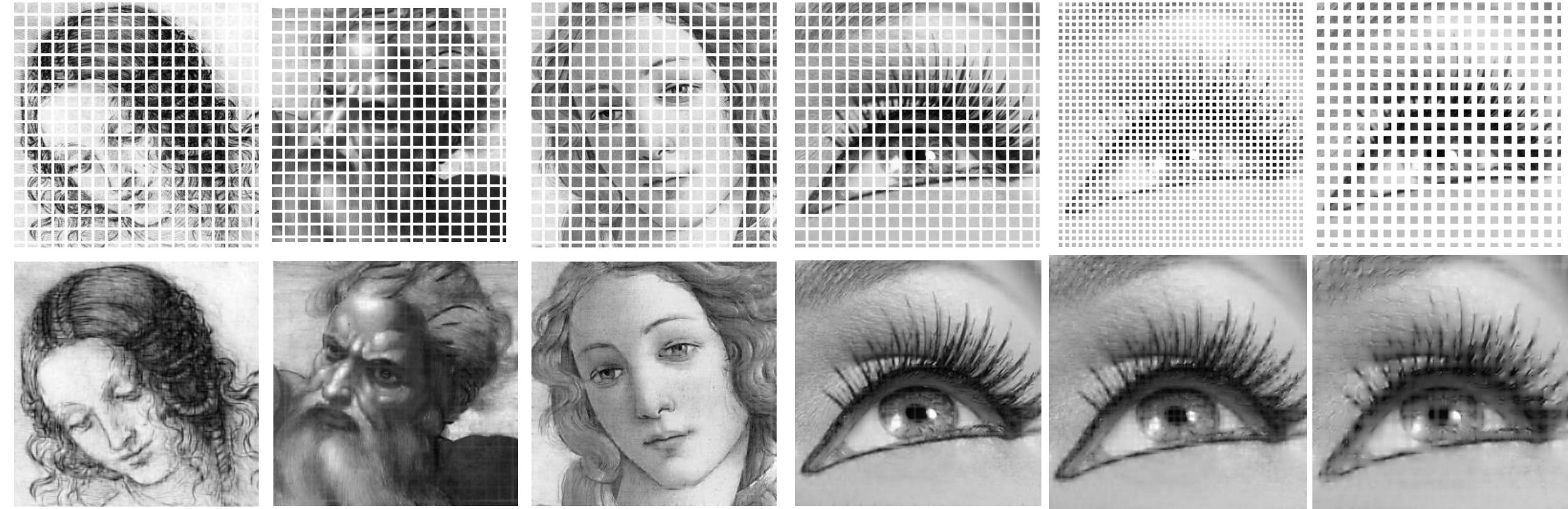
On utilise pas le schéma centré pour les dérivées partielles d'ordre 1 car sinon les dérivées d'ordre 2 utilisent le schéma faisant un ‘saut’ :

$$\frac{d^2f}{dx^2} = f(x + 2) - 2.f(x) + f(x - 2)$$

On utilise donc le schéma normal :

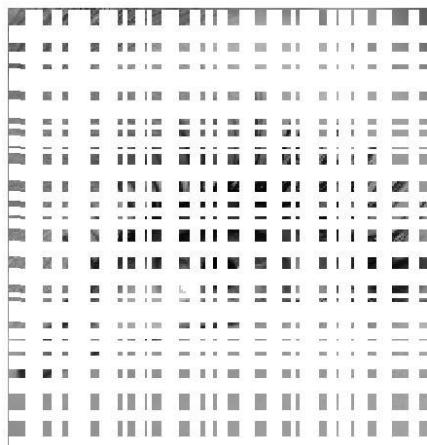
$$\frac{d^2f}{dx^2} = f(x + 1) - 2.f(x) + f(x - 1)$$

# Résultats finaux



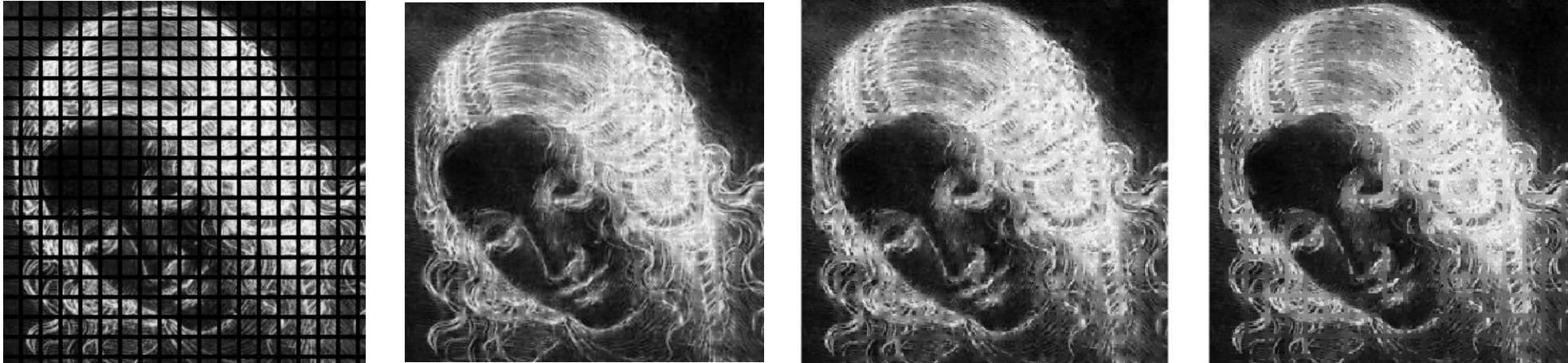
# Avantages et inconvénients de la méthode

- Avantages :
  - si on affine la méthode, elle permet d'avoir des résultats qui sont au niveau de l'état de l'art



- Inconvénients : tel quel, elle ne donne pas des résultats significativement meilleurs que des méthodes beaucoup plus rapides (comme l'équation de la chaleur, typiquement 20 fois plus rapide). Pour éviter cette lenteur on pourrait utiliser la parallélisation des calculs (ex : GPGPU avec Cuda ou OpenCL).

# Comparaison avec des méthodes géométriques plus rapides



- De gauche à droite : l'image masquée, l'inpainting avec la restauration statique utilisant la diffusion hypoelliptique, l'inpainting utilisant l'équation de la chaleur (utilisant notre code), l'inpainting utilisant la minimisation de la variation totale (utilisant la bibliothèque ‘tvreg’). Les résultats sont de qualités semblables (PSNR=22 environ, contre 10.5 pour l'image masquée), mais les temps d'execution sont de 1 à 2 minutes contre 20 minutes pour la restauration statique.

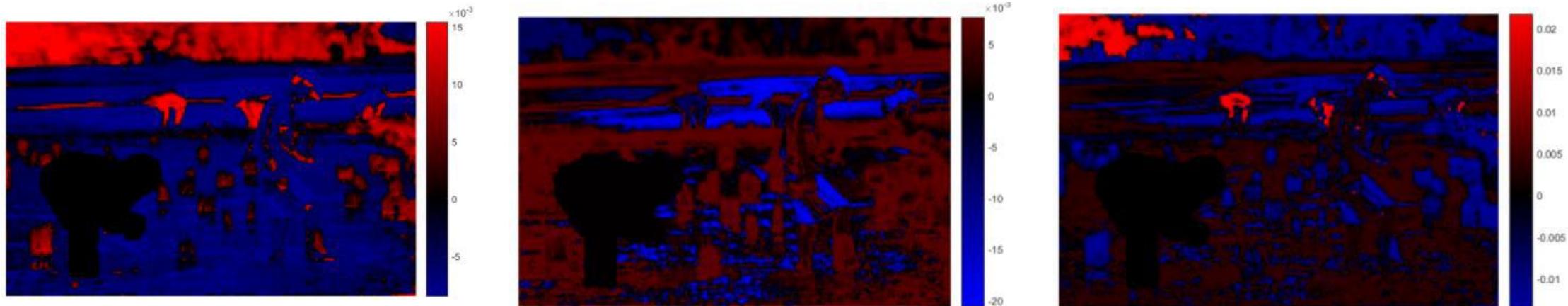
# Partie 2 : Inpainting par « Diffusion maps » et relaxation spectrale

- 9 étapes (les étapes 1 à 5 sont la ‘diffusion maps’).
  - 1) Choix d’une représentation de la texture entourant les pixels (fonction associant à un couple  $(x,y)$ , un vecteur représentant la texture entourant ce pixel). Exemples : patchs carrés, local binary patterns.
  - 2) Construction de la matrice (sparse) des distances entre patchs (KNN) : utilisation de (generalized PatchMatch).
  - 3) Construction de la matrice de corrélation entre les patchs
  - 4) Normalisation de la matrice de corrélation
  - 5) Diagonalisation : chaque eigenvector peut être représenté comme une image
  - 6) Inpainting simple (exemples : Criminisi, équation de la chaleur)
  - 7) KNN des pixels inconnus vers les pixels connus (utilisation de generalized PatchMatch)
  - 8) Relaxation spectrale
  - 9) Récupération des intensités des pixels par moyennage

# Diffusion maps : intuition

- Représenter la texture de manière optimale par plusieurs images.

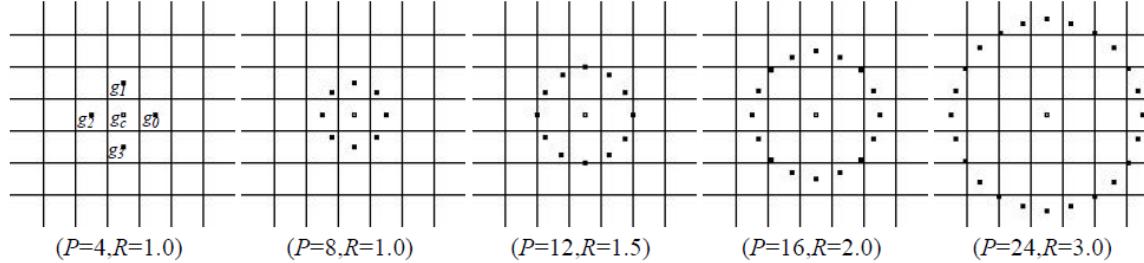
Ces images sont obtenues par ‘diffusion maps’ qui est une technique de réduction de dimension provenant de la théorie spectrale des graphes.



# Diffusion maps (étapes 1 à 5)

- Etape 1 : Représentation de chaque pixel par un vecteur représentant la texture de l'image autour de ce pixel.
- Exemples de tels vecteurs :
  - Composants RGB du pixel (patch de taille 1x1)
  - Composante RGB + les 2 composantes du gradient de l'image en niveau de gris (descripteur de Wexler)
  - Patch carré de taille  $(2n+1)^2$  autour du pixel
  - Patch non carré (par exemple rond), ou extrait par la multiplication avec une fenêtre non constante comme une fenêtre de Hanning
  - Autre descripteur de texture tel que les Local Binary Pattern (LBP) Histograms

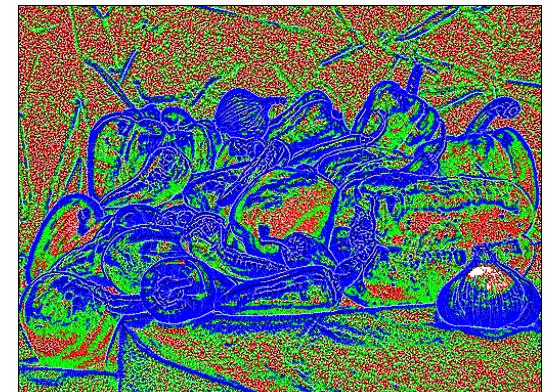
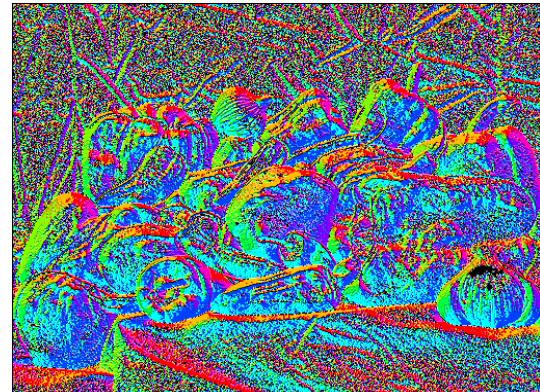
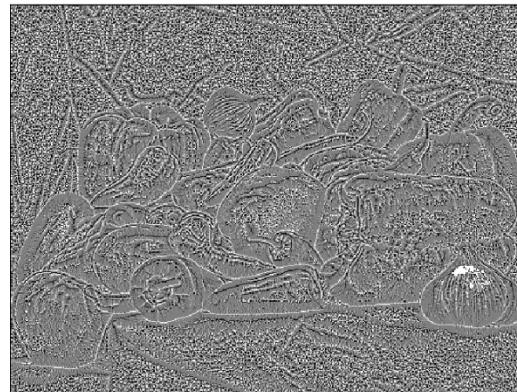
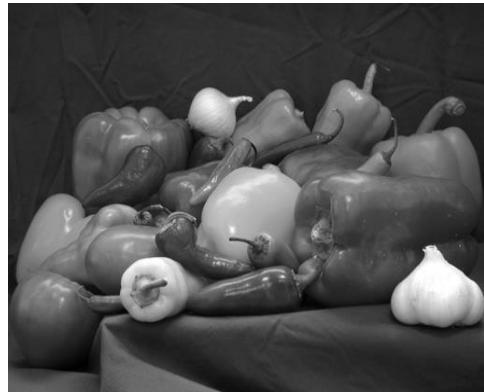
# Pointwise LBP



- On extrapole les valeurs des points situés sur un cercle autour du pixel. Le rayon  $R$  et le nombre de point  $N$  sont au choix. Le type d'extrapolation sera par exemple l'extrapolation bilinéaire ou trilinéaire.
- On associe au pixel une chaîne de 0 et de 1 construite en regardant pour chaque point du cercle (le premier point et le sens de rotation sont au choix) si la valeur extrapolée est supérieure (on associe alors un 1) ou inférieure (on associe un 0) à la valeur du pixel central.
- Exemple : Si le pixel central vaut 18, et que les pixels du cercle valent 23, 10, 35, 91, 2, 38, 7, 15, alors la chaîne est 1, 0, 1, 1, 0, 1, 0, 0

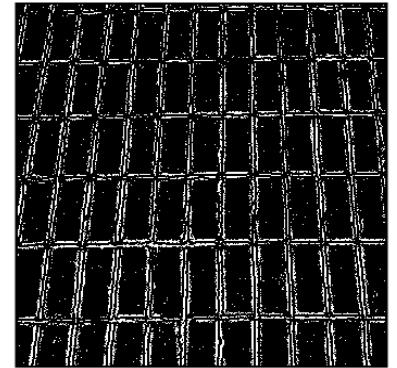
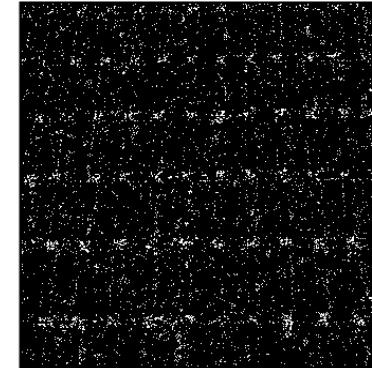
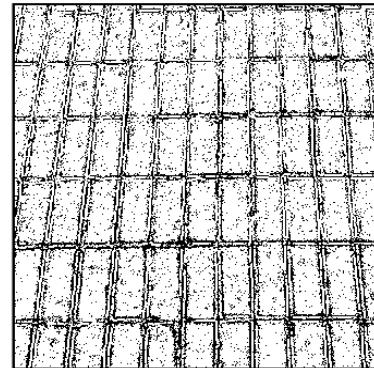
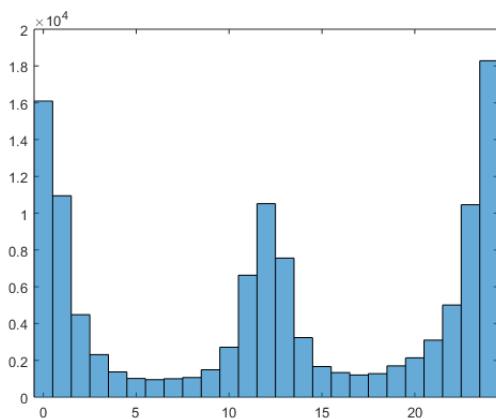
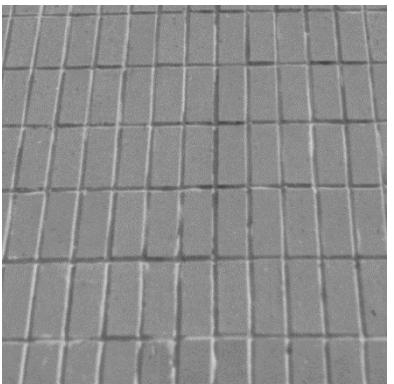
# Pointwise LBP (suite)

- On associe ensuite au pixel un nombre dépendant de cette chaîne de 0 et de 1. Le LBP classique est de prendre le nombre décrit en binaire par la chaîne de nombre. Il y a alors  $2^N$  valeurs possibles. Une autre façon (appelée rotationnal invariant LBP) est de compter le nombre de 1 dans la suite de nombres. Il y a alors  $N+1$  valeurs possibles.
- Résultats pour 3 types de pointwise LBP (rotationnal invariant, uniform, et rotationnal invariant avec la variance) :



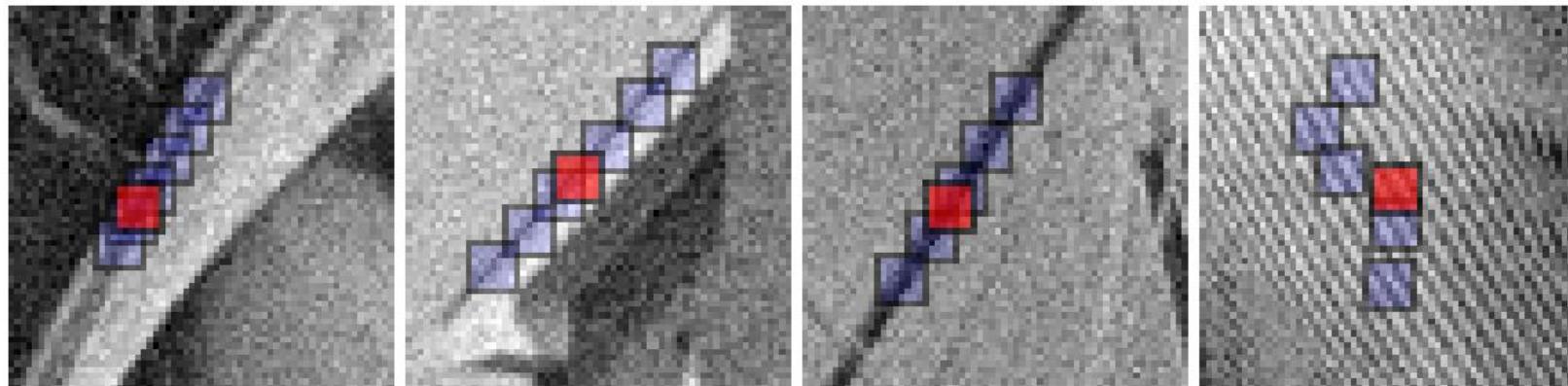
# Histogram LBP

- Pour chaque valeur possible du pointwise LBP, on construit une image binaire étant l'ensemble des pixels du pointwise LBP ayant cette valeur. Ensuite, on convolute avec un noyau au choix chaque image binaire (carré constant dans le cas de l'histogramme classique).
- Résultats pour  $(R,N)=(3,24)$  :



## Etape 2 : KNN des patchs

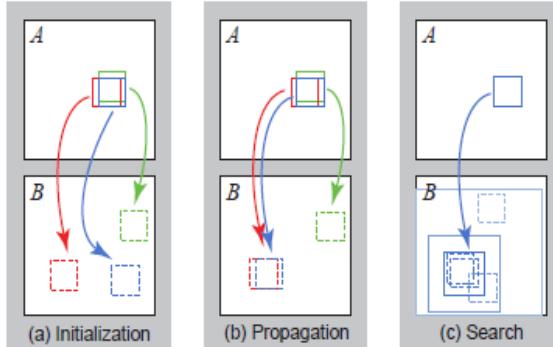
- Pour chaque patch, on veut trouver les patchs les plus semblables dans l'image, en utilisant une métrique au choix (le plus souvent la distance euclidienne).



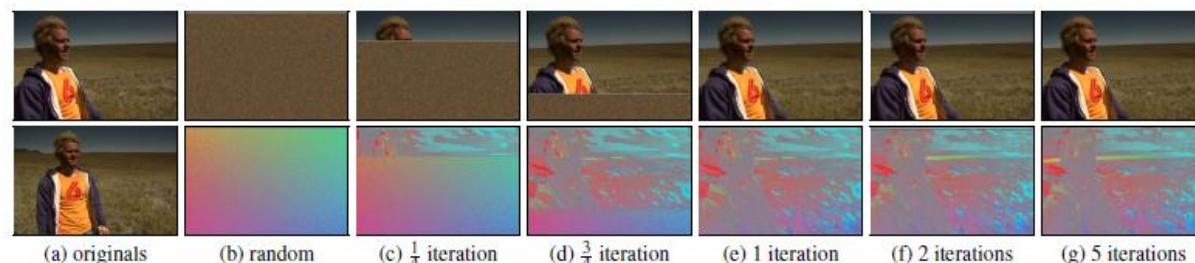
- Calculer cela est extrêmement long si on fait une recherche extensive ( $O(n^2)$  où  $n$  est le nombre de pixels). On va donc utiliser une méthode appelée ‘Generalized patch match’ qui est linéaire en le nombre de pixels.

# Generalized PatchMatch

- Nous l'avons codé en C++/Mex (utilisable depuis Matlab)



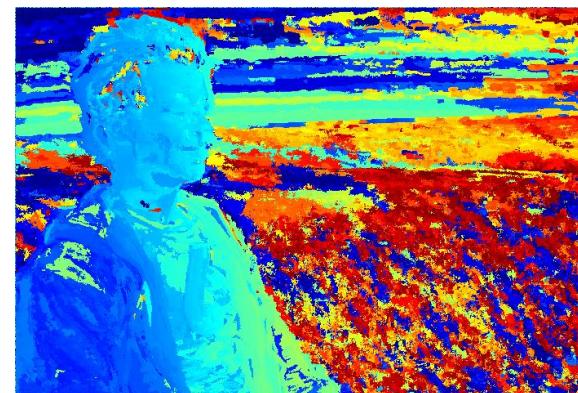
**Figure 2:** Phases of the randomized nearest neighbor algorithm: (a) patches initially have random assignments; (b) the blue patch checks above/green and left/red neighbors to see if they will improve the blue mapping, propagating good matches; (c) the patch searches randomly for improvements in concentric neighborhoods.



**Figure 3:** Illustration of convergence. (a) The top image is reconstructed using only patches from the bottom image. (b) above: the reconstruction by the patch “voting” described in Section 4, below: a random initial offset field, with magnitude visualized as saturation and angle visualized as hue. (c)  $\frac{1}{4}$  of the way through the first iteration, high-quality offsets have been propagated in the region above the current scan line (denoted with the horizontal bar). (d)  $\frac{3}{4}$  of the way through the first iteration. (e) First iteration complete. (f) Two iterations. (g) After 5 iterations, almost all patches have stopped changing. The tiny orange flowers only find good correspondences in the later iterations.

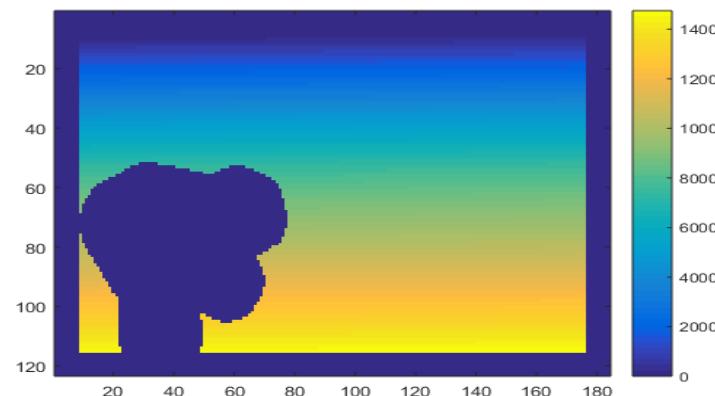
# Sortie de patch match (KNN)

- Output de patch match :
- Pour chaque  $i$  de 1 à  $K$ , on a un ‘X map’ et un ‘Y map’ :



# Matrice des distances entre patchs

- On numérote les pixels dont les patches sont entièrement connus, et on utilise les KNN trouvés avec PatchMatch pour remplir les lignes de la matrice des distances :



## Etape 3 : Matrice des corrélations

- On choisit (le plus souvent) un noyau gaussien :

$$C(x, y) = \exp\left(-\frac{d(x, y)^2}{2\sigma^2}\right)$$

- On peut choisir  $\sigma = cte = median_{(x,y) \in image}(d(x, y))$  par exemple, où bien prendre  $\sigma$  en fonction de  $x$  et de  $y$ .

# Etape 4 : Normalisation de la matrice des corrélations

- Il existe principalement deux types de normalisations (au choix):
  - La normalisation de type ‘matrice markovienne’ (aussi appelée de type ‘marche aléatoire’). On divise chaque ligne par sa somme. Formellement :
$$D_{ii} = \sum_j w_{ij}, \text{ et } W_{mar} = D^{-1}W$$
  - La normalisation symétrique :
$$W_{sym} = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$$
Ces matrices sont semblables :  $W_{sym} = D^{\frac{1}{2}}W_{mar}D^{-\frac{1}{2}}$ . On va ensuite extraire les vecteurs propres associés aux plus grandes valeurs propres de la matrice de corrélation normalisée.
- Formalisation différente mais équivalente mathématiquement : trouver les vecteurs propres associés aux plus petites valeurs propres de la matrice du Laplacien du graphe, qui vaut selon son type de normalisation :

$$L = D - W, L_{mar} = I - W_{mar}, L_{sym} = I - W_{sym}$$

## Etape 5 : diagonalisation (fin de diffusion maps)

- La diagonalisation est faite par Matlab (fonction ‘eigs’)
- Chaque vecteur propre peut être représenté comme une image en replaçant les valeurs à leur position (x,y) en fonction de l’indication établi précédemment.

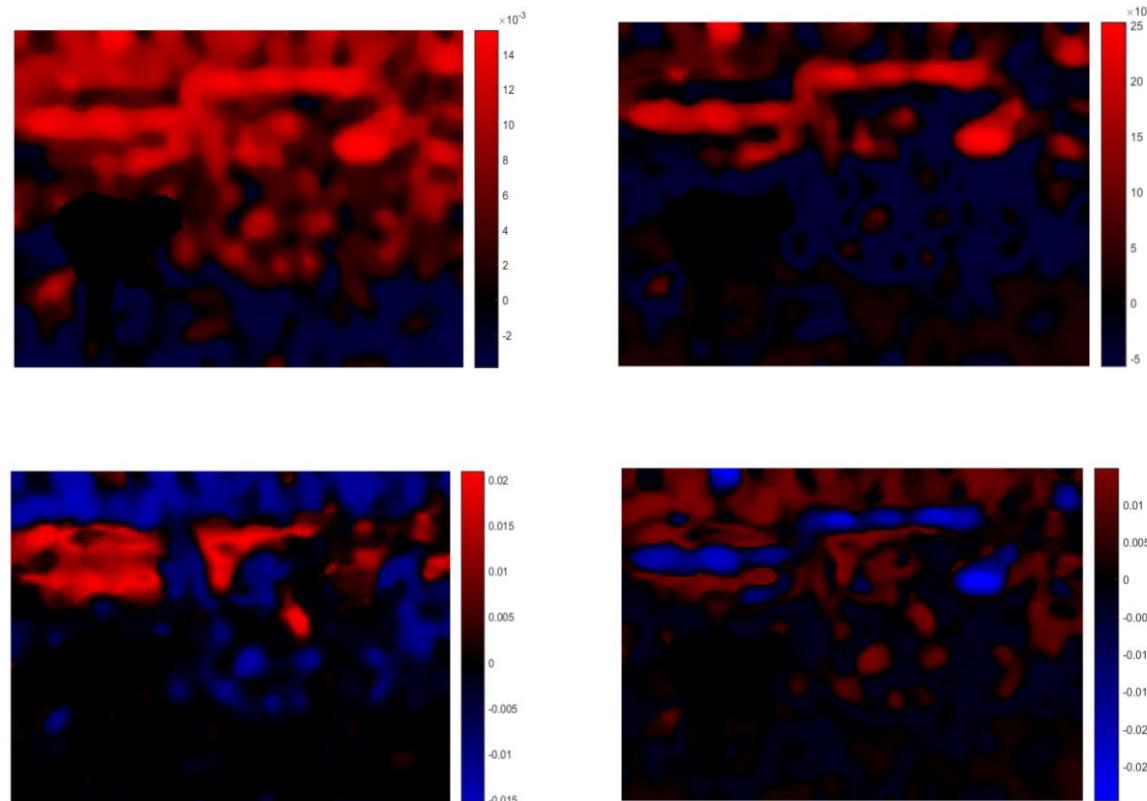
# Comparaison des résultats selon le type de descripteur (1 : Wexler)

- Wexler : vecteurs propres 6 et 8 : puisque Wexler comporte la couleur (composantes RGB), les pixels ayant la même couleur sont groupés dans les mêmes vecteurs propres (ici les vecteurs propre numéro 6 et 8). Le fait que le résultat s'apparente à une segmentation est utilisé dans Shi et Malik (2000).



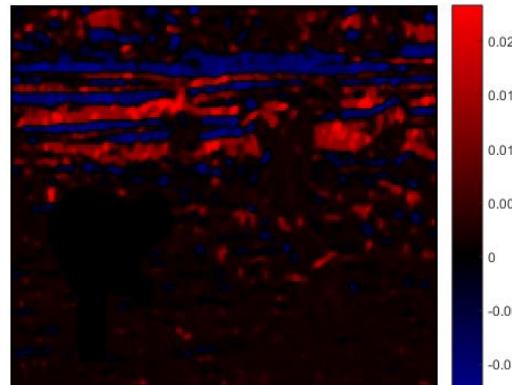
# Comparaison des résultats selon le type de descripteur (2 : rotationnal invariant LBP)

- Rotationnal invariant LBP : les composantes sont plus smooths



# Comparaison des résultats selon le type de descripteur (3 : uniform LBP)

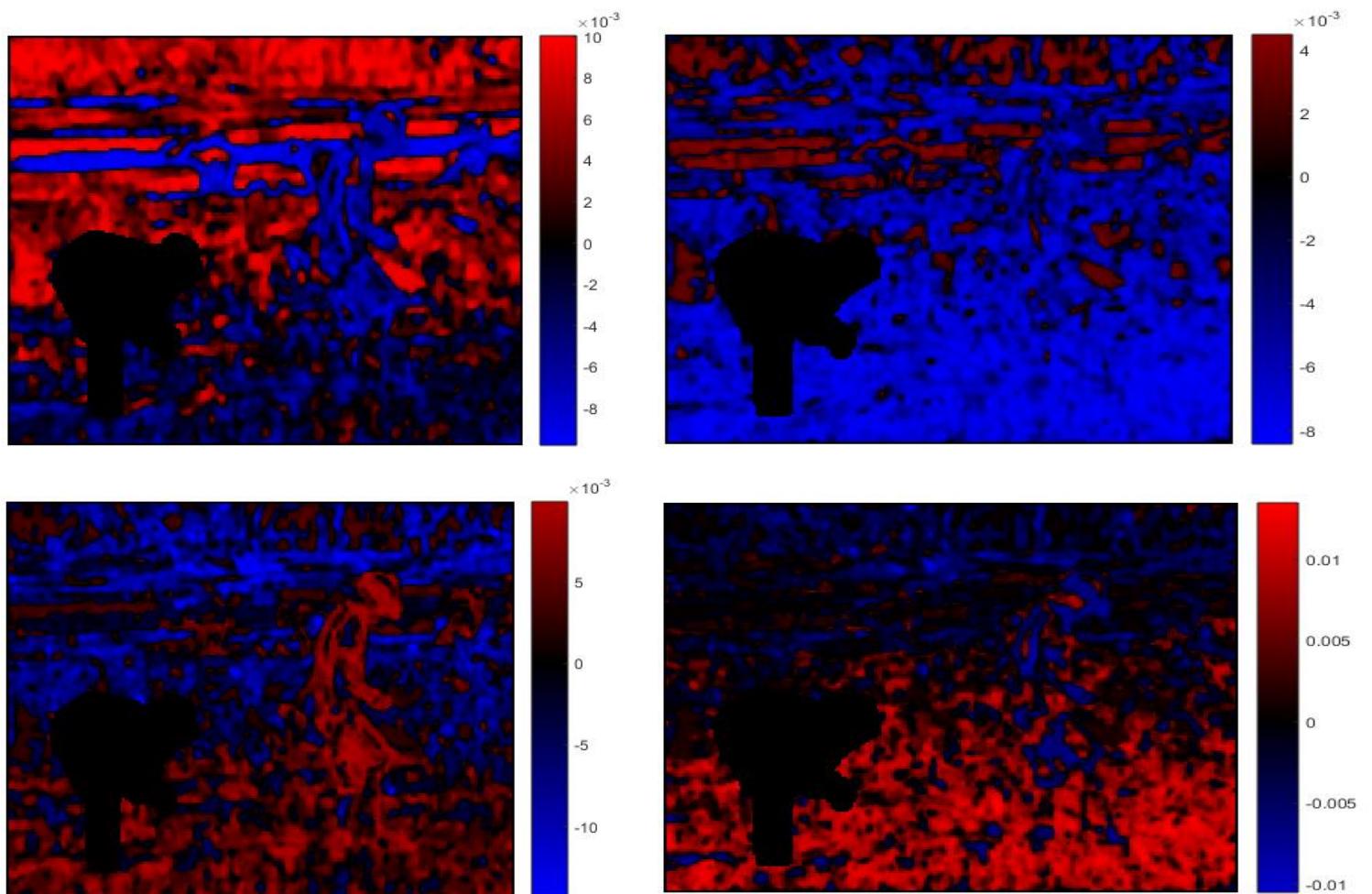
- Uniform LBP :



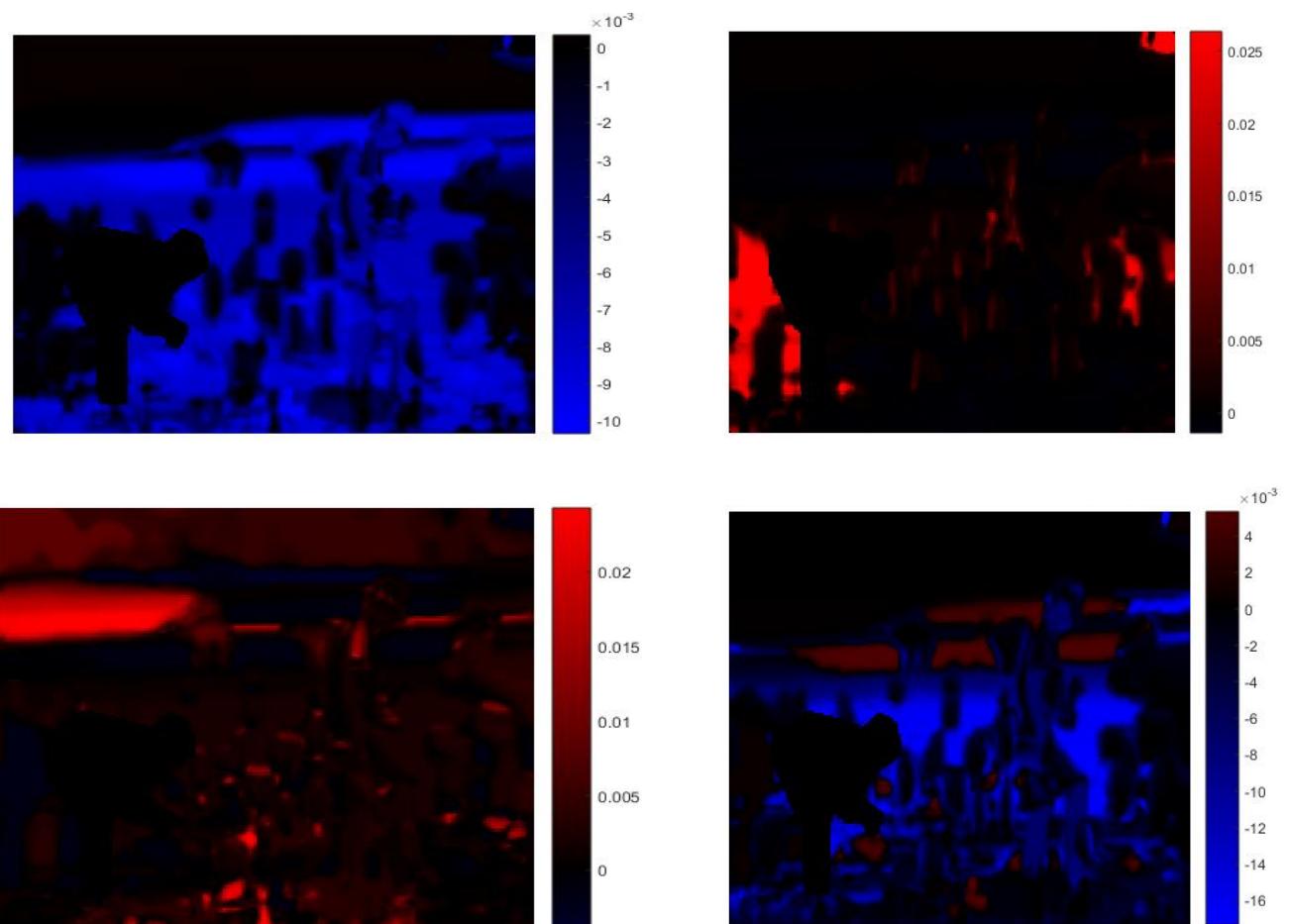
- We can see that the components are smooth but they take into account the orientation similarities, contrary to rotationnal invariant LBP.

# Comparaison des résultats selon le type de descripteur (4 : LBP with variance)

- LBP with Variance :



# Comparaison des résultats selon le type de descripteur (5 : windowing)

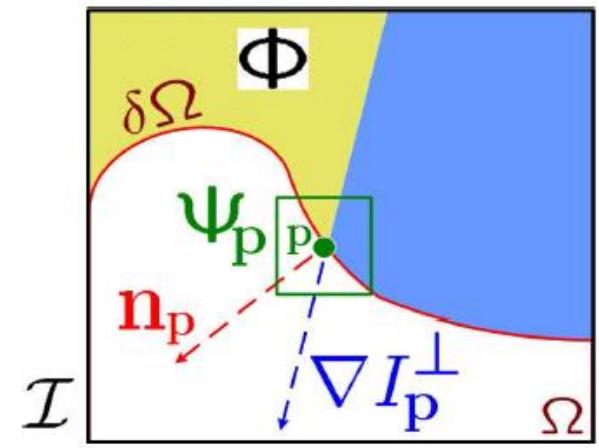
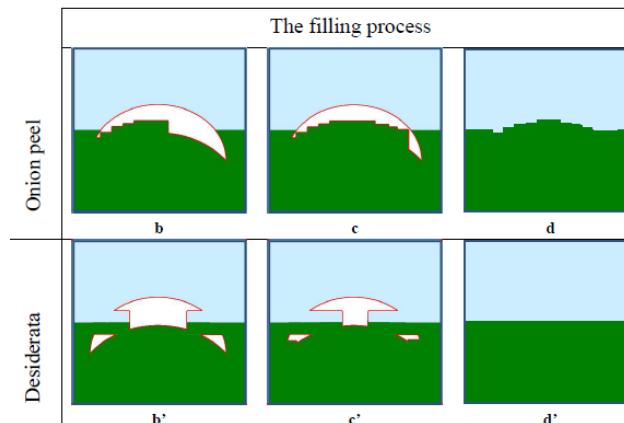
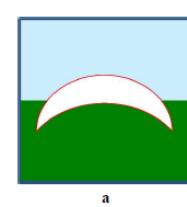
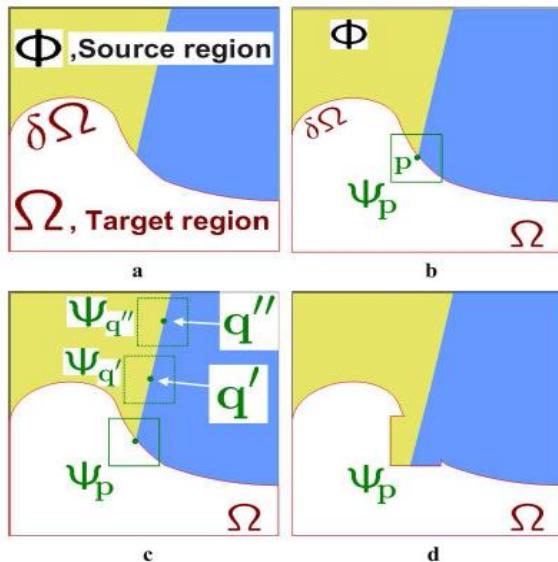


# Etape 6 : Inpainting des images obtenues (1)

- Dans le cas des images obtenues avec le descripteur de Wexler, les auteurs Gepshtein et Keller recommandent d'utiliser l'algorithme ‘best exemplar’ de Criminisi. Nous l'avons codé en Matlab et C++/Mex.

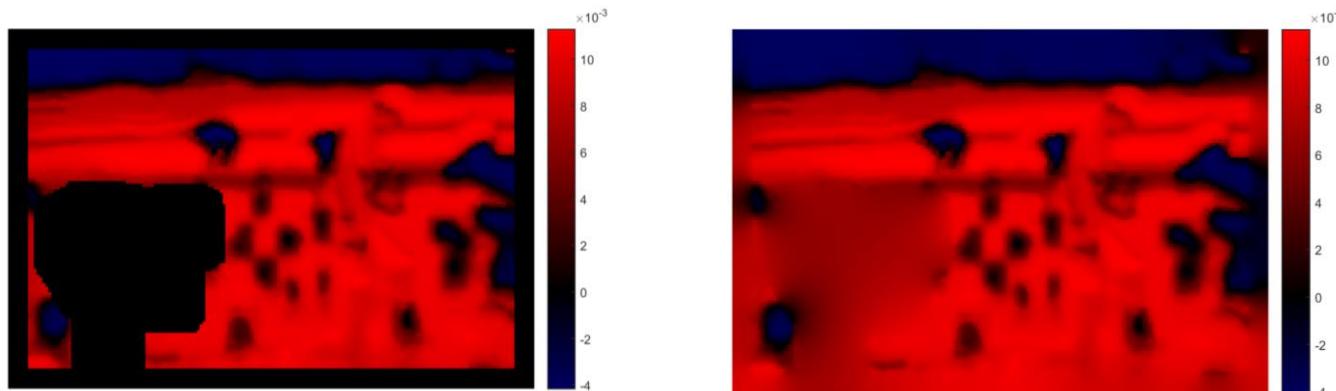


# Criminisi ‘best-exemplar’ inpainting



# Etape 6 : Inpainting des images obtenues (2)

- Dans le cas des images obtenues en prenant les descripteurs LBP, les auteurs remarquent que comme ces images sont assez smooths, on peut les inpainter avec une équation de la chaleur :  $\Delta u = 0$ . Nous l'avons discrétisé en utilisant le Laplacien discrétisé  $\frac{1}{20} \begin{pmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{pmatrix}$ , et en utilisant le solveur linéaire de Matlab. Une méthode alternative serait de coder une méthode de résolution itérative comme Jacobi ou de Gauss-Seidel.



# Etapes 7, 8 et 9

- Nous n'avons pas eu le temps de coder les étapes finales de l'algorithme de Gepshtein et Keller, mais voici tout de même la description de ces étapes.
- Etape 7 : pour tous les pixels inconnus, on cherche les  $K$  plus proches voisins (avec  $K=10$  par exemple) parmi les patchs entièrement connus, en utilisant comme espace de représentation les coordonnées issues des diffusion maps. On utilise pour cela l'algorithme PatchMatch.
- Etape 8 : On minimise une fonctionnelle de type SSD permettant, pour chaque pixel inconnu, de choisir un des  $K$  patchs parmi le choix qui nous est donné par l'étape précédente. Cette étape est une optimisation combinatoire qui est un problème NP, donc pour le résoudre de façon approximée, les auteurs proposent d'utiliser une relaxation linéaire du problème, qui se résoud par obtention d'un vecteur propre principal.

# Dernière étape, résultats de Gepshtain et Keller

- Etape 9 : pour chaque pixel, plusieurs patchs intersectent ce pixel (ceux associés aux pixels proches du pixel en question). On moyennent toutes les valeurs que le pixel est censé avoir d'après chacun des patchs l'intersectant, ce qui permet d'obtenir la valeur finale de ce pixel.



# Conclusions sur cet algorithme

- Cet algorithme est intéressant mais ne peut avoir un temps d'exécution raisonnable que grâce à PatchMatch, comme le signalait les auteurs, qui n'avait pas eu le temps de coder PatchMatch, et étaient donc limités à de petites images, ou bien nécessitaient de très gros temps de calculs pour calculer les plus proches voisins des patchs.
- Nous avons codé PatchMatch en C++/Mex afin de pouvoir ramener cet algorithme à des temps d'exécution plus raisonnables.
- Cependant, nous n'avons pas eu le temps de coder la relaxation spectrale. Au-delà du manque de temps, le passage de l'article décrivant la relaxation spectrale est assez peu clair, ce qui nous a rendu la tâche plus difficile. Nous suggérons aussi d'utiliser un champs de Markov (dont l'énergie peut être optimisée par recuit simulé par exemple) pour faire l'étape 7 à la place de la relaxation spectrale, mais nous n'avons pas eu le temps de le coder non plus.

# Conclusion

- Nous avons à la fois eu l'occasion d'appliquer ce que nous avons appris à Supaero en Matlab, et nous avons aussi appris à utiliser le C++ avec Matlab grâce à la compilation de fichiers Mex, que nous avons aussi appris à debugger. Ceux-ci permettent une execution plus rapide, en particulier lors de boucles, qui sont lentes en Matlab. Nous avons aussi découvert comment faire des GUI en Matlab permettant le réglage interactif de paramètres.
- Sur le plan théorique, nous nous sommes entraînés à faire des revues de littérature, et nous avons découvert des champs théoriques que nous ignorions auparavant, en particulier concernant l'inpainting, les techniques non locales, et le machine learning non supervisé.
- Nous avons aussi découvert Toronto et avons eu la chance de vivre quelques mois dans un pays étranger, ce qui nous a permis d'améliorer notre anglais, et de rencontrer des gens sympathiques avec une culture un peu différente de la notre.

Merci de votre attention