Suivi d'objets d'intérêt dans une séquence d'images : des points saillants aux mesures statistiques

Vincent Garcia

11 décembre 2008

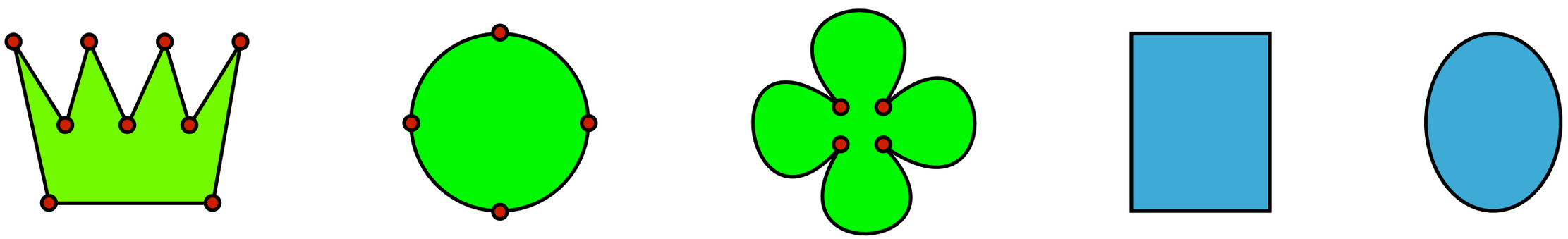
INTRODUCTION

Introduction

- Traitement d'images
  - Mathématiques appliquées
  - Étudier et/ou transformer les images
  - ➡ Réduire coup de stockage ou extraire une information sémantique
- Suivi d'objets
  - Vidéo : séquence d'images
  - Détection d'un objet sur la première image
  - Trouver la position/forme/taille de l'objet pour chaque image
  - Problème complexe, souvent réalisé manuellement
- Applications
  - Vidéo-surveillance (analyse traffic routier), rotoscopie (préservation d'anonymat, colorisation), compression vidéo, etc.

SUIVI D'OBJETS

- Objet
  - Région délimitée par un contour
  - Paramétrisation d'un contour
    - Points d'échantillonnage + Courbe (e.g. polygone, spline, Bézier)
    - Forme simple (e.g. rectangle, ellipse)
    - Paramétrisation doit être adaptée à l'objet et au problème / application
  - Contour édité manuellement sur la première image



Plan

- Suivi de la couronne de l'objet
  - Suivi des points d'échantillonnage par une méthode de block-matching
  - Masquage partiel des pixels du fond
  - Mesure statistique de similarité : entropie du résiduel
  - Validation sur séquences synthétiques et sur séquences réelles

- Suivi d'objets & mesures statistiques
  - Suivi d'objets
    - Descripteur : couleur + géométrie
    - Mesure de similarité : divergence de Kullback-Leibler calculée à partir de la distance au kPPV
  - Implémentation GPU de la recherche des kPPV
  - Expérimentations sur données synthétiques et sur séquences réelles

– 1 –

SUIVI DE LA COURONNE DE L'OBJET

SUIVI DE LA COURONNE DE L'OBJET

- Problème
  - Contour initial à l'image I1
  - Calculer le contour sur les autres images
  - Gestion des déformations locales
  - Calcul à partir contour précédant

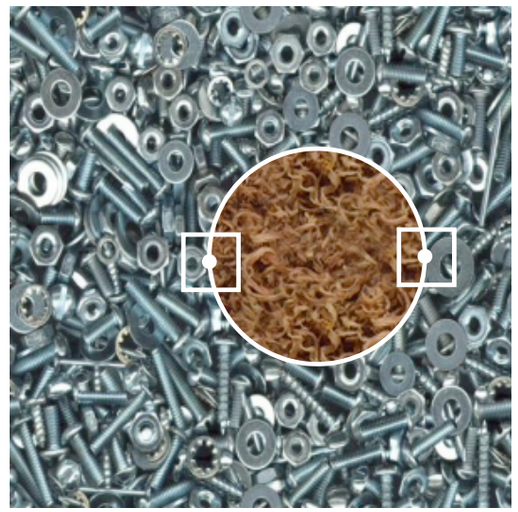
- Objet = points d'échantillonnage + courbe

- Approche proposée
  - Estimation du mouvement des points d'échantillonnage
  - Utilisabilité d'un algorithme de type block-matching

Approche classique

- Algorithme
  - Bloc à l'image It (matrice de pixel YUV, 33x33 pixels)
  - Trouver le bloc le plus similaire dans l'image It+1

$$v_i = \arg\min_u \sum_{x \in B_i} \varphi(r(x,u))$$

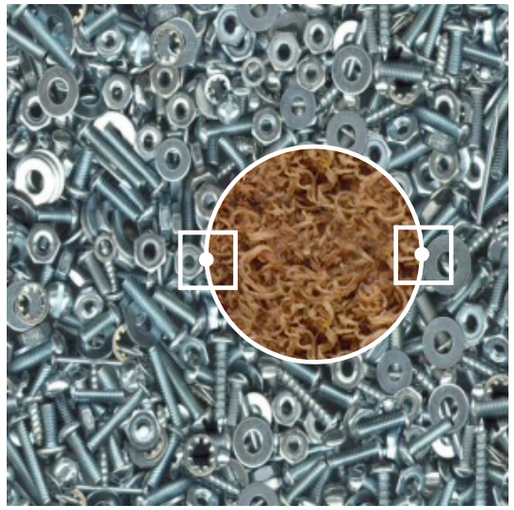


Vtex

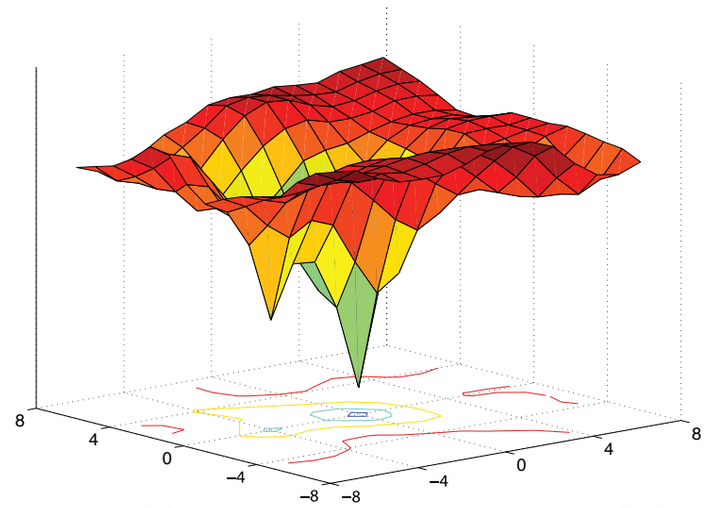
- vi : translation
- Bi : bloc
- Résiduel : $r(x,u) = I^t(x) - I^{t+1}(x+u)$
- SAD : $\varphi(x) = |x|$

Approche classique

- Séquence
  - 300 x 300 pixels
  - Objet texturé
  - Fond texturé
  - Mouvement : -4 pixels

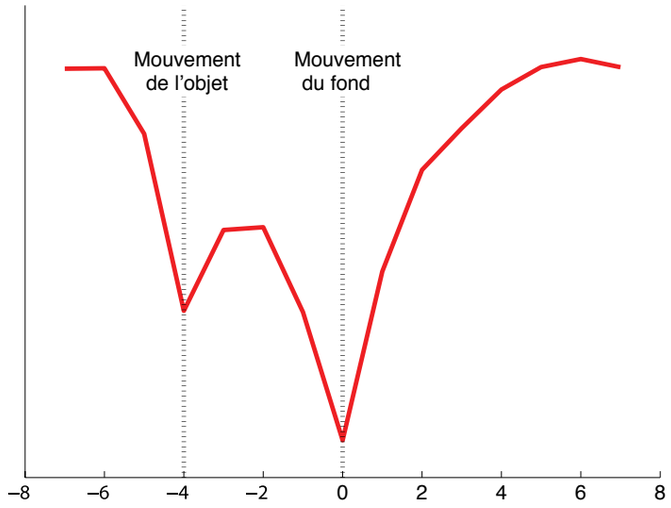


Vtex

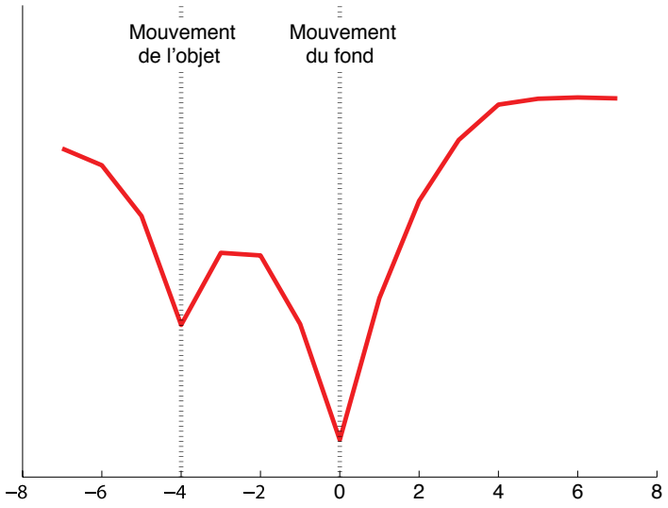


Bloc gauche - critère 3D

- Résultats
  - 2 minima
    - Mouvement fond (extremum)
    - Mouvement objet
  - Bloc
    - Pixels du fond (majoritaires)
    - Pixels de l'objet



Bloc gauche - profil 2D



Bloc droite - profil 2D

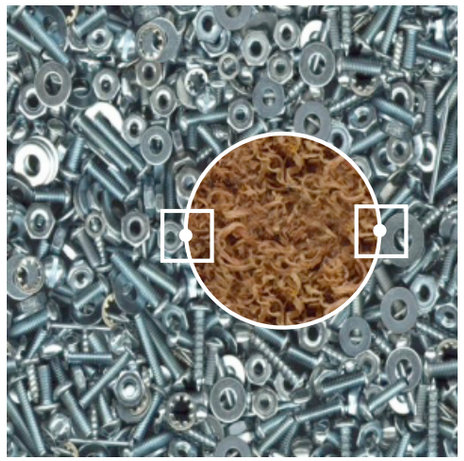
Masquage du fond

Principe

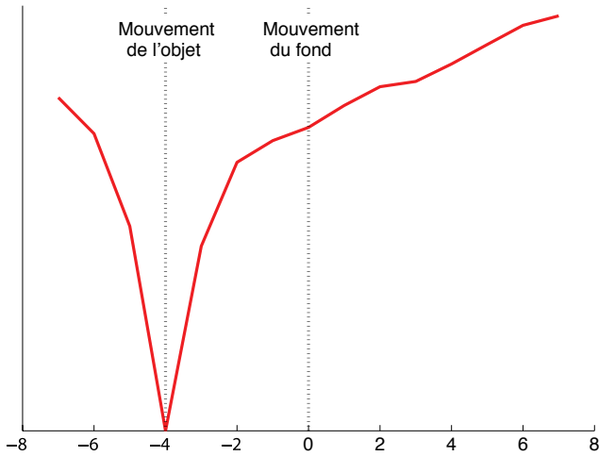
- Masquer les pixels du fond
- Masquage : $\Omega_i = B_i \cap D^t$

$$v_i = \arg\min_u \sum_{x \in \Omega_i} \varphi(r(x,u))$$

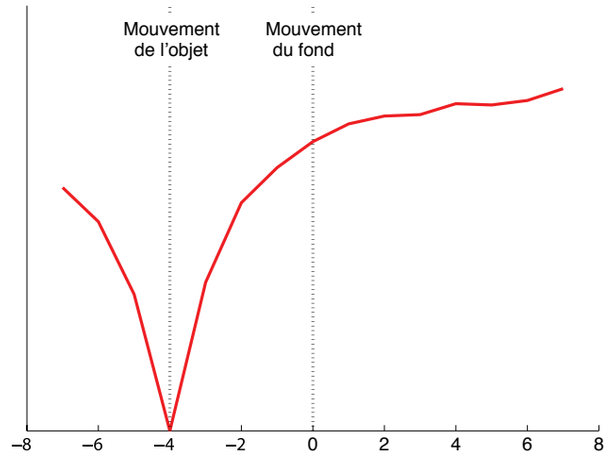
Résultats



Vtex

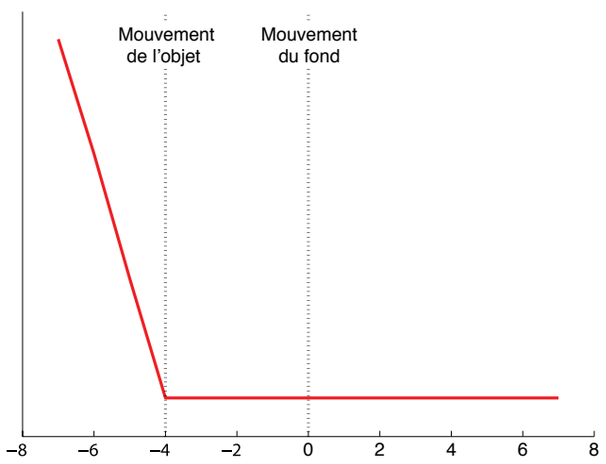


Bloc gauche

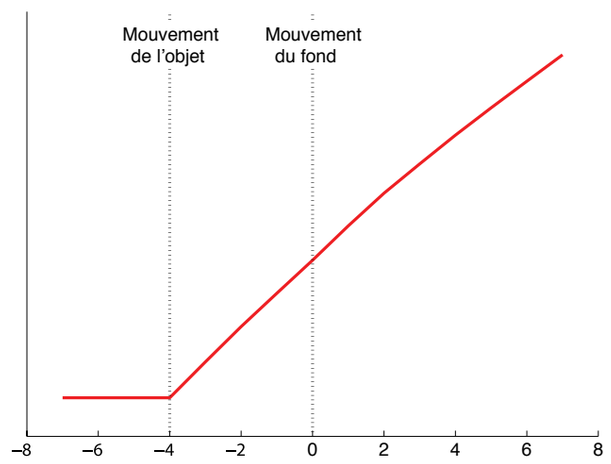


Bloc droite

Vhom



Bloc gauche



Bloc droite

Masquage partiel du fond

Principe

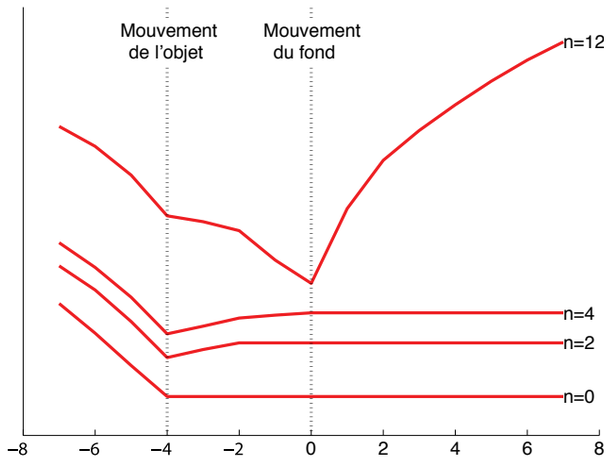
- Masquer partiellement les pixels du fond
- Masquage : $\tilde{\Omega}_i = B_i \cap d_R(D^t)$
- dR : dilatation mathématique de rayon R

$$v_i = \arg\min_u \sum_{x \in \tilde{\Omega}_i} \varphi(r(x,u))$$

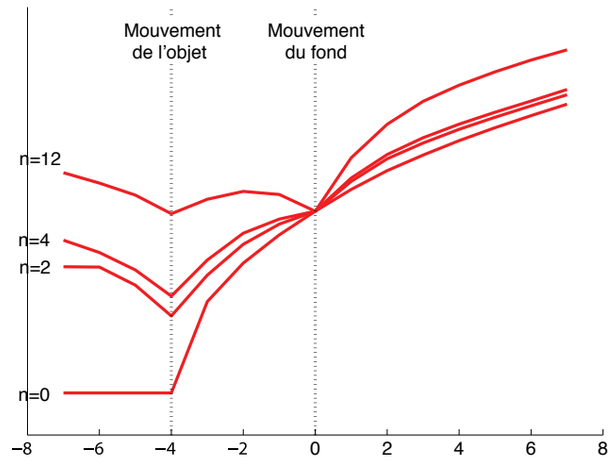
Résultats

- Dilatation permet de déterminer un minimum global
- Minimum dépend du rayon de dilatation

Vhom



Bloc gauche

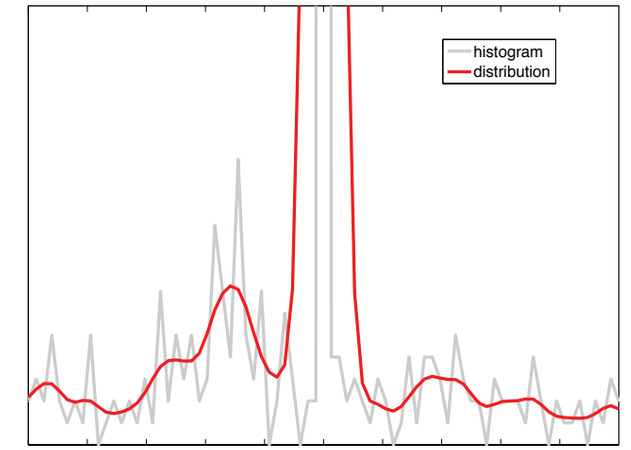


Bloc droite

CRITÈRES DE BLOCK-MATCHING

- Rayon de dilatation déterminant → étendre [Rmin, Rmax] au maximum

- Critère SAD
  - Efficace si résiduel paramétrique (Laplacien)
  - Hypothèse fausse en général



- Critère non paramétrique
  - Entropie du résiduel (Ahmad-Lin)
  - Dépend de la distribution réelle du résiduel
  - Estimation de la distribution du résiduel
    - Méthode à noyaux (Parzen)
    - Recherche des k plus proches voisins (kPPV)

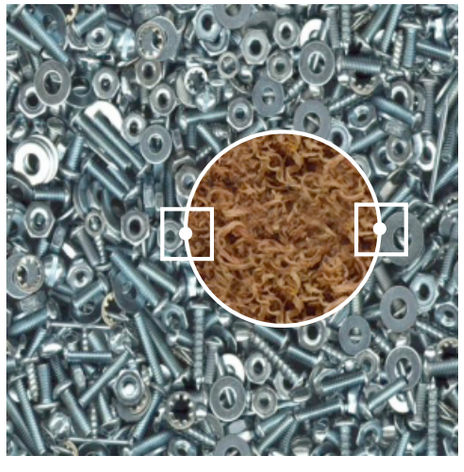
$$v_i = \arg\min_u -\frac{1}{|\tilde{\Omega}_i|} \sum_{x \in \tilde{\Omega}_i} \log(\widehat{f}(r(x,u)))$$

$$\widehat{f}(x) = \frac{1}{n\sigma\sqrt{2\pi}} \sum_{j=1}^{n} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

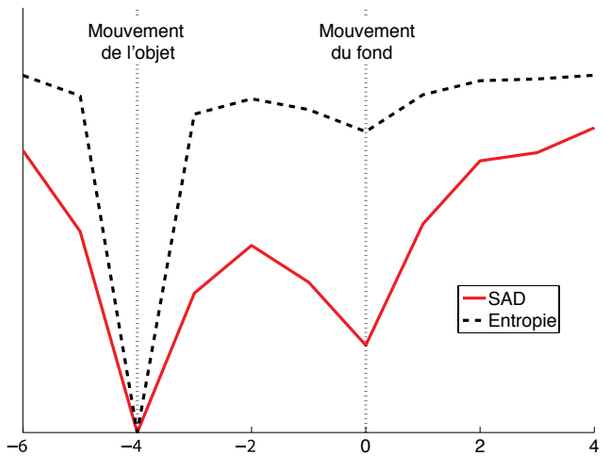
$$\widehat{f}(x) = \frac{k}{N_{\tilde{\Omega}_i} \; v_d \; \rho_k(x,\tilde{\Omega}_i)^d}$$

SAD vs Entropie

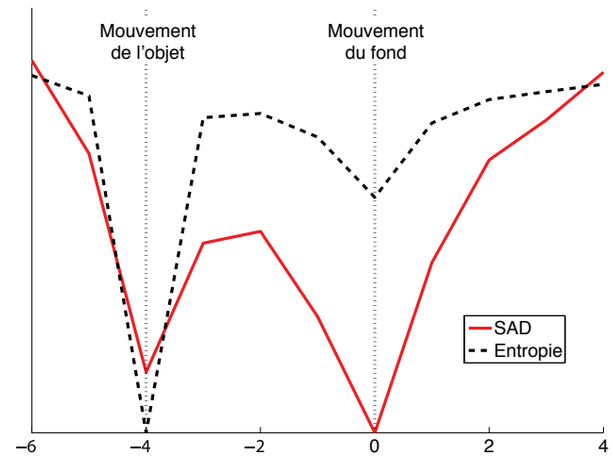
Profils



Vtex

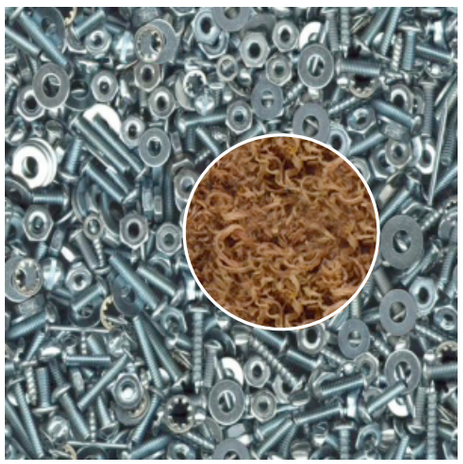


Bloc gauche - R=10



Bloc gauche - R=14

SAD & entropie, rayon = 5 pixels



11

19

116

SAD vs Entropie

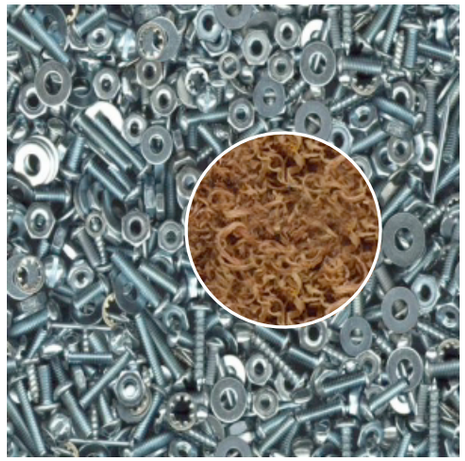
- Rayon R=10 pixels

SAD

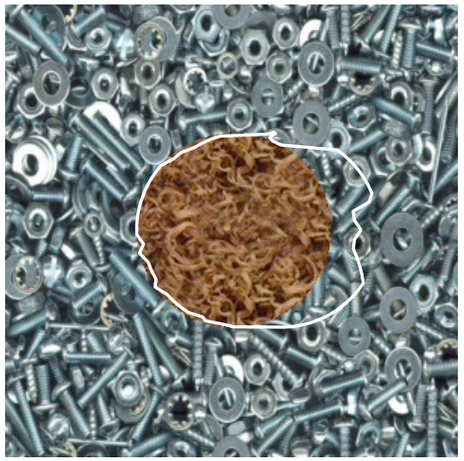
Entropie

SAD vs Entropie - Synthétique

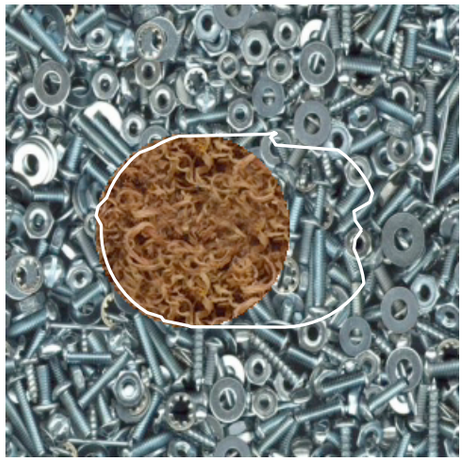
- Rayon R=14 pixels
- SAD



11



19



116

- Entropie

11

19

116

SAD vs Entropie - Car map

- Carmap - 320 x 240 pixels
- Rayon R = 5 pixels
- SAD

11

115

129

- Entropie

11

115

129

SAD vs Entropie - Soccer

- Carmap - 704 x 576 pixels
- Rayon R = 10 pixels
- SAD

11

15

19

- Entropie

11

15

19

Resultats - Ice

- Ice - 704 x 576
- Rayon R = 10 pixels
- SAD & Entropie

Conclusion

- Suivi d'objets déformables : block-matching (suivi rapide)
- Masquage partiel des pixels du fond (outliers)
  - Diminuer influence outliers
  - Garder la structure de bord
- Critère d'estimation non paramétrique : **entropie du résiduel**
  - Amélioration du suivi en diminuant l'impact des outliers
- Dilatation optimale
  - Indépendante pour chaque point d'échantillonnage
  - Proportion pixels fond / objet
    - Précision du contour initial
    - Courbure locale
  - Texture locale fond / objet
  - Netteté locale (motion blur)

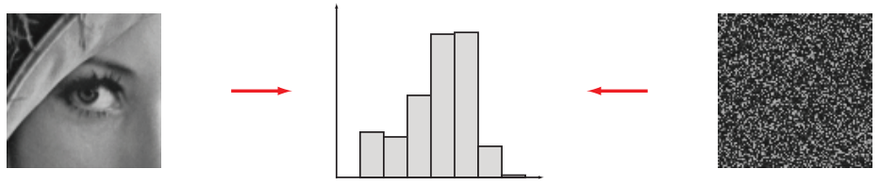
– 2 –

SUIVI D'OBJETS & MESURES STATISTIQUES

MOTIVATIONS

- Suivi d'objets
  - Boîte de référence à l'image I1
  - Trouver la boîte correspondante à l'image It+1
  - Gérer les déformations, rotations, occultations, etc.

- Comparaison de boîtes : critère de ressemblance usuels
  - Résiduel (SAD, SSD) : strictement géométrique
  - Information statistique (couleur) : sans géométrie



- Idée
  - Enrichir l'espace de description (augmenter pouvoir discriminant)
  - Conserver un critère statistique

Motivations

- Suivi d'objets selon Elgammal et al.
  - Descripteur : Couleur + position
  - Critère : Divergence de Kullback-Leibler (DKL)
  - Géométrie souple

- Divergence de Kullback-Leibler

$$D_{\mathrm{KL}}(P\|Q) = \int_{x\in\mathbf{R}^d} f_P(x)\log\frac{f_P(x)}{f_Q(x)}dx$$

- Inconvénient
  - Estimation des densités de probabilité de P et Q

MOTIVATIONS

- Suivi d'objets selon Boltz et al.
  - Descripteur : Couleur + position
  - Critère : DKL
  - Approximation de la DKL par la méthode des k plus proches voisins (kPPV)

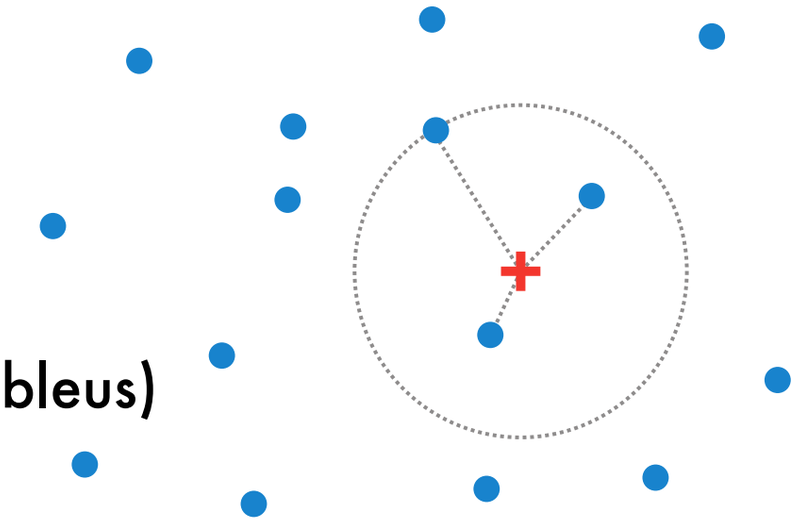
$$D_{\mathrm{KL}}(P\|Q) = \log \frac{N_{\mathcal{Q}}}{N_{\mathcal{P}} - 1} + \frac{d}{N_{\mathcal{P}}} \sum_{i=1}^{N_{\mathcal{P}}} \log(\rho_k(p_i, \mathcal{Q})) - \frac{d}{N_{\mathcal{P}}} \sum_{i=1}^{N_{\mathcal{P}}} \log(\rho_k(p_i, \mathcal{P} \setminus \{p_i\}))$$

- Algorithme
  - Localiser l'objet sur la première image → construction ensemble P
  - Hypothèse : suivi réalisé jusqu'à l'image I^t → (δ_n, σ_n)
  - Trouver le couple $(\delta_{n+1}, \sigma_{n+1})$ (ensemble Q) minimisant DKL

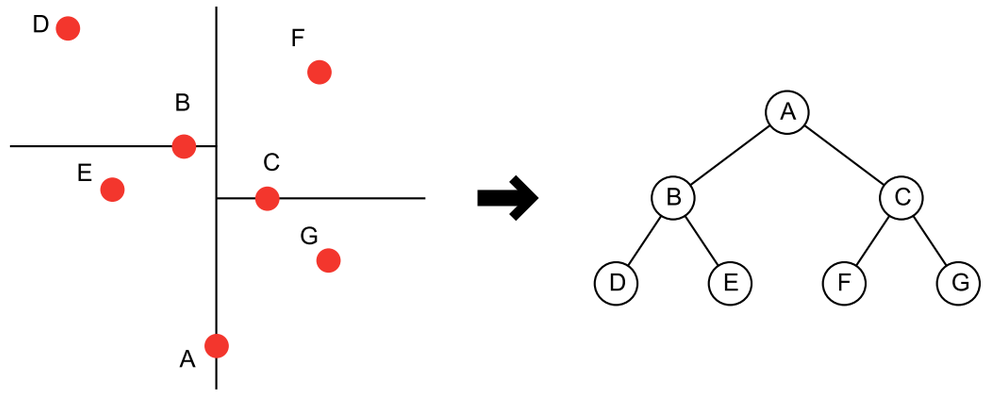
- Inconvénient : Recherche des kPPV lente en pratique

Recherche des kPPV

- Problème
  - Point requête (croix rouge)
  - Trouver les k plus proches points de référence (points bleus)



- Approches classiques
  - Recherche exhaustive (notée BF pour *brute-force*)
  - Partition de l'espace (kd-tree, etc.)
  - *Locality Sensitive Hashing*

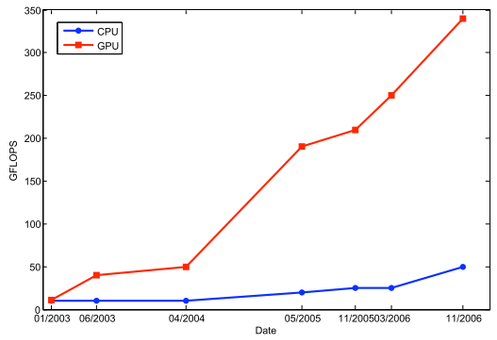


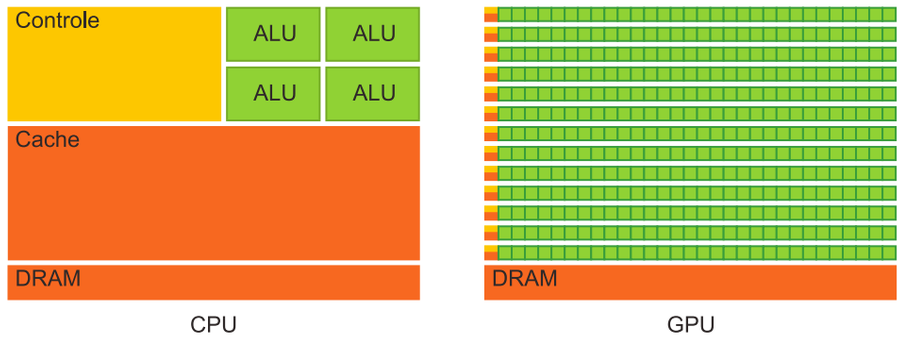
- BF hautement parallélisable : programmation GPU

PROGRAMMATION GPU

GPU : *Graphics Processing Unit*

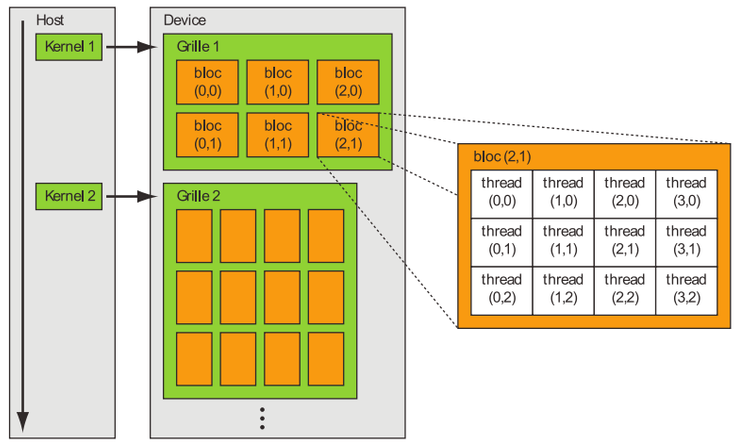
- Spécialisé dans le calcul parallèle pour des applications graphiques





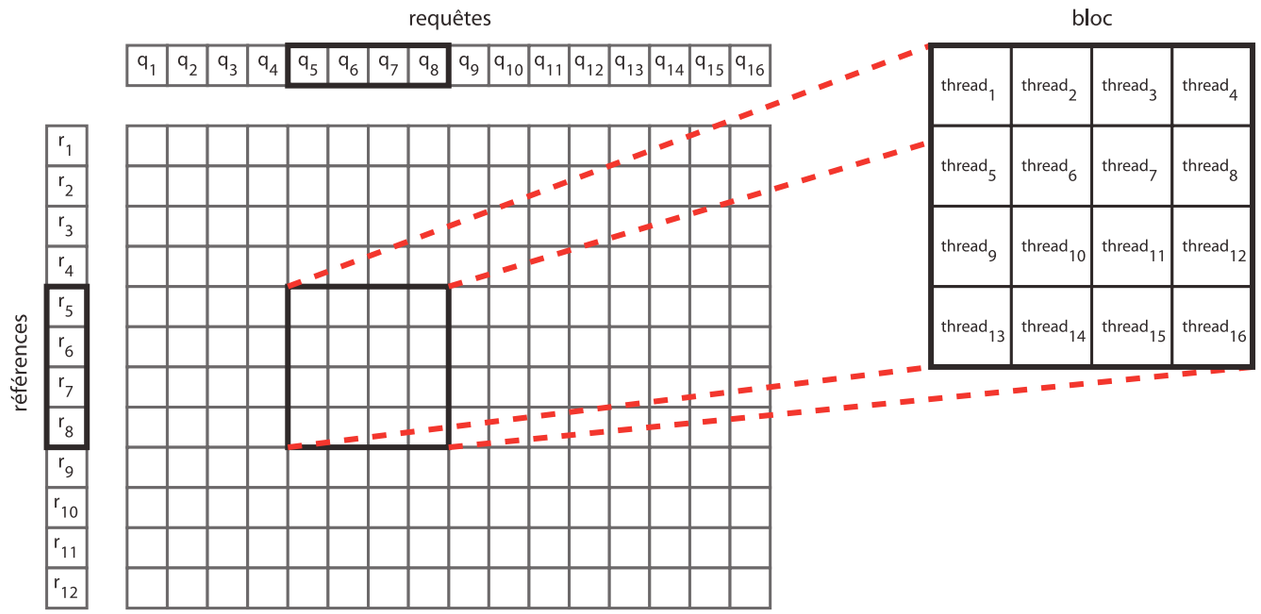
API NVIDIA CUDA (*Compute Unified Device Architecture*)

- Profiter de la puissance des GPU pour des applications non graphiques (GPGPU)
- Programmation parallèle en langage C
  - Transférer des données sur la carte graphique
  - Définir une grille de calcul
  - Définir une fonction (kernel)
  - Calculer en **parallèle** le kernel pour chaque élément de la grille

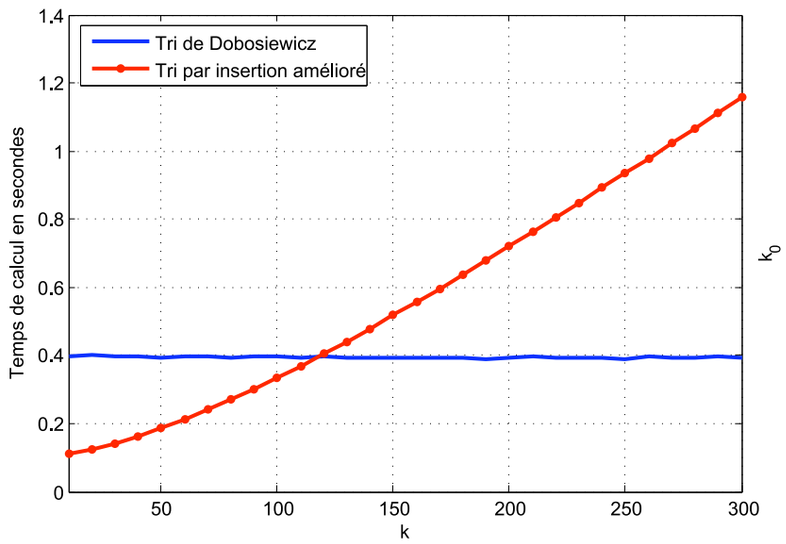


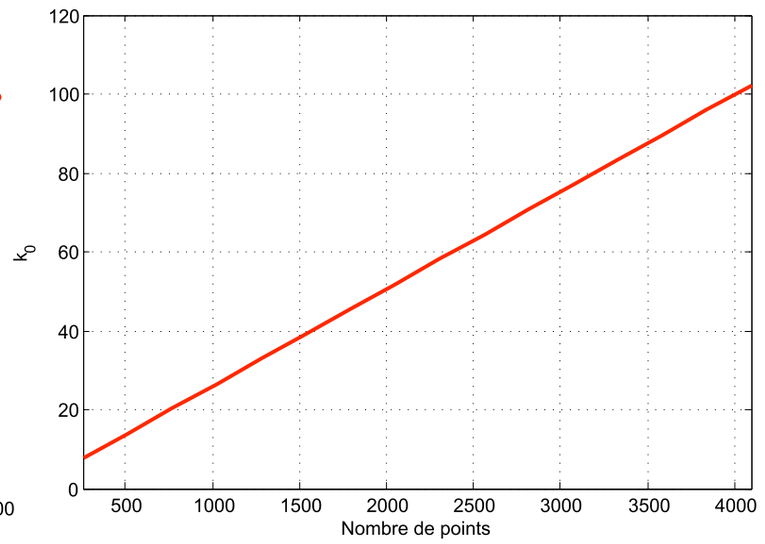
Recherche des kPPV en GPU

- Organisation du code
  - Définition d'une matrice de distances
  - Kernel 1 : Calcul des distances
  - Kernel 2 : Tri des distances



- Tri
  - ~~Quicksort (récursif)~~
  - Tri de Dobosiewicz
  - Tri par insertion (modifié)





4800 points de dimension 64

Expérimentations sur données synthétiques

- But
  - Etudier l'intérêt de la programmation GPU pour la recherche des kPPV
  - Confronter notre approche aux approches et implémentations classiques
- Approches testées
  - BF-MATLAB, BF-C, BF-CUDA
  - ANN-C++ (*Approximate Nearest Neighbor*, Mount and Arya)
- Ordinateur
  - Pentium IV 3.4 GHz, 2Go DDR2
  - NVIDIA GeForce 8800GTX (16x8 processeurs à 1.35 GHz)
  - Windows XP, CUDA 1.1
- Données
  - Nombres aléatoires U(0,1)

Expérimentations sur données synthétiques

- BF-CUDA est l'implémentation la plus rapide
  - 407 X plus rapide que BF-Matlab
  - 295 X plus rapide que BF-C
  - 148 X plus rapide que ANN-C++

Dimension	Methods	n=1200	n=4800	n=9600	n=19200
d=8	BF-Matlab	0.51	7.84	35.08	148.01
	BF-C	0.13	1.90	7.53	29.21
	ANN-C++	0.13	0.81	2.43	6.82
	BF-CUDA	**0.01**	**0.04**	**0.13**	**0.43**
d=16	BF-Matlab	0.74	12.60	51.64	210.90
	BF-C	0.22	3.45	13.82	56.29
	ANN-C++	0.26	5.04	23.97	91.33
	BF-CUDA	**0.01**	**0.06**	**0.17**	**0.60**
d=32	BF-Matlab	1.03	21.00	84.33	323.47
	BF-C	0.45	7.51	30.23	116.35
	ANN-C++	0.39	9.21	39.37	166.98
	BF-CUDA	**0.01**	**0.08**	**0.24**	**0.94**
d=96	BF-Matlab	3.30	55.77	231.69	901.38
	BF-C	2.54	39.26	168.58	674.88
	ANN-C++	1.20	19.68	82.45	339.81
	BF-CUDA	**0.02**	**0.15**	**0.57**	**2.29**

k=20

Expérimentations sur données synthétiques

- Répartition du temps de calcul CPU/GPU
- Paramètres
  - d = 32
  - n = 4800
  - k = 20

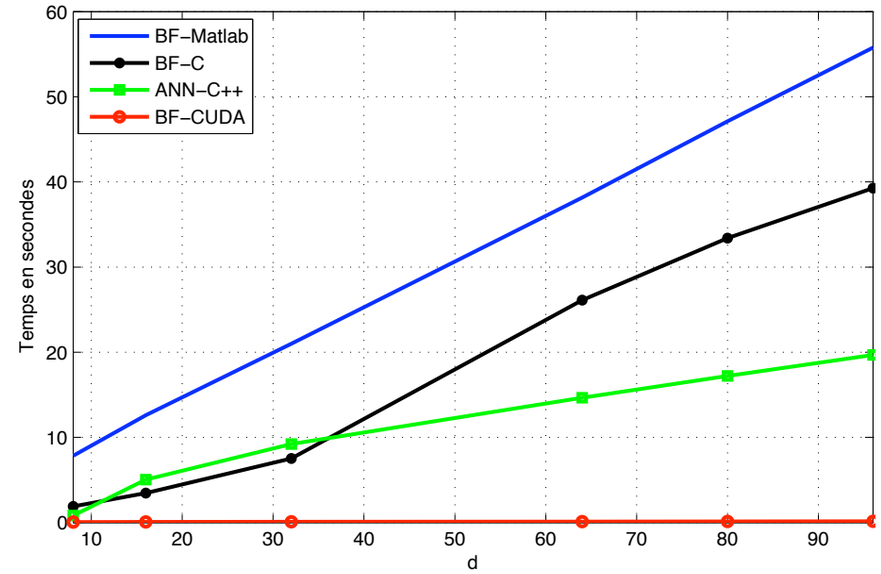
	CPU	GPU
Distances	91%	66%
Tri	4%	32%
Autre	5%	2%

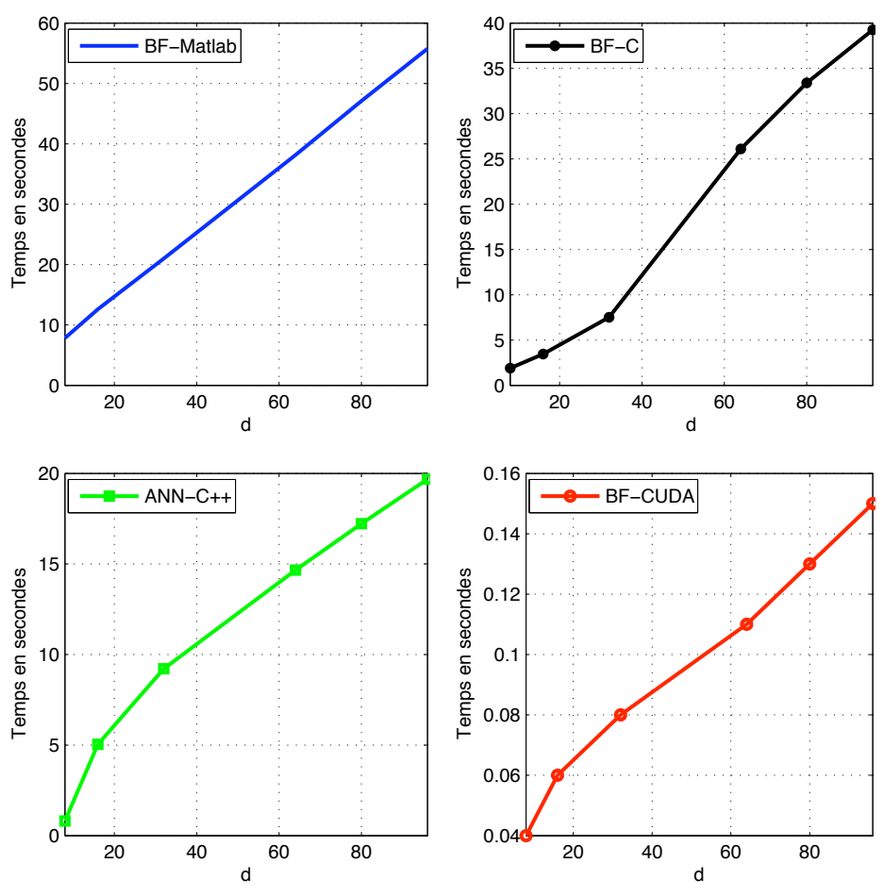
- Le calcul des distances est mieux adapté à la parallélisation que le tri

Expérimentations sur données synthétiques

- Influence de la dimension
  - d intervient dans le calcul des distances
  - Temps de calcul : Augmentation linéaire avec d
  - Influence réduite pour CUDA
  - Part du calcul des distances augmente avec d

d	8	16	32
Distances	37%	51%	66%
Tri	62%	47%	32%
Autre	1%	2%	2%
Total	0.040s	0.055s	0.076s

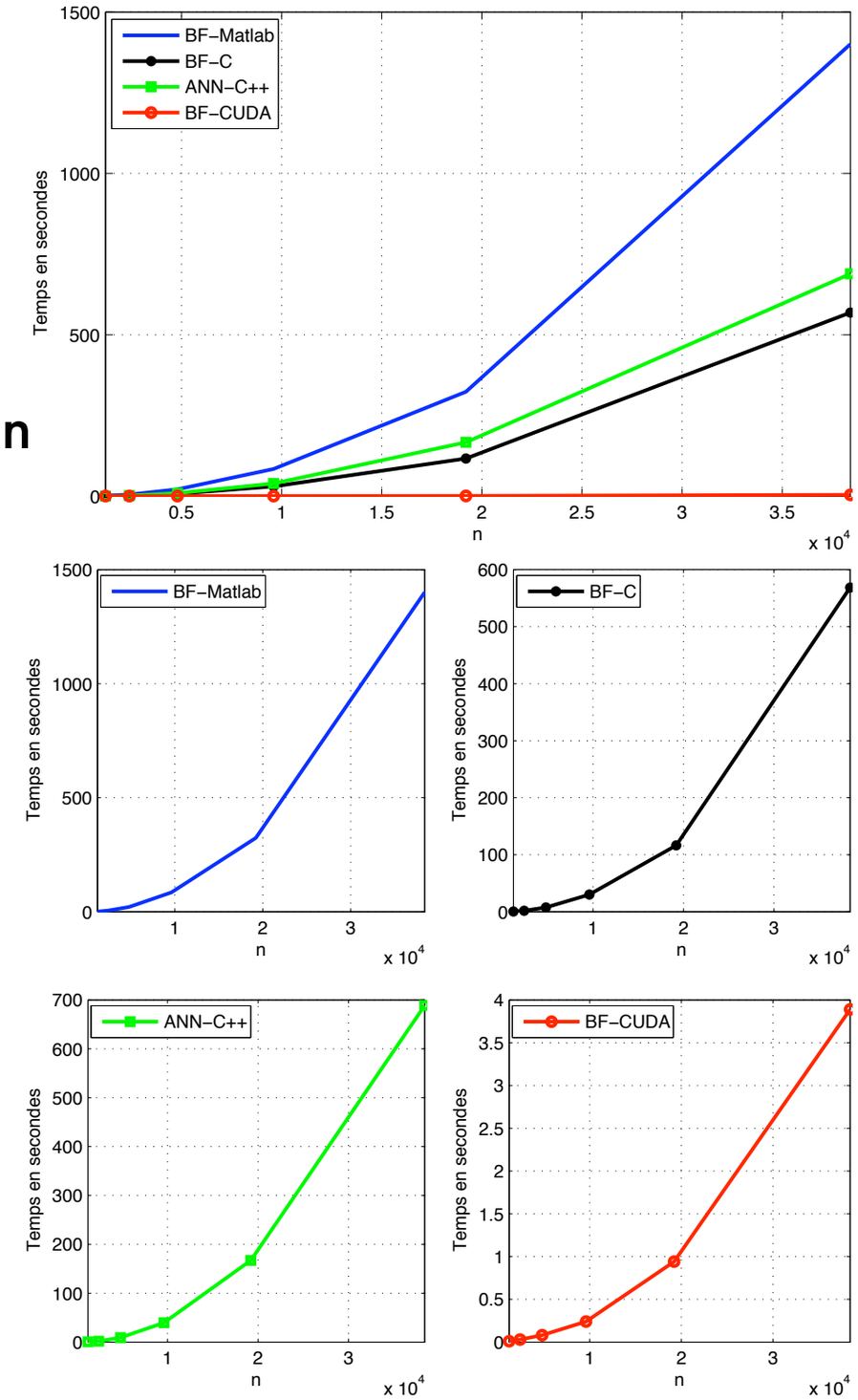




Expérimentations sur données synthétiques

- Influence du nombre de points
  - n intervient dans le calcul des distances et dans le nombre de tris
  - Temps de calcul : Augmentation polynomial avec n
  - Influence réduite pour CUDA
  - Part du calcul des distances augmente avec n

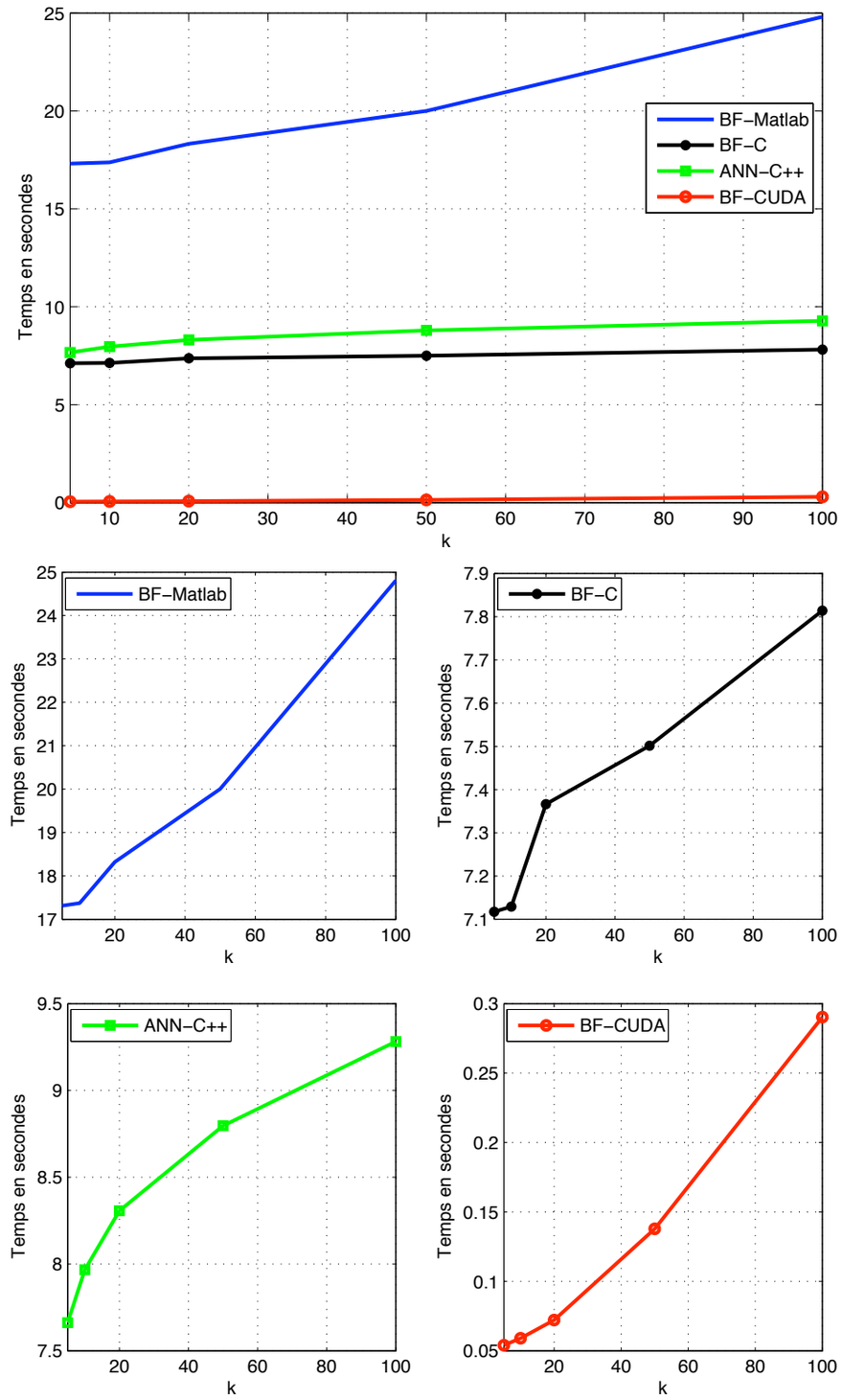
n	2400	4800	9600
Distances	28%	51%	59%
Tri	70%	47%	40%
Autre	2%	2%	1%
Total	0.023s	0.055s	0.169s



Expérimentations sur données synthétiques

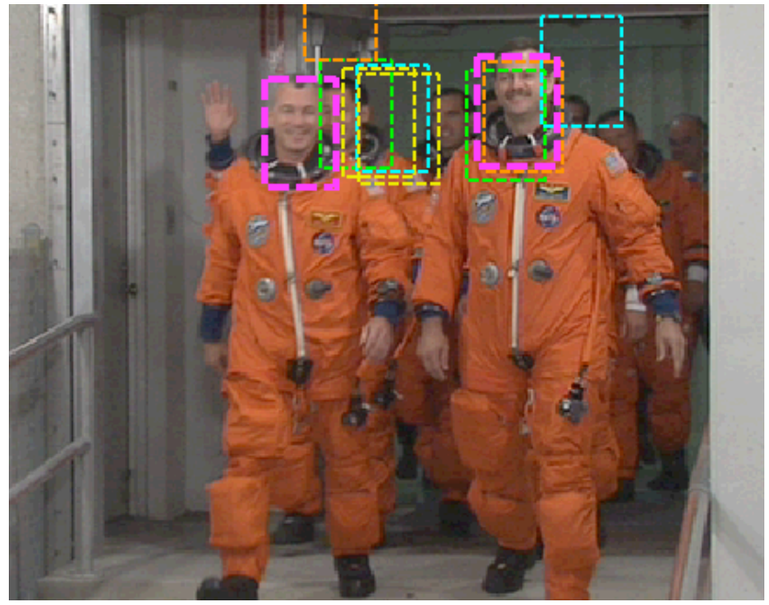
- Influence du paramètre k
  - k intervient dans l'étape de tri
  - Temps de calcul : Augmentation linéaire avec k
  - Part du tri augmente avec k

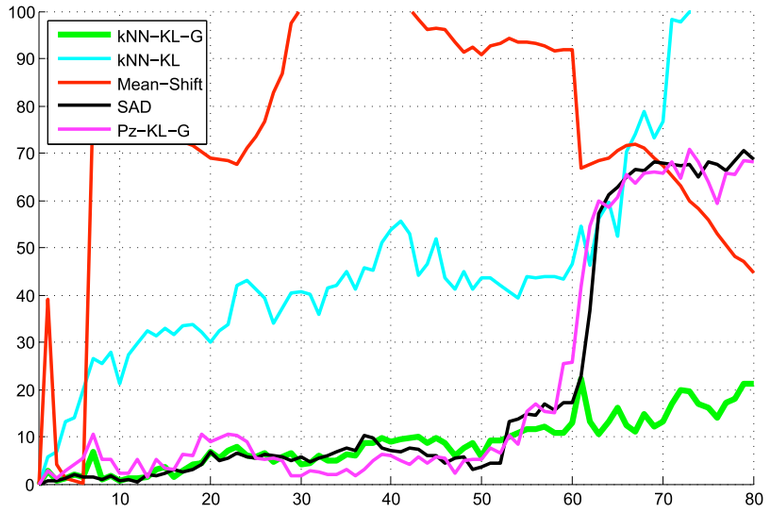
k	5	10	20
Distances	82%	71%	51%
Tri	15%	26%	47%
Autre	3%	3%	2%
Total	0.033s	0.037s	0.055s



Expérimentations sur données réelles

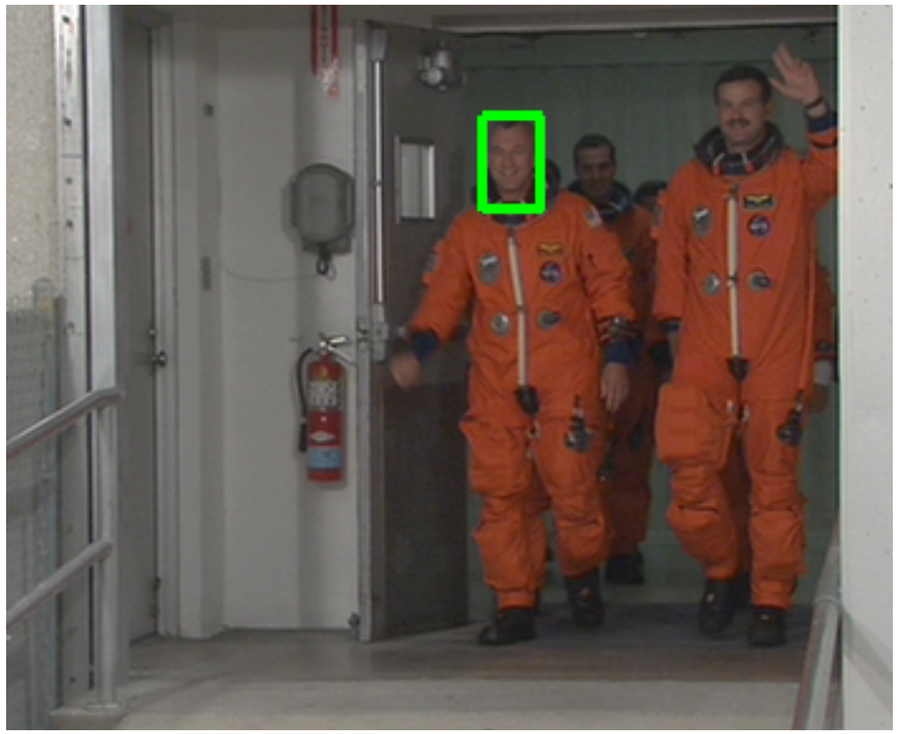
- Boltz et al.
  - couleur + géométrie + DKL + kPPV
  - Meilleurs résultats
- But
  - Essayer de nouveaux descripteurs
  - Combinaisons de composantes
    - C (YUV)
    - C3 (patch 3x3)
    - C9 (patch 9x9)
    - G (gradient)
    - P (position)
  - Comparaison programmation CPU (ANN-C++) et GPU (BF-CUDA)

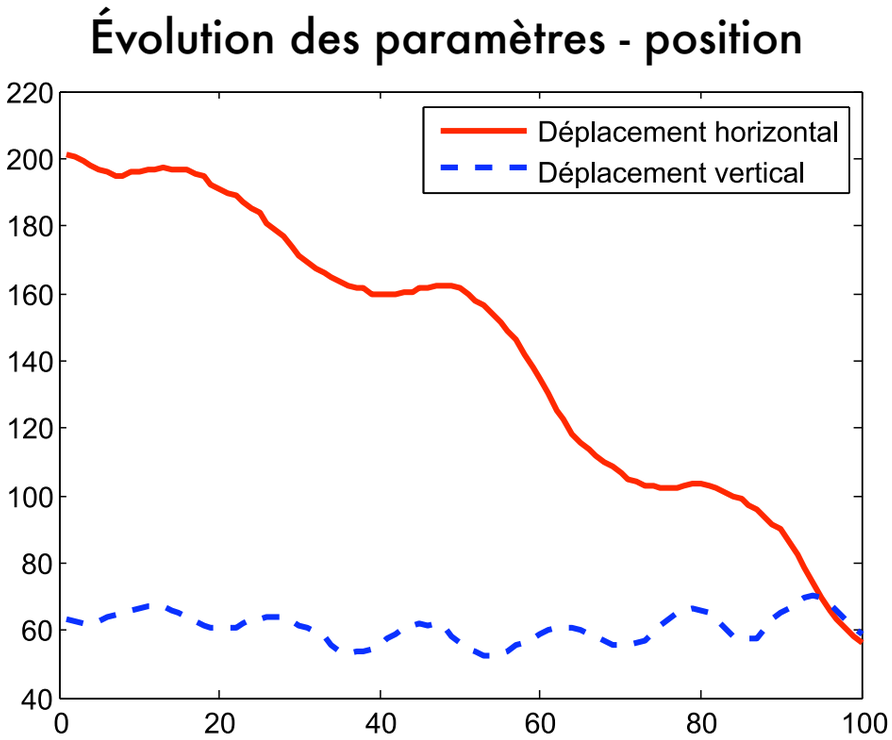


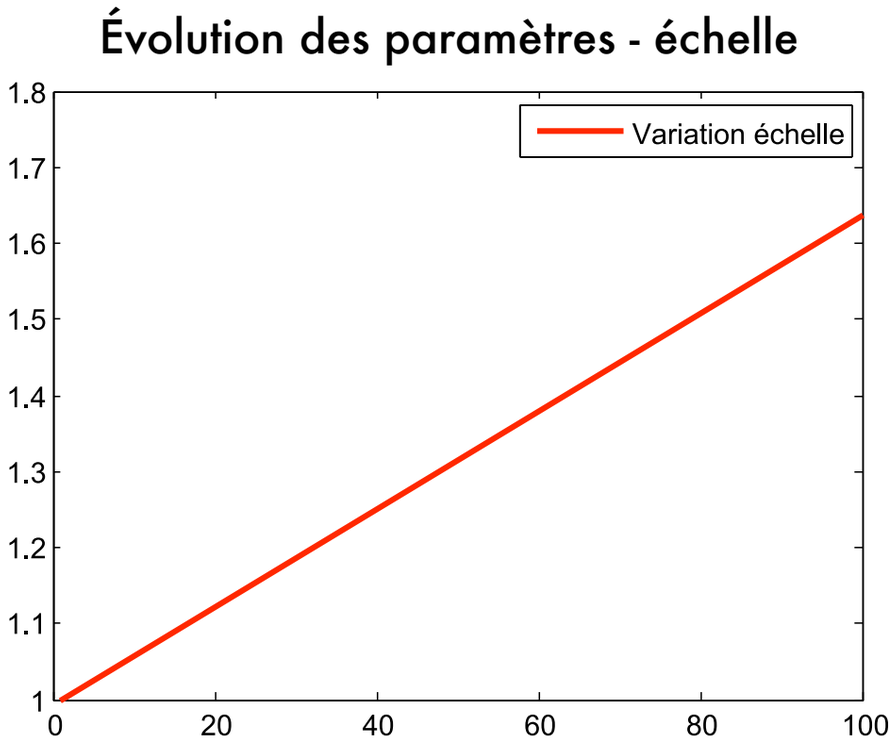


Expérimentations sur données réelles

- Séquence Crew
  - 352x288 pixels, 100 images
  - Objet : 900 pixels
  - Difficultés
    - Mouvement complexe
    - Flashs

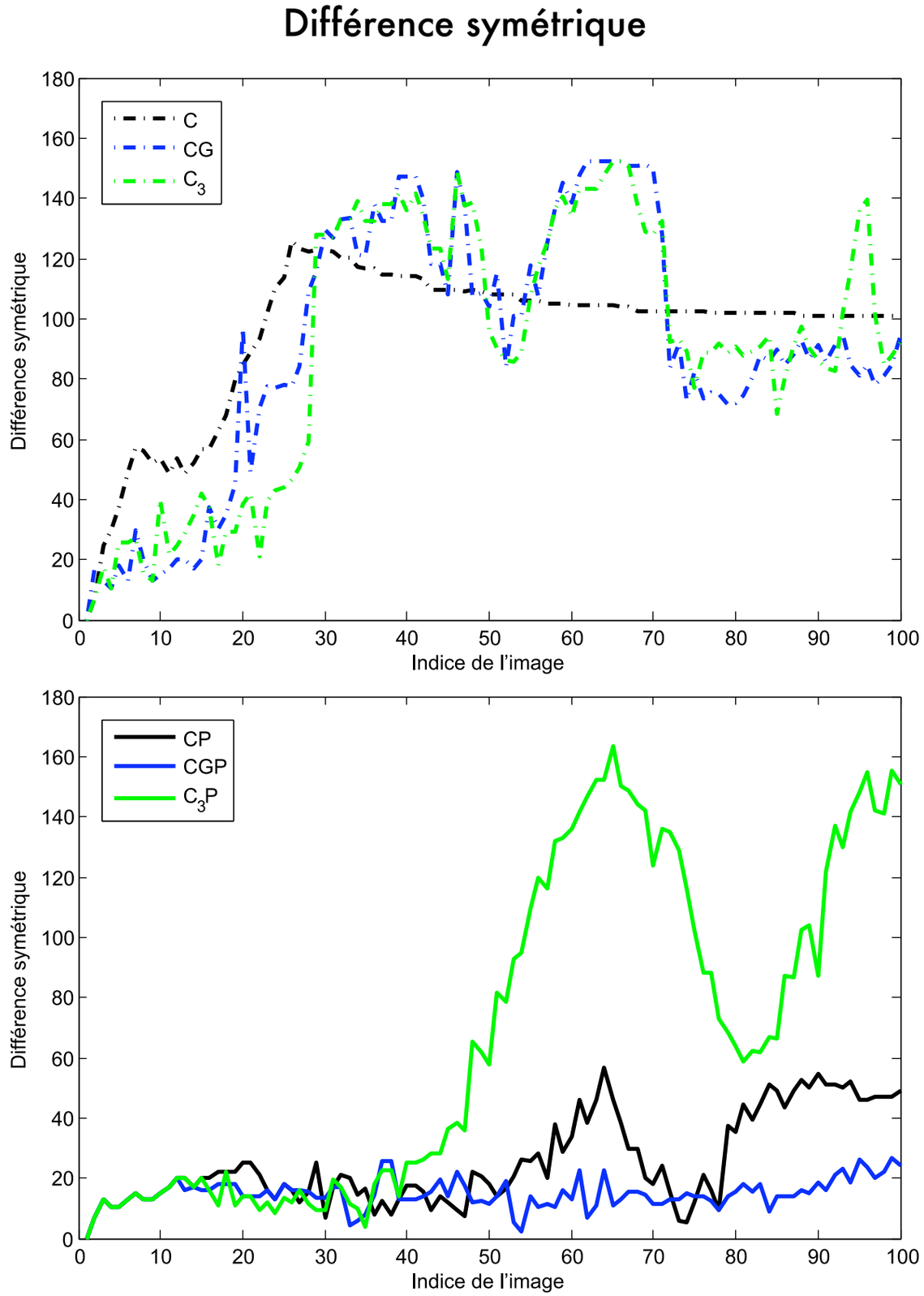






Expérimentations sur données réelles

- Descripteurs testées
  - C, CP, CG, CGP, C3, C3P
- Critère
  - Différence symétrique
- Résultats
  - Information géométrique vitale
  - Meilleur descripteur : CGP



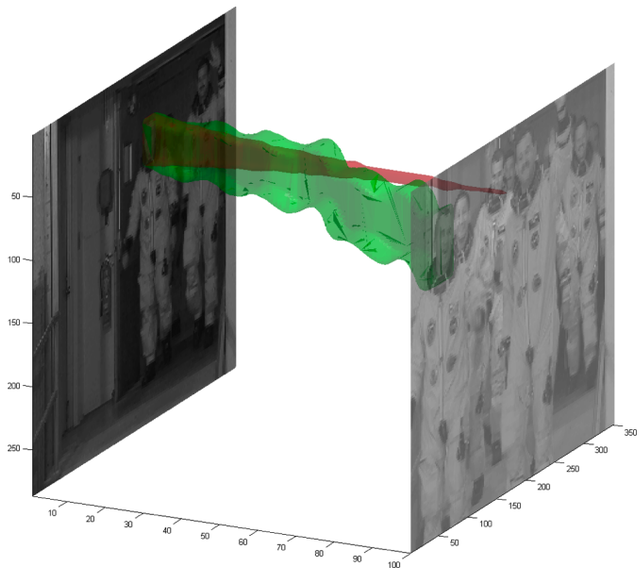
Expérimentations sur données réelles

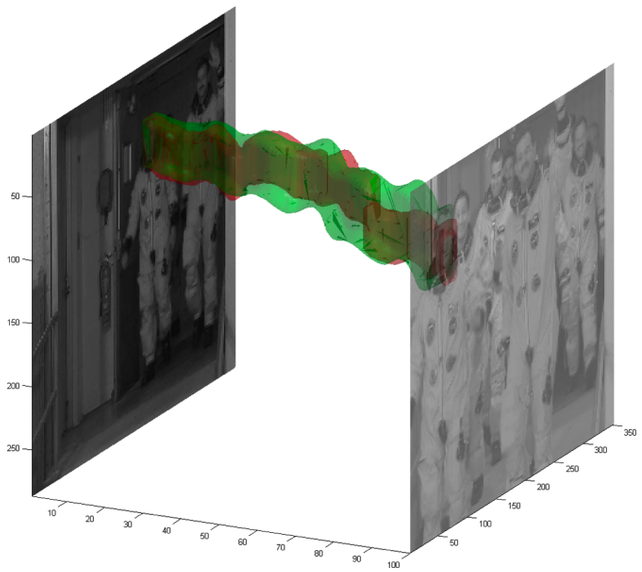
C

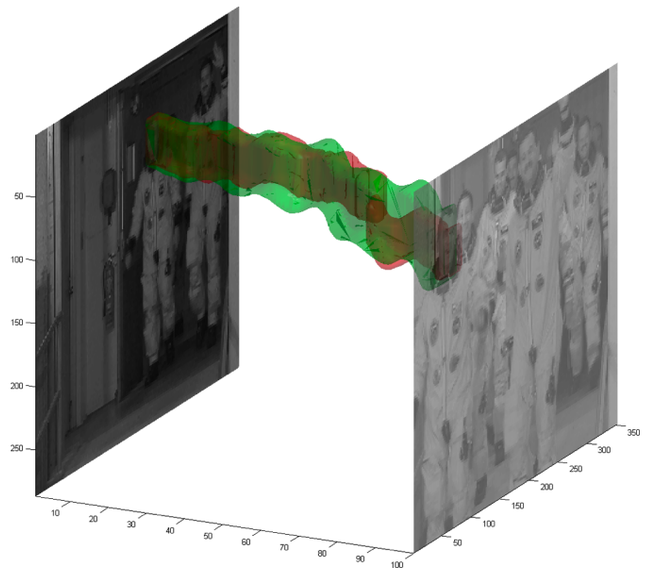
CG

C3

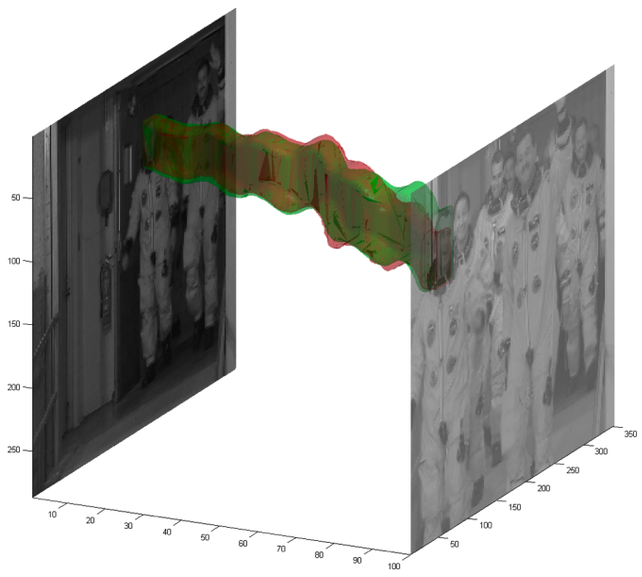
Sans géométrie

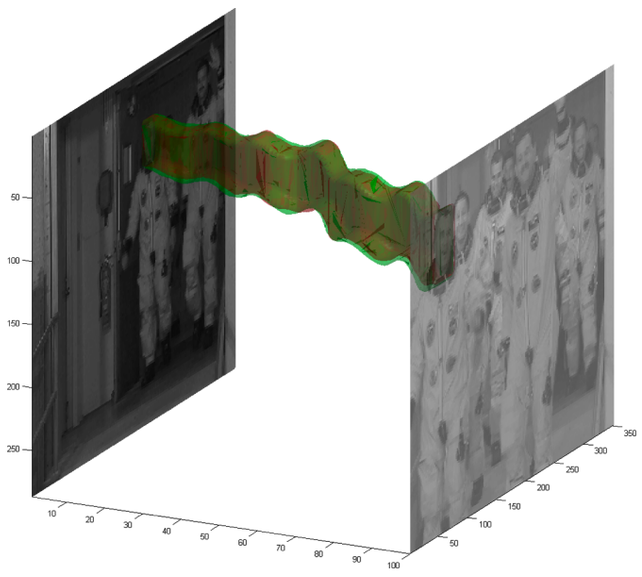


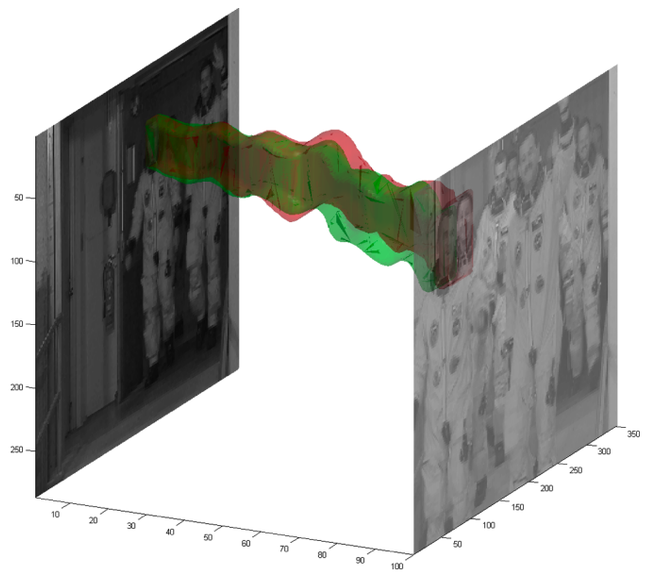




Avec géométrie

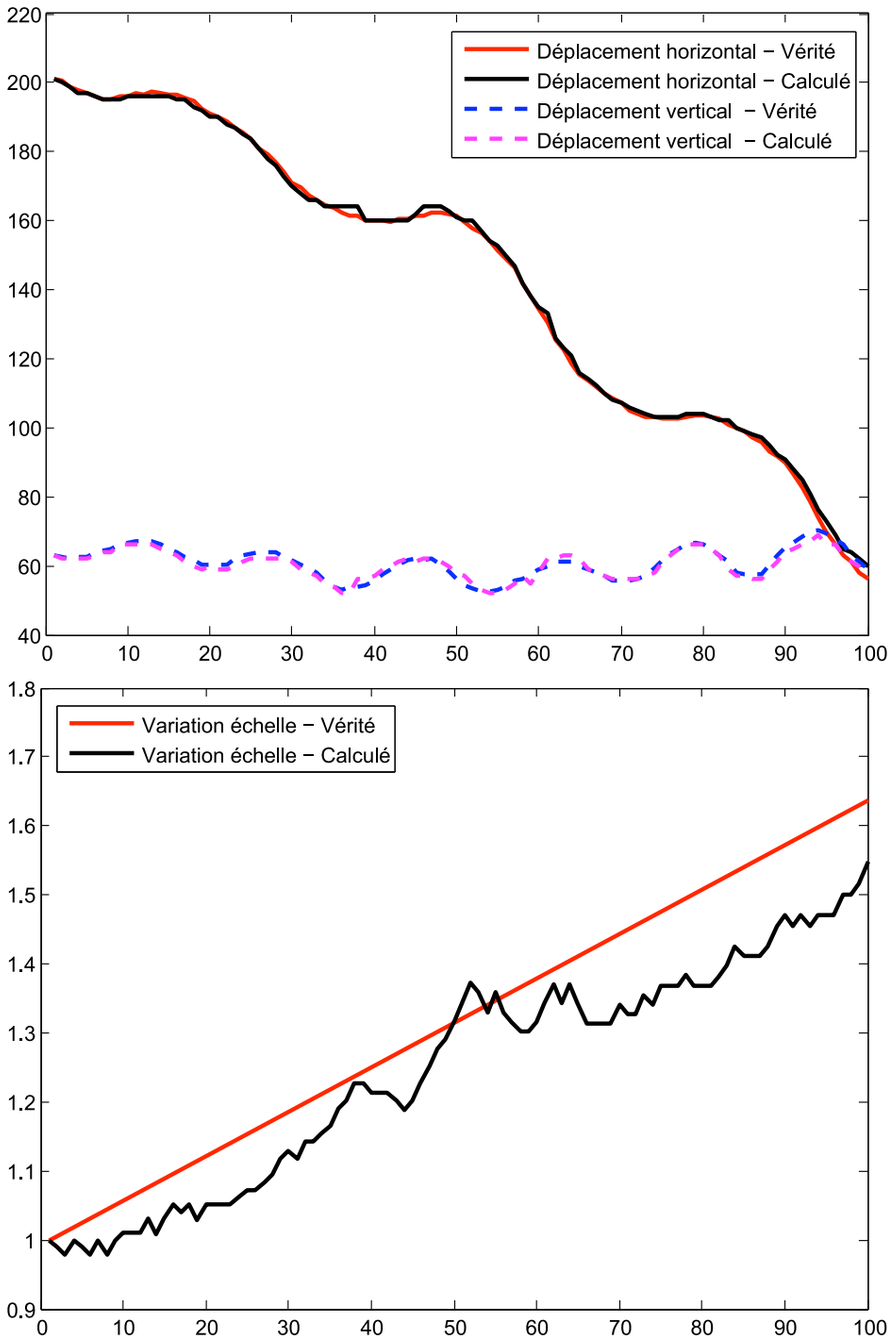






Expérimentations sur données réelles

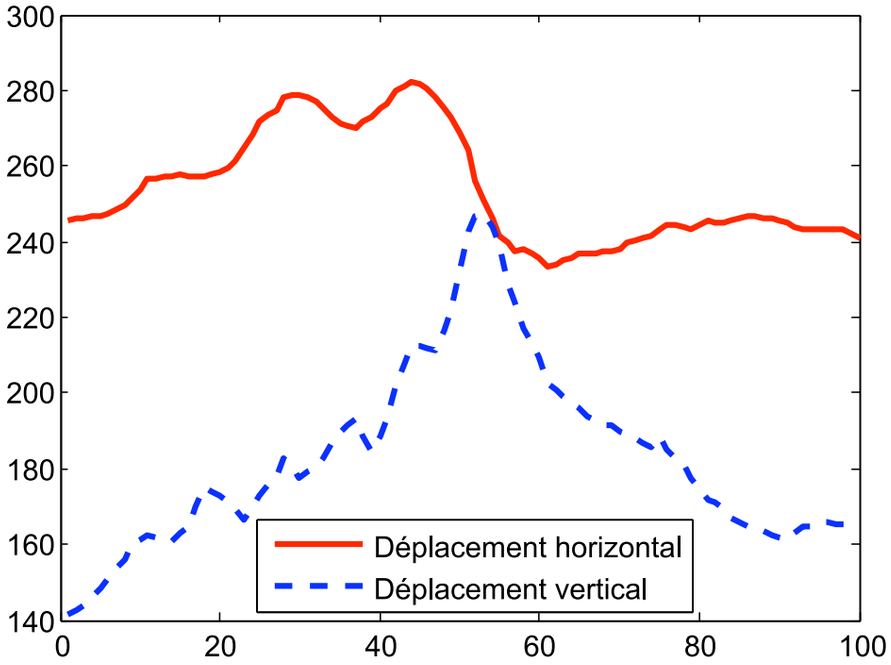
Erreur sur l'estimation des paramètres



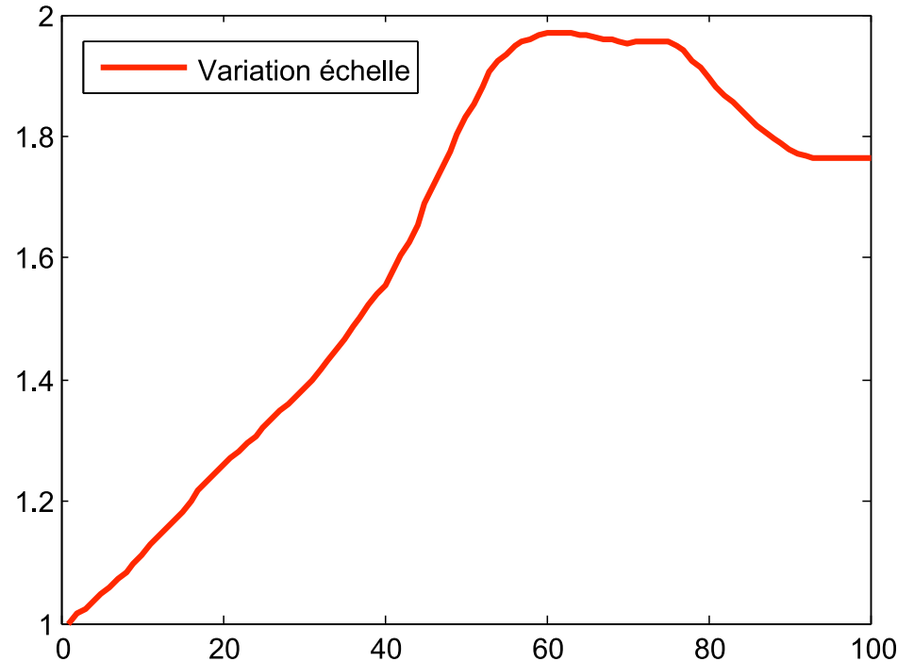
Expérimentations sur données réelles

- Séquence Poltergay
  - 720x576 pixels, 100 images
  - Objet : 9000 pixels
  - Difficultés
    - Mouvement complexe
    - Couleurs sombres

Évolution des paramètres - position



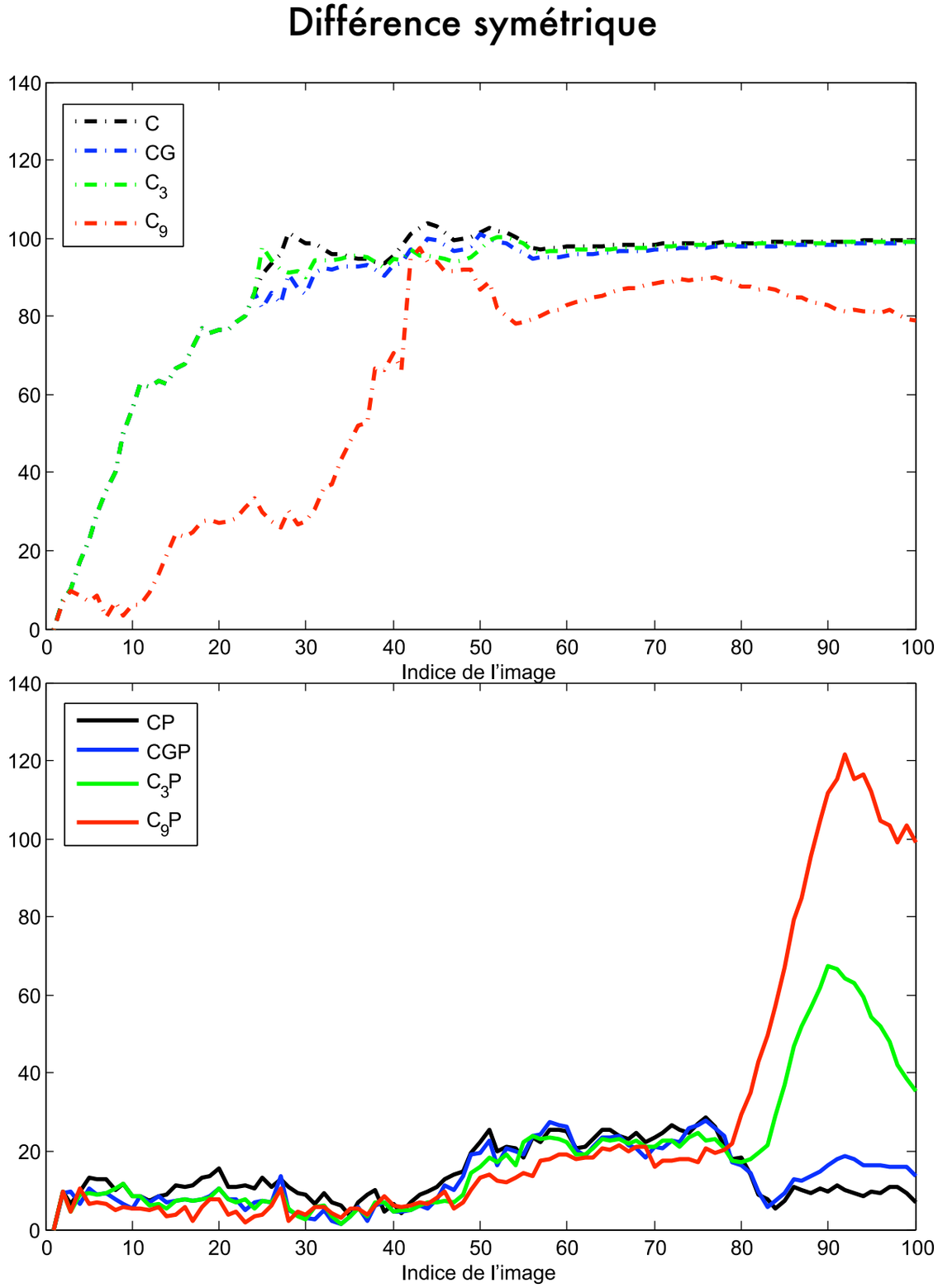
Évolution des paramètres - échelle



Expérimentations sur données réelles

- Descripteurs testées
  - C, CP, CG, CGP, C3, C3P, C9, C9P

- Résultats
  - Information géométrique vitale
  - Meilleur descripteur : CGP
  - Patch 9x9 donne les meilleurs résultats jusqu'à l'image 80



Expérimentations sur données réelles

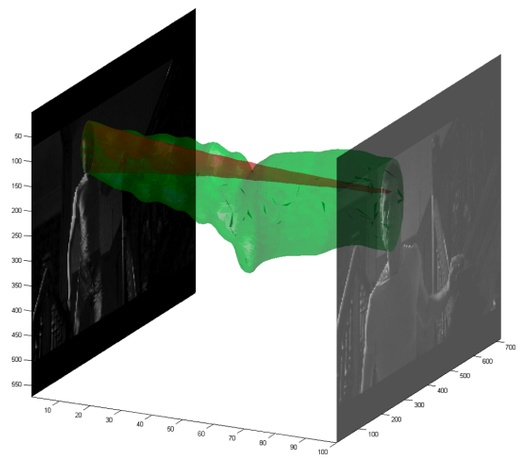
C

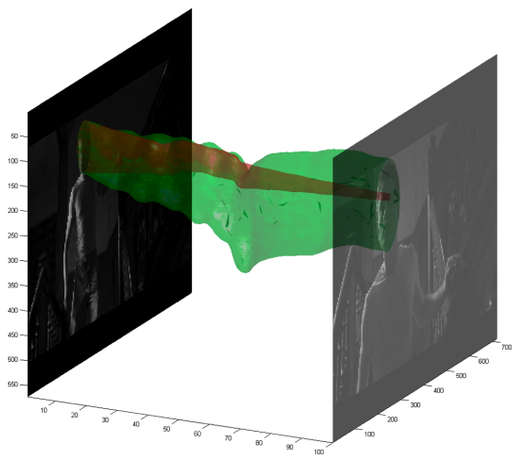
CG

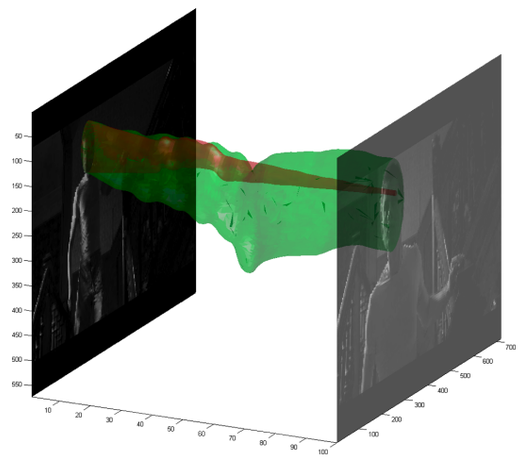
C3

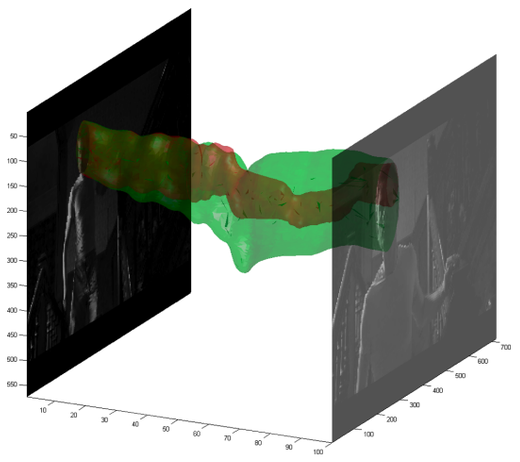
C9

Sans géométrie

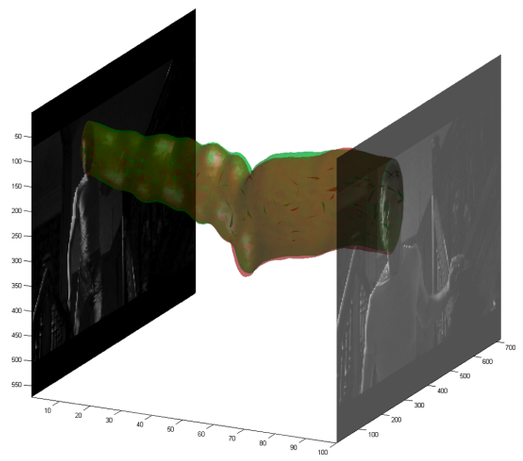


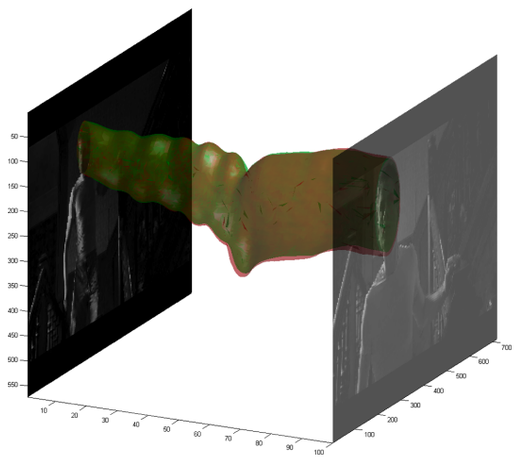


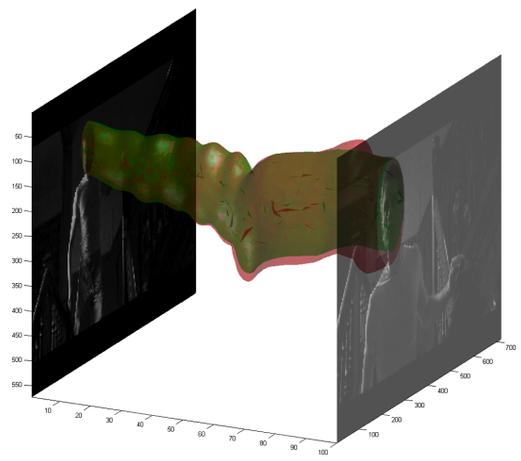


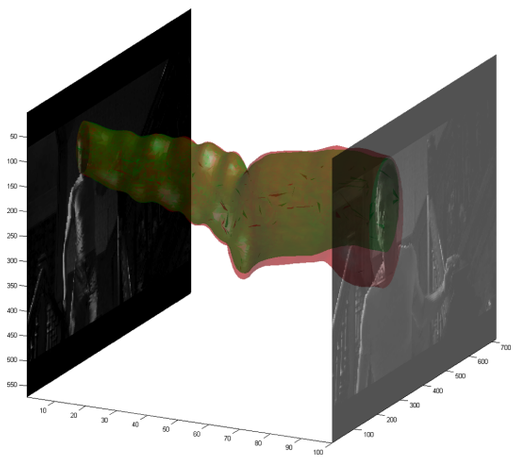


Avec géométrie



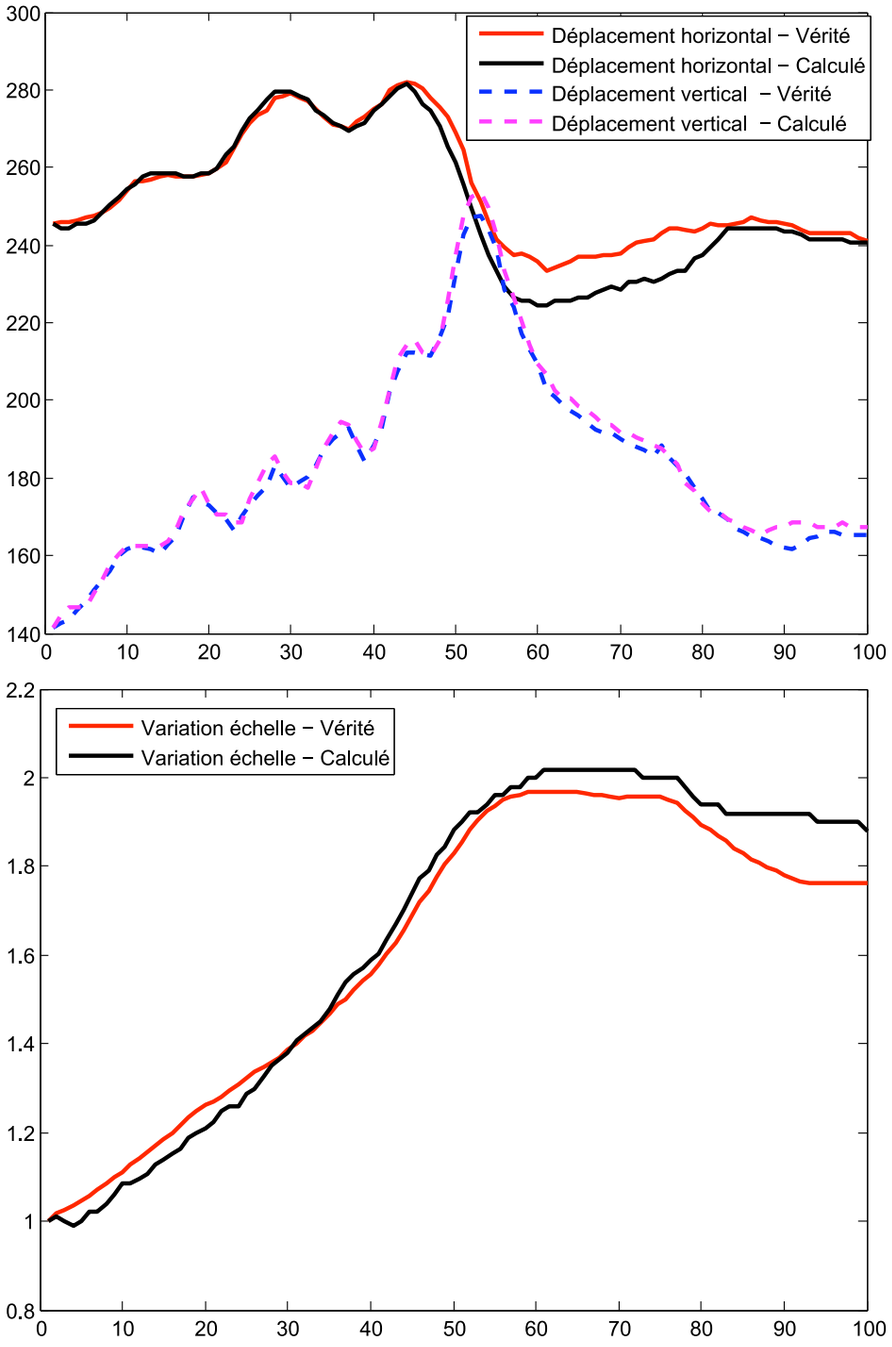


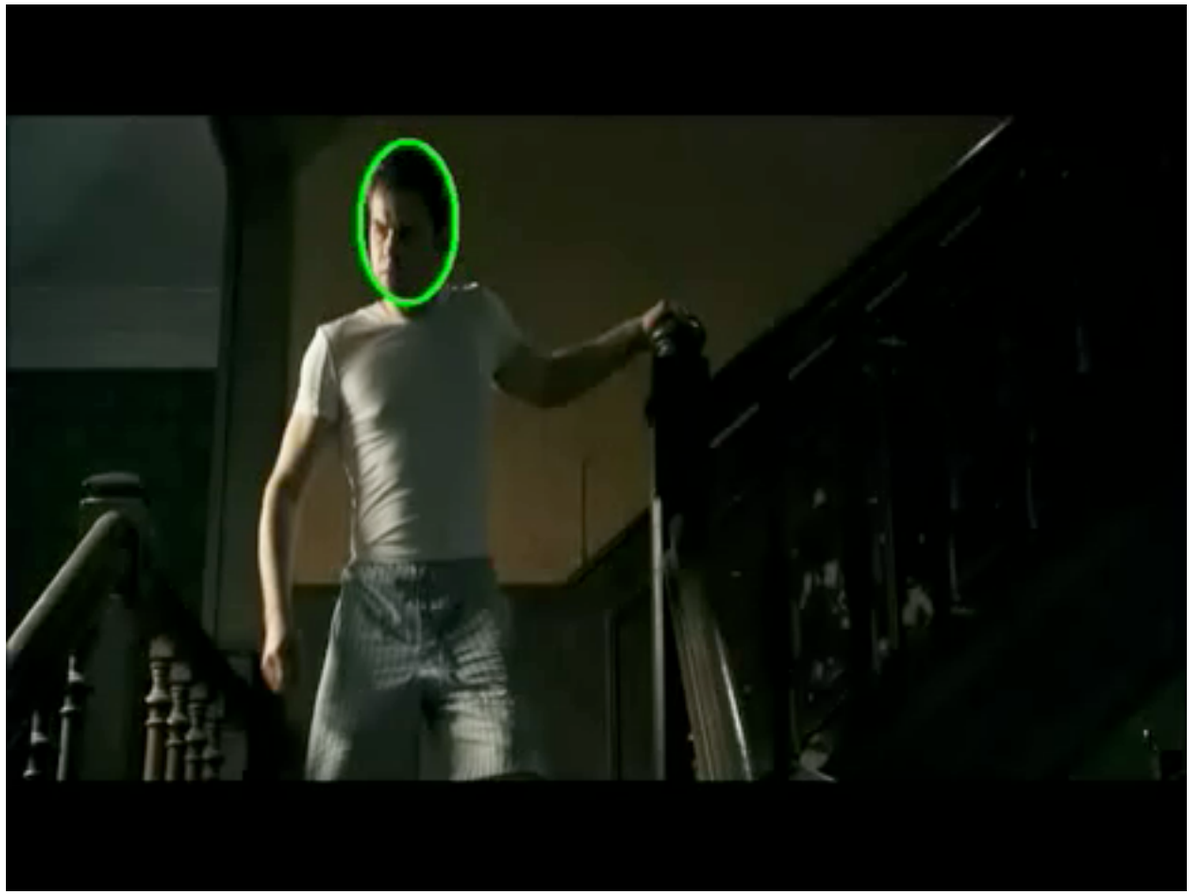




Expérimentations sur données réelles

Erreur sur l'estimation des paramètres





Expérimentations sur données réelles

- CPU vs. GPU
  - GPU plus rapide que CPU
  - Gain moins élevé
  - Temps dépend de la dimension et des parcours
  - ANN adapté à des données réelles
- Intérêt du GPU
  - Accélérer le suivi
  - Essayer de nouveaux descripteur

Crew

Descripteur	Dimension	ANN-C++	BF-CUDA	Gain
C	3	1m 33s	53s	1.8
CP	5	2m 05s	1m 05s	1.9
CG	5	2m 35s	1m 07s	2.3
CGP	7	4m 27s	1m 19s	3.3
C3	11	6m 40s	1m 17s	5.2
C3P	13	5m 43s	1m 12s	4.8

Poltergay

Descripteur	Dimension	ANN-C++	BF-CUDA	Gain
C	3	7m 30s	5m	1.5
CP	5	9m 45s	6m 10s	1.6
CG	5	11m 10s	5m 52s	1.9
CGP	7	1h 03m	42m	1.5
C3	11	32m	8m	3.6
C3P	13	1h 25m	1h 18m	1.1
C9	83	6h 59m	42m	9.8
C9P	85	27h 16m	5h 44m	5.3

VALORISATION DES TRAVAUX

- Projet Wired Smart (ANR/RIAM)
  - Realviz, ENS, CERTIS (ENPC), Mikros Image, CReATIVe (I3S)
  - Intégrer un algorithme de suivi d'objets dans le logiciel Coloris (Mikros Image)
    - CReATIVe : Algorithme de suivi + Implémentation GPU de la recherche des kPPV
    - Mikros Image : Intégration dans Coloris

- Résultats (descripteur UVxy, k=3)
  - Cercle rouge, 100 images
    - BF-C : 77 minutes 40 secondes
    - BF-CUDA : 21 secondes (**gain = 221X**)
  - Poltergay, 100 images
    - BF-C : 560 minutes
    - BF-CUDA : 1 minutes 10 secondes (**gain = 480X**)

Conclusion & perspectives

- Résultats
  - Méthode exhaustive (GPU) plus rapide que ANN (CPU)
  - Gain très important sur données synthétiques : 150X ANN-C++ et 300X BF-C
  - Gain significatif pour le suivi : 2-5X ANN-C++
  - Autres applications : Indexation d'images (10X), débruitage d'images (10X)
  - Librairie CUBLAS (CUDA BLAS) : accélération en haute dimension (2X BF-CUDA pour d=100)
- Programmation GPU
  - Révolution pour le calcul scientifique (GPGPU - API NVIDIA CUDA)
  - **Outil** adapté aux problèmes (algorithmes) parallélisables (e.g. kPPV)
  - Accélérer les calculs / Nouvelles approches / Lever certaines limitations
- Perspectives
  - Étudier d'autres algorithmes de recherche des kPPV : kd-tree, vp-tree, k-means, etc.
  - Appliquer implémentation kPPV GPU à d'autres applications (haute dimension)

Questions ???

- A -

SUIVI D'OBJETS & TRAJECTOIRES DE POINTS SAILLANTS

SUIVI D'OBJETS & TRAJECTOIRES

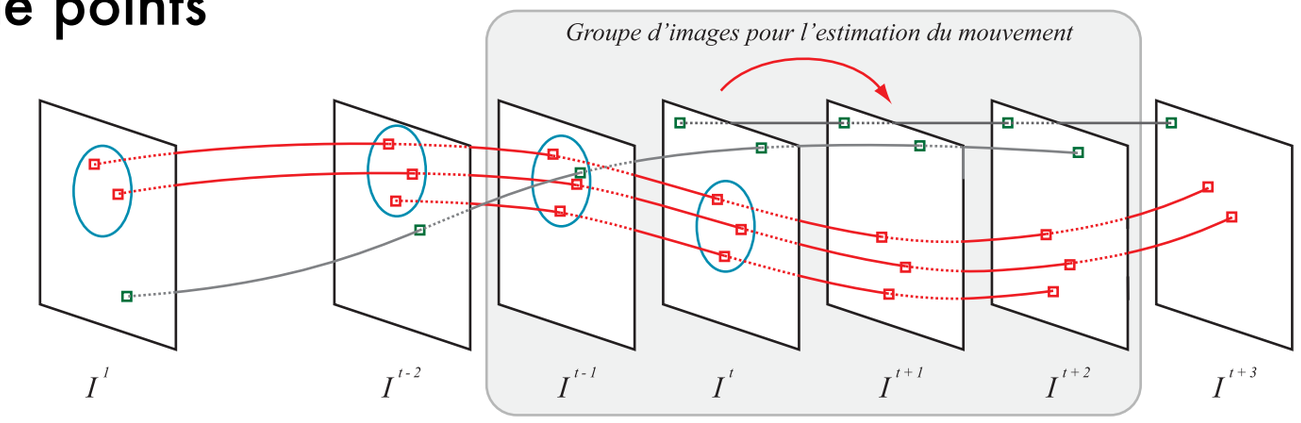
- Problème
  - Contour initial à l'image I1
  - Calculer le contour sur les autres images

- Objet = points d'échantillonnage + courbe

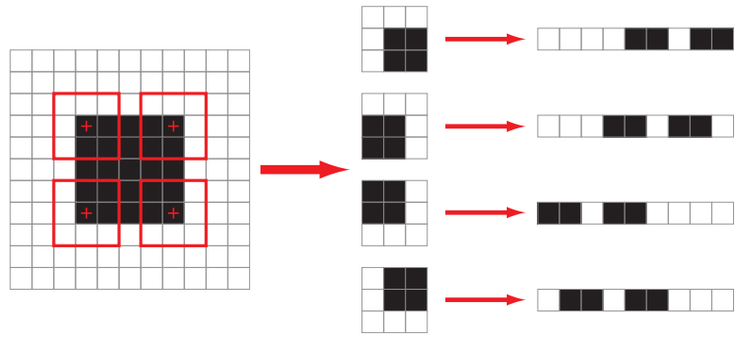
- Approche proposée
  - Construction de trajectoires de points saillants
  - Analyse de ces trajectoires pour l'estimation du mouvement
  - GOP + pondération spatio-temporelle

TRAJECTOIRES DE POINTS SAILLANTS

- Construction de trajectoires
  - Appariement de points saillants entre 2 images consécutives
  - Mise bout-à-bout des couples de points



- Description locale
  - Extraction de points saillants (Harris-Stephens)
  - Construction d'un descripteur (Patch)
  - Définition d'une mesure de similarité (SAD)

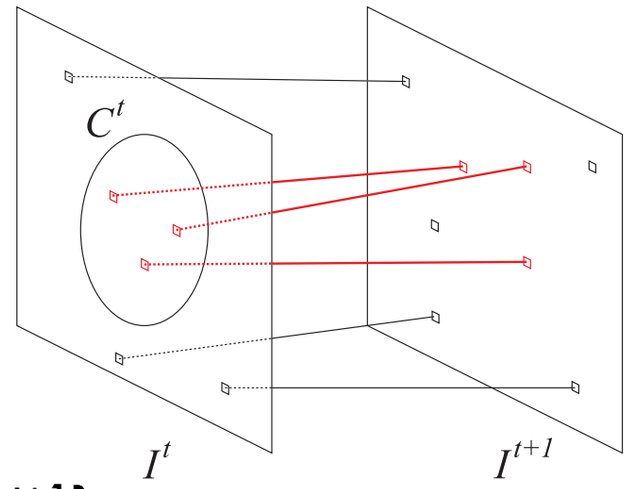


$$A(x,y) = \begin{bmatrix} \left(G_\sigma * \left(\frac{\partial I}{\partial x}\right)^2\right)(x,y) & \left(G_\sigma * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) \\ \left(G_\sigma * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) & \left(G_\sigma * \left(\frac{\partial I}{\partial y}\right)^2\right)(x,y) \end{bmatrix}$$

$$\begin{aligned} C(x,y) &= \lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2 \\ &= \det(A(x,y)) - \kappa.\mathrm{trace}^2(A(x,y)) \end{aligned}$$

MOUVEMENT GLOBAL ENTRE 2 IMAGES

- Algorithme
  - Contour de référence : C^t
  - Extraction des couples de points $\{p_i^t, p_i^{t+1}\}$
  - Estimation du mouvement M^t de C^t à partir des couples $\{p_i^t, p_i^{t+1}\}$
  - Calcul de C^{t+1} : application de M^t à C^t



- Estimation du mouvement
  - Mouvement affine
  - Minimisation d'un M-estimateur

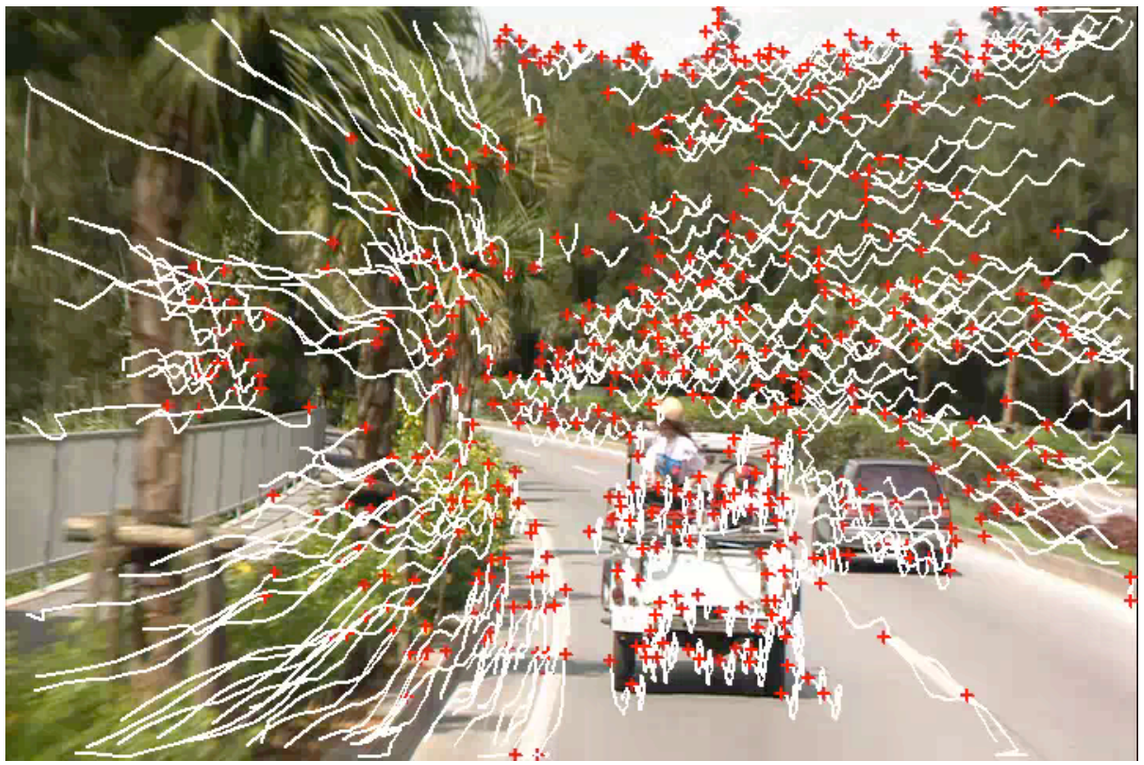
$$M^t = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}$$

$$\widehat{M^t} = \arg\min_M \sum_{i \in [1, N_p]} f(\|\epsilon_{i,M}^t\|)$$

$$\epsilon_{i,M}^t = p_i^{t+1} - M p_i^t$$

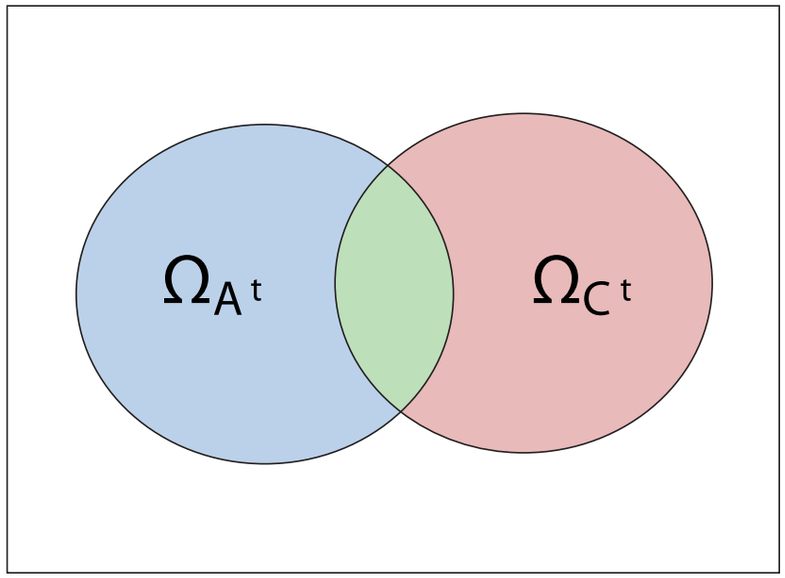
  - Simplexe ou recuit simulé

- Driving
  - 720 x 480 pixels
  - 30 images
  - Suivi précis malgré l'occultation
  - Comment exprimer l'erreur de suivi?



Critère de qualité

- Image It
  - Contour animateur (groundtruth) : Ω_{A^t}
  - Contour calculé : Ω_{C^t}



- Erreur de suivi à l'image I^t

$$E(\Omega_{A^t}, \Omega_{C^t}) = 100 \frac{\#(\Omega_{A^t} \Delta \Omega_{C^t})}{\#\Omega_{A^t}}$$

- Différence symétrique

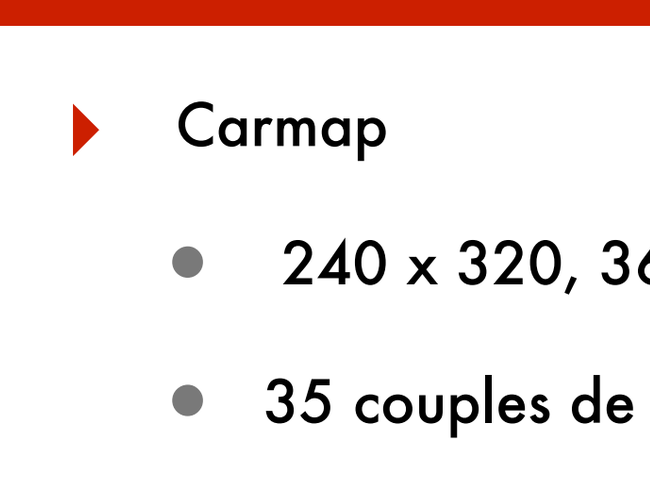
$$\Omega_{A^t} \Delta \Omega_{C^t} = \{x | (x \in \Omega_{A^t}) \vee \vee (x \in \Omega_{C^t})\}$$

- Erreur de suivi sur la séquence

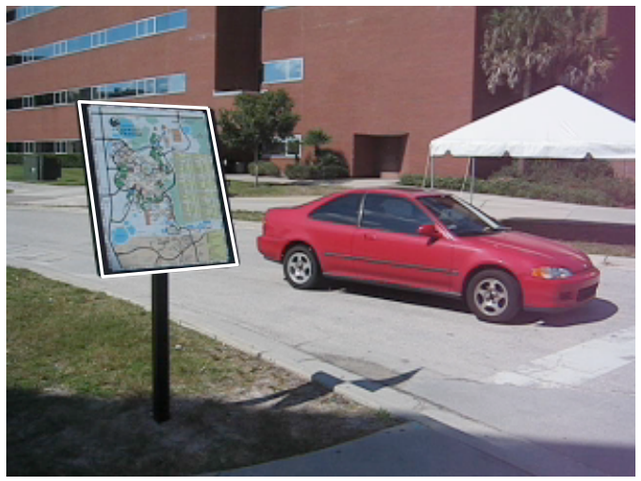
$$E(A, C) = \frac{1}{N_V} \sum_{t=2}^{N_V} E(A^t, C^t)$$

RÉSULTATS

- Carmap
  - 240 x 320, 36 images
  - 35 couples de points
  - Erreur de suivi : 4% de pixels mal-classés



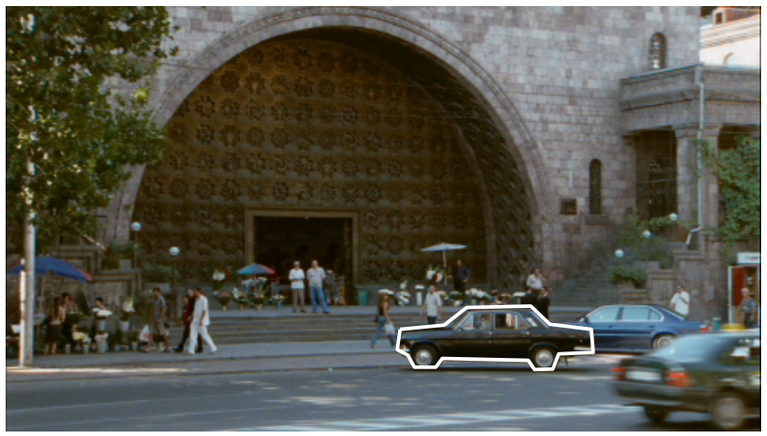
I^1



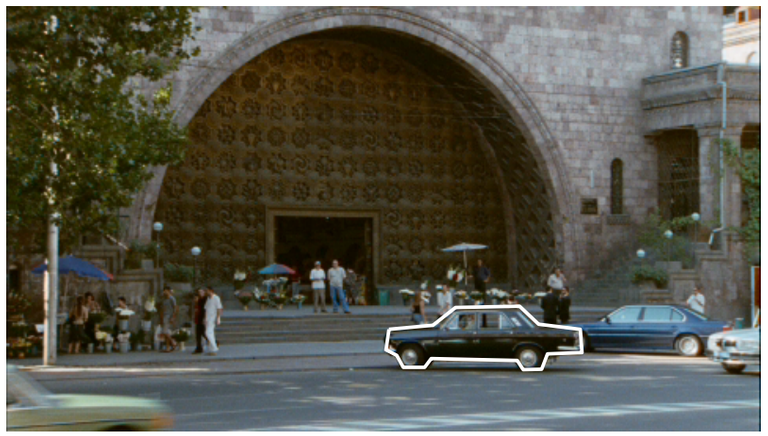
I^{36}

RÉSULTATS

- Arménie (“Voyage en Arménie”, Guédiguian, 2006)
  - 480 x 270 pixels, 40 images
  - **11 couples de points**
  - Erreur de suivi : 42% de pixels mal-classés



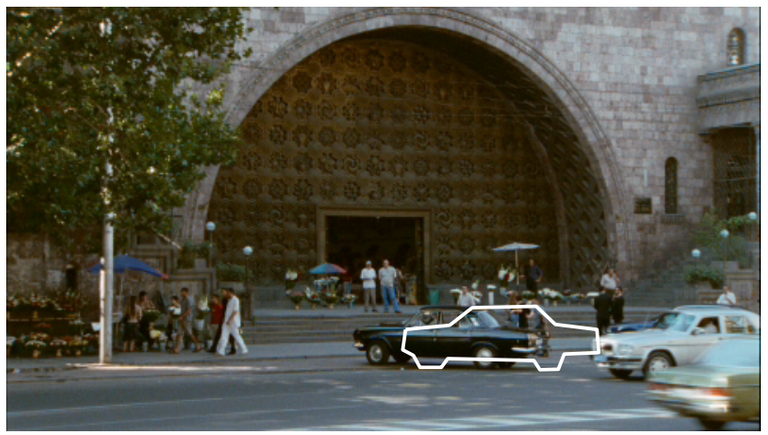
I^1



I^8

I^{16}

I^{24}



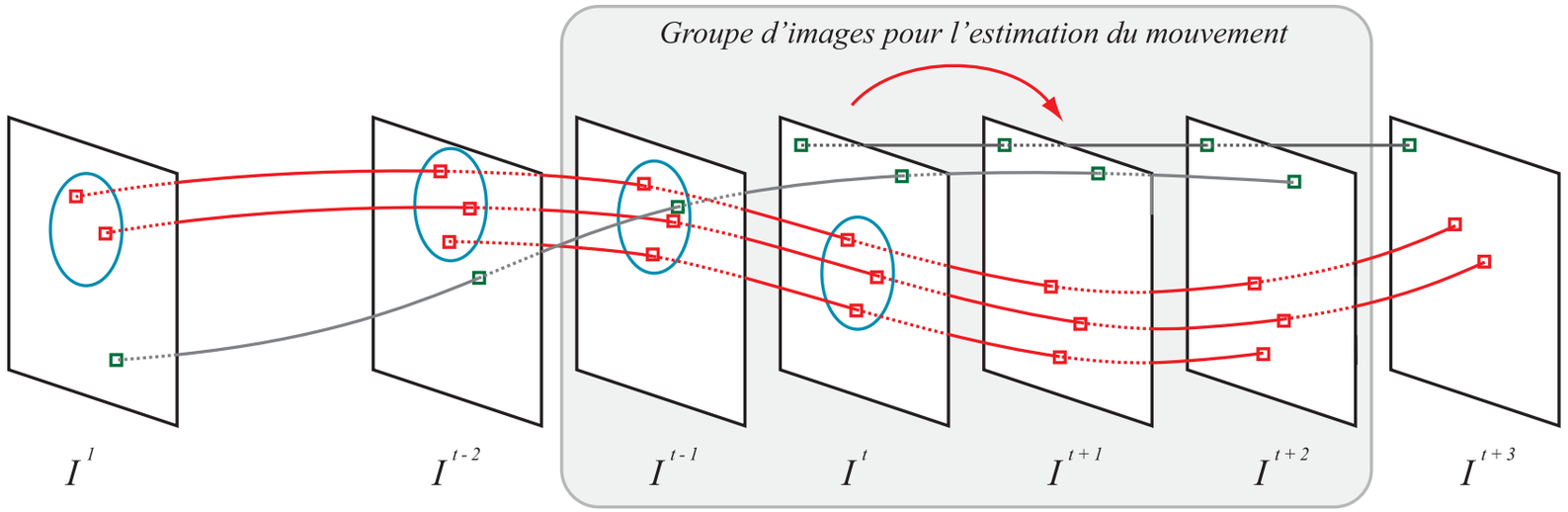
I^{32}

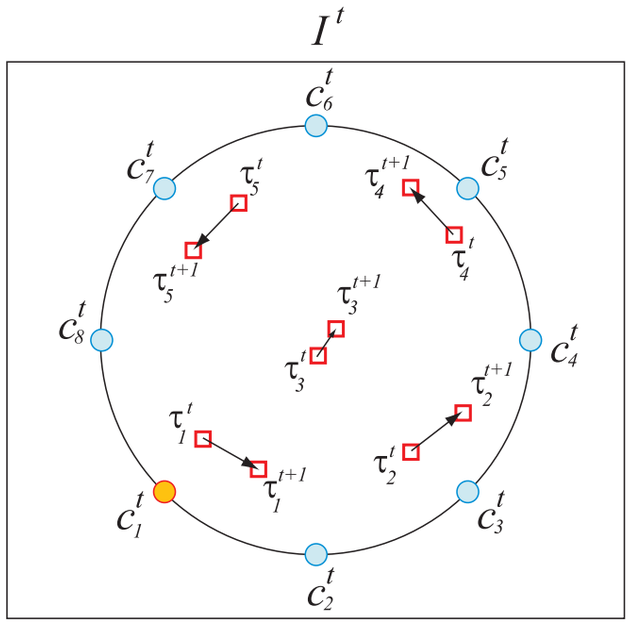
I^{40}

Mouvement local sur un GOP glissant

- Inconvénients
  - Qualité de l'estimation du mouvement dépend du nombre de couple de points
  - Mouvement global uniquement entre 2 images

- Solution
  - Extraction des couples sur un groupe d'images + pondération temporelle
  - Estimation locale du mouvement = pondération spatiale





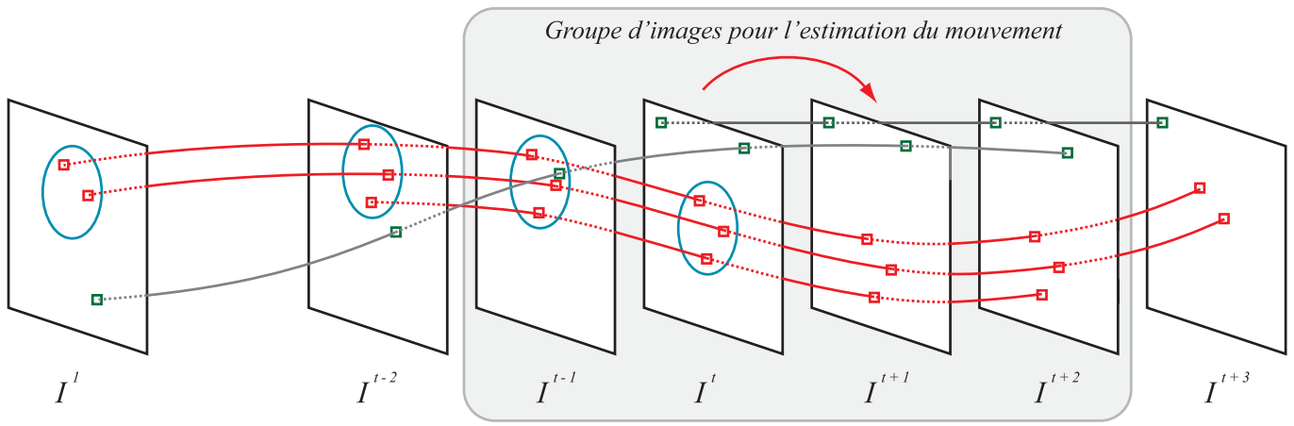
MOUVEMENT LOCAL SUR UN GOP GLISSANT

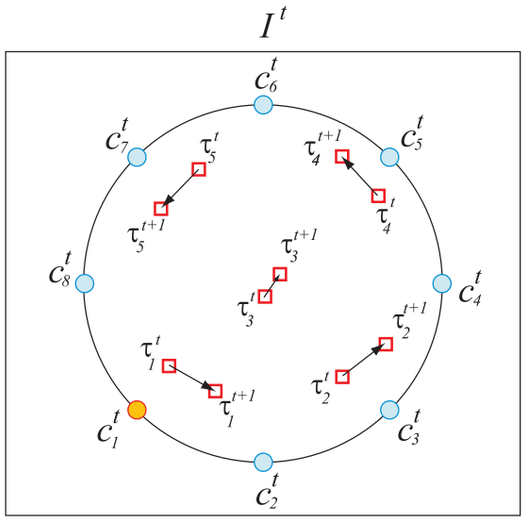
- Estimation du mouvement

$$\widehat{M}_k^t = \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t-g-1}^{t+g-1} \alpha^{t,j} \beta_{k,i}^t f(\|\epsilon_{i,M}^j\|)$$

$$\epsilon_{i,M}^t = p_i^{t+1} - M p_i^t$$

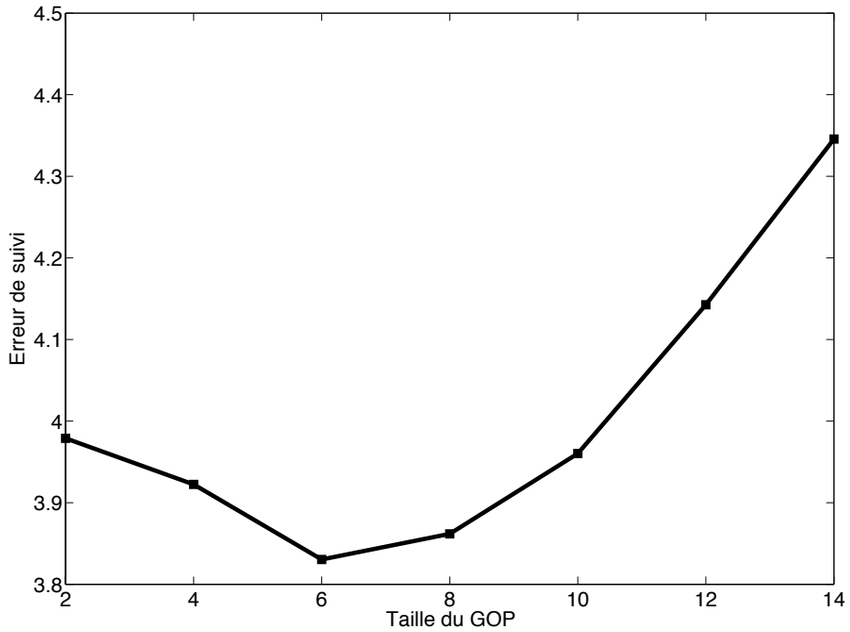
- Couple de points $\{p_i^i, p_i^{i+1}\}$
  - Pondération temporelle $\alpha^{t,j} = \varphi(t-j)$
  - Pondération spatiale $\beta_{k,i}^t = \phi(\mathcal{D}(c_k^t, \{p_i^j, p_i^{j+1}\}))$



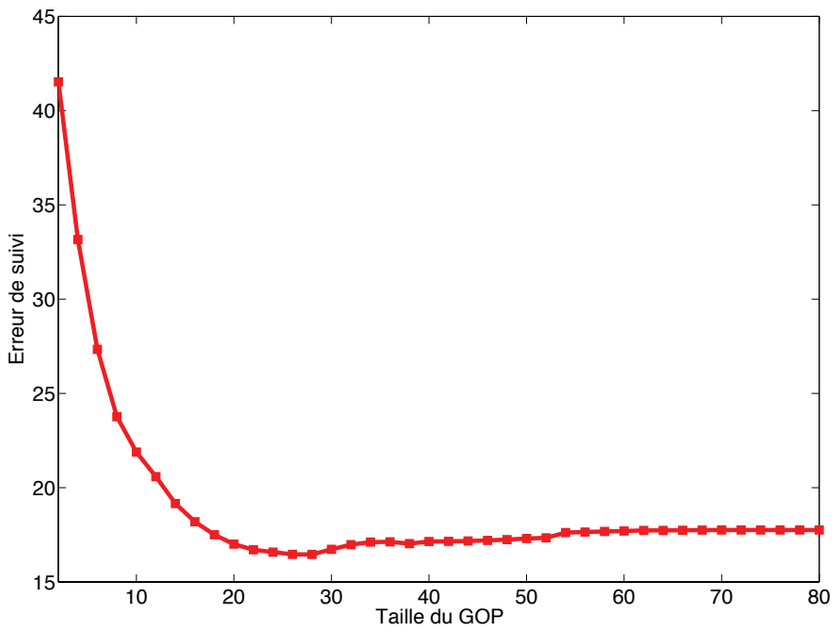


RÉSULTATS

- Carmap : GOP = 6 → erreur < 4%



- Arménie : GOP = 27 → erreur = 17%



RÉSULTATS

- Football
  - 352 x 288 pixels
  - 20 images
  - Difficulté : mouvement rapide

- Crew
  - 352 x 288 pixels
  - 100 images
  - Difficultés
    - Variation intensité (flashs)
    - Mouvement complexe

RÉSULTATS

- Speed
  - 646 x 272 pixels
  - 100 images
  - Difficulté : occultation