

Rapport de projet  
Détection de flot optique

Département de Génie Mathématique  
Semestre 8 - 2016/2017



*Auteurs :*

Emeric QUESNEL  
Timothée SCHMODERER

*Référente :*  
Carole LE GUYADER

## Table des matières

# 1 Introduction

Parmi les nombreux domaines de recherche en Traitement d'Images, une place importante est réservée à la détection de mouvements d'objets sur des images (sur une séquence vidéos par exemple). L'importance considérable que représente l'étude du comportement dynamique d'objets se justifie à travers la multitude d'applications reposant sur ces méthodes. Plus concrètement, les progrès de l'informatique ont ouvert de nouvelles thématiques de recherche qui exploitent pleinement les méthodes développées dans le cadre de l'analyse de mouvement des objets : citons le thème de la vision artificielle (ou vision par ordinateur) et plus particulièrement le sujet de la conduite assistée qui repose sur la détection en temps réel d'obstacles (piétons notamment) et l'interprétation des variations du mouvement de la scène observée, pouvant par exemple servir à anticiper un freinage brutal (et donc un éventuel accident de la route imminent) en palliant la lenteur du temps de réaction du conducteur et en actionnant le système de freinage avant le conducteur lui-même (un système de freinage anticipé équipe déjà certains modèles de voiture Tesla). Dans une autre perspective, la déformation de tissus et d'organes ou l'estimation du débit sanguin constituent des exemples d'applications biomédicales dans lesquelles la prise en compte de l'analyse du mouvement entre images joue un rôle important. La liste n'est évidemment pas exhaustive et nous renvoyons le lecteur à [?] pour de plus amples informations.

Initiées dès le début des années 1980 par Berthold K.P. Horn et Brian G. Schunck, l'analyse du mouvement entre 2 images, et plus précisément le calcul du flot optique entre ces images, s'est considérablement développée depuis. Pour le projet qui nous concernait, celui-ci consistait à implémenter en Matlab différentes méthodes d'estimation du flot optique entre 2 images données en partant de l'algorithme original des 2 auteurs. Avant de rentrer dans le vif du sujet, précisons sans plus attendre la signification de cette notion de flot optique qui nous a accompagnés tout au long de ce projet.

## Définition 1.1 : *Flot Optique* ([?])

Le flot optique (ou flux optique) est le mouvement apparent des objets, surfaces et contours d'une scène visuelle, causé par le mouvement relatif entre un observateur (l'œil ou une caméra) et la scène.

Finissons par préciser que ce projet est proposé par M. Gwenael Mercier<sup>1</sup> aux étudiants de deuxième année de Polytechnique sur le site de M. Grégoire Allaire : [http://www.cmap.polytechnique.fr/~allaire/cours\\_X\\_annee2.html](http://www.cmap.polytechnique.fr/~allaire/cours_X_annee2.html).

1. Lien vers le sujet : <http://www.cmap.polytechnique.fr/~allaire/map431/MiniProjets/sujets2014/sujet12.pdf>

## 2 Prérequis

### 2.1 Équations d'Euler - Lagrange [?]

#### Définition 2.1 : Principe de moindre Action

Le principe de moindre action est l'hypothèse physique selon laquelle la dynamique d'une quantité physique (position, vitesse, accélération ...) peut se déduire à partir d'une unique grandeur appelée action en supposant que les valeurs de la dynamique permettent à celle-ci d'avoir une valeur optimale.

Toutes les équations de la mécanique peuvent être reformulées à partir de ce principe. Les équations d'Euler - Lagrange en sont une conséquence. Celles-ci ont su démontré leur efficacité dans de très nombreux problèmes : optique, mécanique céleste, problème isopéri-métrique, problème brachistochrone ...

Soit un système physique décrit par une certaine fonction  $L$ , appelée *Lagrangien*, dépendant de certains paramètres :

$$L : \begin{cases} \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R} & \longrightarrow \mathbb{R} \\ (q(t), \dot{q}(t), t) & \longmapsto L(q(t), \dot{q}(t), t) \end{cases}$$

Où :

$$q(t) = \begin{pmatrix} q_1(t) \\ \vdots \\ q_n(t) \end{pmatrix} \quad \dot{q}(t) = \begin{pmatrix} \dot{q}_1(t) \\ \vdots \\ \dot{q}_n(t) \end{pmatrix}$$

représente l'état du système à l'instant  $t$  et  $n$  est le nombre de degré de liberté du système. Nous définissons alors l'*action* du système par l'intégrale de  $L$  entre deux instants donnés :

$$A = \int_{t_0}^{t_1} L(q(t), \dot{q}(t), t) dt \quad (1)$$

Nous supposons que  $L \in C^1(\mathbb{R}^n \times \mathbb{R}^n \times [t_0, t_1])$ , que l'action est finie et continue.

#### Théorème 2.2 : Équations d'Euler - Lagrange

Si l'action  $A$  définie ci-dessus (??) est minimale alors :

$$\forall i \in \llbracket 1; n \rrbracket \quad \frac{\partial L}{\partial q_i}(q(t), \dot{q}(t), t) - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i}(q(t), \dot{q}(t), t) = 0$$

**Preuve :**

Une démonstration et un exemple sont donnés à l'annexe ??.

□

**Remarque :** Dans ce projet, nous utiliserons une variante des équations d'Euler-Lagrange, mais le principe d'utilisation est identique :

### Théorème 2.3 : Variante des équations d'Euler-Lagrange

Si l'action s'écrit sous la forme :

$$L[u, v] = \int_{\Omega} L \left( x, y, u, v, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial v}{\partial x}, \frac{\partial v}{\partial y} \right) d\mu$$

Alors le système des équations d'Euler-Lagrange s'écrit sous la forme :

$$\begin{cases} \frac{\partial L}{\partial u} - \frac{\partial}{\partial x} \left( \frac{\partial L}{\partial (\frac{\partial u}{\partial x})} \right) - \frac{\partial}{\partial y} \left( \frac{\partial L}{\partial (\frac{\partial u}{\partial y})} \right) = 0 \\ \frac{\partial L}{\partial v} - \frac{\partial}{\partial x} \left( \frac{\partial L}{\partial (\frac{\partial v}{\partial x})} \right) - \frac{\partial}{\partial y} \left( \frac{\partial L}{\partial (\frac{\partial v}{\partial y})} \right) = 0 \end{cases}$$

## 2.2 Convolution d'images

Une image n'est rien d'autre qu'un tableau de nombre (voir Figure ??). Où dans chaque case est noté le niveau de gris.

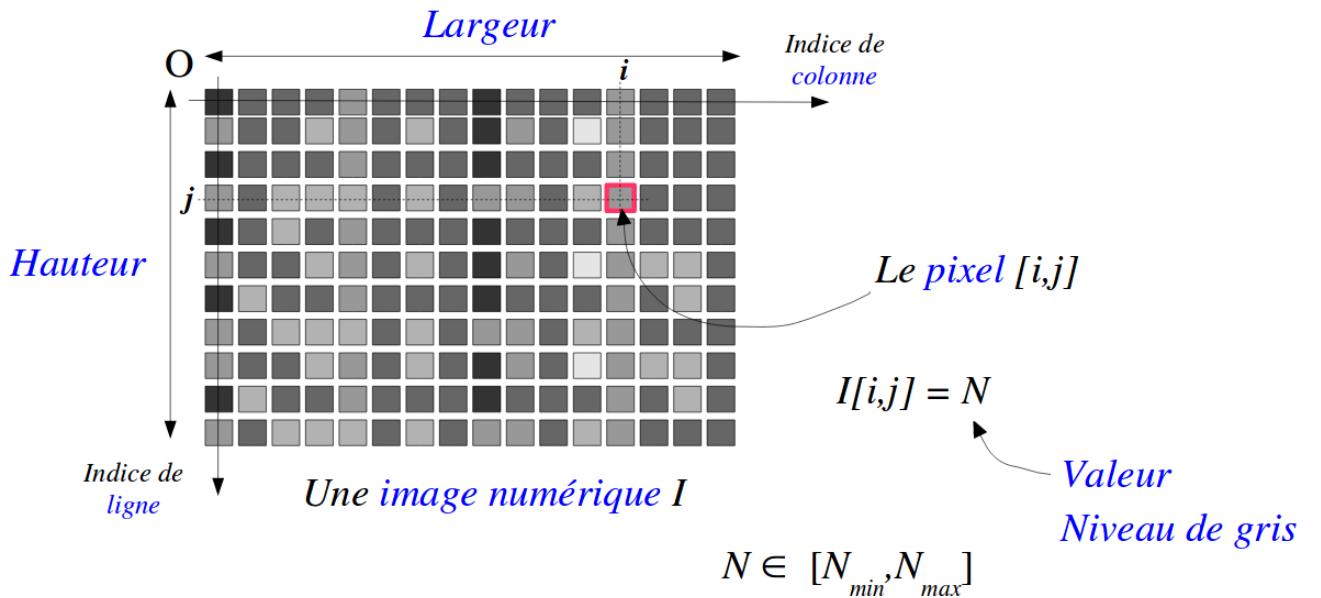


FIGURE 1 – Une image

A plusieurs reprise dans ce projet, nous avons besoin d'obtenir des informations sur un pixel de l'image à partir d'informations sur ses voisins (cela correspond à des opérations de dérivations). Ces calculs, fastidieux en apparence se réduiront à une "simple" opération de convolution entre la matrice représentant l'image et une matrice bien choisie représentant l'opération de dérivation.

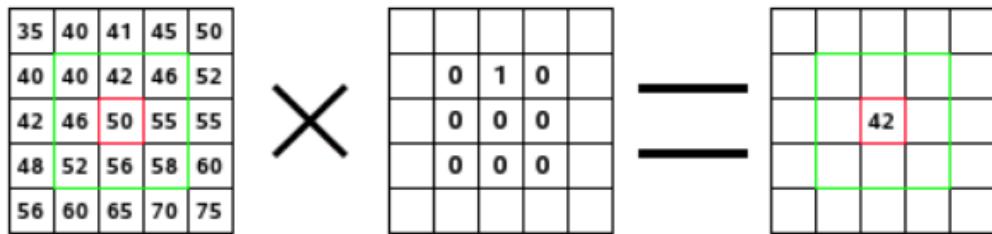


FIGURE 2 – Illustration de la convolution. De gauche à droite : l'image (représentée par ses niveaux de gris), le noyau de convolution, le résultat

Une matrice carré, appelée *noyau de convolution* agit sur l'image de la manière suivante : Pour chaque pixel, le noyau de convolution attribue une nouvelle valeur à partir d'une combinaison linéaire des voisins.

A noter que pour les pixels au bord, les valeurs en dehors de l'image sont prises égales à celles du pixel voisin, ce qui correspond à une condition de Neumann homogène sur les couleurs au bords.

**Exemple :** Appliquons le noyau de convolution de Sobel :  $\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$  à une image.

Ce filtre fait ressortir les discontinuités spatiales dans une image, c'est à dire les contours. Voici les résultats que l'on obtient avec le code ?? : Analysons ce qu'il s'est passé sur



FIGURE 3 – Application du filtre de Sobel sur une image

l'image. Voici une petite partie de l'image telle qu'elle est traitée par le programme.

$$\begin{pmatrix} 98 & 135 & 126 \\ 93 & 139 & 136 \\ 138 & 175 & 173 \end{pmatrix}$$

Cette matrice correspond à l'image suivante (Figure ??). On applique alors le filtre sur le pixel central :

$$\begin{pmatrix} 98 & 135 & 126 \\ 93 & 139 & 136 \\ 138 & 175 & 173 \end{pmatrix} * \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} = \begin{pmatrix} \times & \times & \times \\ \times & -167 & \times \\ \times & \times & \times \end{pmatrix}$$



FIGURE 4 – Image extraite

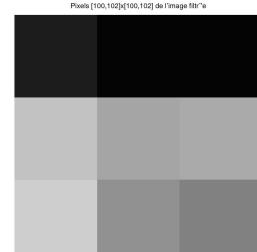


FIGURE 5 – Image filtrée

Voici le code utilisé pour générer cet exemple :

```
img = double(imread('einstein.jpg'));
%filtre de Sobel
s = [1 2 1; 0 0 0; -1 -2 -1];
%Convolution
imgSobel=conv2(img,s,'same');

figure;
subplot(1,2,1)
imshow(img,[0,255])
subplot(1,2,2)
imshow(imgSobel,[0,255]);
```

Code 1 – Code de mise en place du filtre de Sobel

### 3 Méthode de Horn et Schunck

#### 3.1 Le problème

Soit  $I(x, y, t)$  l'intensité lumineuse d'un point de coordonnées  $(x, y)$  à l'instant  $t$  d'une séquence vidéo en noir et blanc, c'est à dire une application de  $[0, 1]^2 \times [0, T]$  dans  $\llbracket 0 \dots 255 \rrbracket$ . L'objectif est de calculer de façon approchée la trajectoire  $(x(t), y(t))$  d'un point matériel au long de la séquence. Pour cela, nous allons chercher à approximer le champ de vecteurs  $h$  suivant (le flot optique) :

$$h = \left( \frac{dx}{dt}, \frac{dy}{dt} \right) = (u, v)$$

Nous faisons l'hypothèse fondamentale que l'intensité lumineuse des points est invariante au cours du temps :

$$\frac{dI}{dt} = 0$$

Nous faisons également l'hypothèse que le déplacement des objets est petit par rapport à l'image, et ce pour qu'il y ait un sens à utiliser des dérivées temporelles et spatiales de l'image. Nous renvoyons à la section ?? pour la prise en charges des grands déplacements. Un calcul direct par la règle de la chaîne montre que ces hypothèses entraînent :

$$\boxed{\nabla I \cdot h + \frac{\partial I}{\partial t} = 0} \quad (2)$$

De cette équation, nous pouvons en conclure que la composante du vecteur déplacement dans le sens du gradient d'intensité est :

$$-\frac{I_t}{\sqrt{I_x^2 + I_y^2}} \quad \text{Où l'indice marque la variable de différentiation}$$

Cependant, cela ne nous donne pas la composante du déplacement dans le sens des iso-luminosité (c'est à dire orthogonalement au gradient). Ainsi, le champ de déplacement ne peut pas être calculé sans ajout de nouvelles contraintes.

Les auteurs Horn et Schunck [?] ajoutent l'hypothèse de régularité du flot de vecteurs pour obtenir une solution de l'équation (??). Ils proposent de minimiser l'énergie suivante sur le domaine  $\Omega$  de l'image :

$$\boxed{J(u, v) = \int_{\Omega} (I_x u + I_y v + I_t)^2 + \alpha^2(|\nabla u|^2 + |\nabla v|^2) dx \quad u, v \in H^1(\Omega)} \quad (3)$$

Où,  $\alpha$  est un coefficient de pénalisation, plus  $\alpha$  est grand plus la régularité du champ de déplacement obtenue sera grande. Observons que si  $\alpha = 0$  alors le minimum de la fonctionnelle est exactement atteint lorsque  $(u, v)$  vérifient (??).

Le domaine  $\Omega$  étant compact (rectangle de l'image ici),  $J$  étant continue (car le flot est supposé régulier), la fonctionnelle  $J$  atteint bien son minimum et ce minimum est unique car  $J$  est coercive.

Appliquons les équations d'Euler-Lagrange à  $J$  (??). Nous remarquons que celui-ci est exactement sous la forme du théorème ?? . Un calcul immédiat nous donne alors :

$$\begin{cases} I_x (I_x u + I_y v + I_t) - \alpha^2 (\Delta u) = 0 \\ I_y (I_x u + I_y v + I_t) - \alpha^2 (\Delta v) = 0 \end{cases} \quad (4)$$

## 3.2 Discrétisation de la méthode

Nous allons donc devoir approximer les dérivées partielles de  $I$  (ce sont des constantes du problème) et approximer le laplacien de  $u$  et  $v$  par un schéma aux différences finies.

### 3.2.1 Approximation des dérivés partielles de $I$

Nous nous servons de la différence d'intensité entre les pixels des deux images autour d'un même pixel.

$$\begin{aligned} I_x(i, j) &= \frac{1}{4} (I_{1,i,j+1} - I_{1,i,j} + I_{1,i+1,j+1} - I_{1,i+1,j} + I_{2,i,j+1} - I_{2,i,j} + I_{2,i+1,j+1} - I_{2,i+1,j}) \\ \forall i, j \in \Omega \quad I_y(i, j) &= \frac{1}{4} (I_{1,i+1,j} - I_{1,i,j} + I_{1,i+1,j+1} - I_{1,i,j+1} + I_{2,i+1,j} - I_{2,i,j} + I_{2,i+1,j+1} - I_{2,i,j+1}) \\ I_t(i, j) &= \frac{1}{4} (I_{2,i,j} - I_{1,i,j} + I_{2,i+1,j} - I_{1,i+1,j} + I_{2,i+1,j+1} - I_{1,i+1,j+1} + I_{2,i,j+1} - I_{1,i,j+1}) \end{aligned}$$

Où,  $I_{k,i,j}$  dénote le  $(i, j)$ ème pixel de la  $k$ ième image. Et nous imposons des conditions de Neumann homogène aux bords, ce qui se traduit que si l'indice  $(i, j)$  dépasse  $\Omega$  nous fixons la valeur à celle du pixel  $(i, j \pm 1)$  (si dépassement de l'image en lignes) et  $(i \pm 1, j)$  (si dépassement de l'image en colonnes).

### 3.2.2 Approximation du laplacien

Le schéma de discrétisation du laplacien utilisé est très classique[?] :

$$\Delta f \approx \bar{f} - f$$

Où  $\bar{f}(x, y)$  est donné par :

$$\begin{aligned} \frac{1}{6}(f(x-h, y) + f(x+h, y) + f(x, y+h) + f(x, y-h)) + \\ \frac{1}{12}(f(x-h, y-h) + f(x-h, y+h) + f(x+h, y-h) + f(x+h, y+h)) \end{aligned}$$

C'est à dire, pour nous, en terme d'indices matriciels :

$$\begin{aligned} \frac{1}{6}(f_{i-1,j} + f_{i+1,j} + f_{i,j+1} + f_{i,j-1}) + \\ \frac{1}{12}(f_{i-1,j-1} + f_{i-1,j+1} + f_{i+1,j-1} + f_{i+1,j+1}) \end{aligned}$$

Avec la même condition de Neumann homogène aux bords.

### 3.2.3 Discrétisation du système

Nous pouvons réécrire le système (??) sous la forme approximée suivante :

$$\begin{cases} (\alpha^2 + I_x^2 + I_y^2)(u - \bar{u}) = -I_x(I_x\bar{u} + I_y\bar{v} + I_t) \\ (\alpha^2 + I_x^2 + I_y^2)(v - \bar{v}) = -I_y(I_x\bar{u} + I_y\bar{v} + I_t) \end{cases} \quad (5)$$

Puis nous discrétisons, en posant :  $u^{n+1} = f(\bar{u}^n)$  :

$$\begin{cases} u^{n+1} = \bar{u}^n - I_x \frac{I_x\bar{u}^n + I_y\bar{v}^n + I_t}{\alpha^2 + I_x^2 + I_y^2} \\ v^{n+1} = \bar{v}^n - I_y \frac{I_x\bar{u}^n + I_y\bar{v}^n + I_t}{\alpha^2 + I_x^2 + I_y^2} \end{cases} \quad (6)$$

**Preuve :**

La démonstration est donnée à l'annexe ??.

□

**3.2.4 Contrôle de l'erreur**

Nous considérons ici l'espace des matrices muni de la norme induite par le produit scalaire suivant[?] :

$$\forall A, B \in M_n(\mathbb{R}) \quad \langle A | B \rangle = \text{Tr}({}^t B A)$$

D'où la norme :

$$\forall A \in M_n(\mathbb{R}) \quad \|A\|^2 = \langle A | A \rangle$$

Nous regarderons alors l'erreur du schéma définie par :

$$e = \|u^{n+1} - u^n\|^2 + \|v^{n+1} - v^n\|^2$$

Ce qui nous fournit un critère de convergence lorsque que  $e < \epsilon$ .

**3.3 Implémentation****3.3.1 Approximation des dérivées partielles et du Laplacien**

Ce qui est intéressant dans les schémas présentés plus haut est que ceux-ci s'interprètent très bien en terme de convolution de matrice, qui est une opération très simple à implémenter en Matlab.

**Exemple :** Pour la discréétisation de  $I_x$ , nous utilisons le noyau de convolution :

$$N = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix}$$

Ce qui nous donne ( $I_k$  est la matrice de l'image  $k$ ) :

$$\frac{\partial I}{\partial x} = I_x \approx \frac{1}{4}(I_1 + I_2) * N$$

Nous pourrions émettre un doute sur le fait de convoler avec un noyau non symétrique et des résultats donnés par Matlab. C'est pourquoi nous avons implémenté le schéma avec des boucles, les résultats étaient rigoureusement identiques (Ouf!!).

```
function [Ix, Iy, It] = deriv(I1, I2)
    Ix = conv2(I1+I2, 0.25* [-1 1; -1 1], 'same');
    Iy = conv2(I1+I2, 0.25* [-1 -1; 1 1], 'same');
    It = conv2(I1-I2, 0.25*ones(2), 'same');
end
```

Code 2 – Fonction d'approximation des dérivées partielles de l'intensité

De même pour l'approximation du laplacien nous utilisons un noyau de convolution pour calculer  $\bar{f}$  :

$$N = \frac{1}{12} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

La fonction prend en entrée une matrice  $U$  et renvoie la matrice  $V = \bar{U} = U * N$ .

```

function V = reduc(U)
    % Noyau de convolution
    s = (1/12)*[1 2 1;2 0 2;1 2 1];
    V = conv2(U,s,'same');
end

```

Code 3 – Fonction pour l'approximation du laplacien

### 3.3.2 Le schéma

Voici l'implémentation du schéma (??). La fonction prend en entrée deux images (de tailles identiques), le coefficient de pénalisation  $\alpha$ , une borne de précision  $\epsilon$  et un nombre d'itérations maximum.

La fonction donne en sortie deux matrices  $u$  et  $v$  de même taille que les images initiales, qui donnent en chaque pixel l'approximation du déplacement horizontal et vertical.

```

function [u v] = HS(I1,I2,alpha,eps,niter)

% Calcul des dérivées spatiales et temporel de l'intensité
[Ix,Iy,It] = deriv(I1,I2);

% Initialisation
u = zeros(size(I1));
v = u;
erreur = 1;
k = 0;
C = alpha^2+Ix.^2+Iy.^2;

while erreur > eps && k < niter
    % Calcul deubar et vbar pour l'approximation du laplacien
    uAvg = reduc(u);
    vAvg = reduc(v);

    % Sauvegarde du second membre qui apparait dans les deux
    % équations du système
    tmp = ((Ix.*uAvg)+(Iy.*vAvg)+It)./C;

    % Calcul de l'itération suivante
    u0 = uAvg - Ix.*tmp;
    v0 = vAvg - Iy.*tmp;

    k = k + 1;
    % Calcul de la norme de l'erreur
    erreur = trace((u-u0)'*(u-u0)) + trace((v-v0)'*(v-v0));
    u = u0;
    v = v0;
end
end

```

Code 4 – Fonction implémentant la méthode de Horn et schunck

### 3.4 Mise en oeuvre de la méthode

#### 3.4.1 Initialisation

Voici les deux images utilisées en entrées (Figures ??, ??). Les images font 420 pixels de large par 360 de hauteur.



FIGURE 6 – Image initiale



FIGURE 7 – Image décalée

Comme la différence entre ces deux images n'est pas visible au premier regard, nous donnons donc l'image qui est la différence des deux précédentes. Les zones les plus sombres sont celles où le décalage des couleurs est le plus important :



FIGURE 8 – Différence des deux images

Nous fixons les paramètres à  $\epsilon = 0.1$  et  $\alpha = 15$ .

#### 3.4.2 Résultats

La méthode nous donne en sortie deux matrices  $u$  et  $v$  de même taille que les images initiales. La matrice  $u$  donne pour chaque pixel un déplacement horizontal, et la matrice  $v$  donne pour chaque pixel un déplacement vertical. La méthode, avec les paramètres précédemment annoncés, converge en 1.02 secondes pour 251 itérations.

Le champ de vecteur obtenu est alors (Figure ??) :



FIGURE 9 – Résultats obtenus par la méthode de Horn et Schunck

Nous observons que la méthode fonctionne correctement. Le champ de déplacement obtenu est cohérent avec les déplacements observés sur la paire initiale.

### 3.4.3 Variations sur le thème des résultats

Varions le coefficient  $\alpha$ . Premièrement regardons ce qu'il se passe lorsque  $\alpha$  est grand. Disons  $\alpha = 100$ . La méthode converge en 366 itérations et nous obtenons le champ de déplacement suivant (Figure ??) :



FIGURE 10 – Résultats obtenus par la méthode de Horn et Schunck avec  $\alpha = 100$

Nous remarquons que le champ est beaucoup plus régulier (c'est à dire que les variations de directions dans les zones de déplacement apparaissent plus lisses), ce qui est logique puisque nous avons imposé un coefficient de pénalisation de la régularité du flot plus important.

Deuxièmement, regardons ce qu'il se passe lorsque  $\alpha = 0$ . La méthode converge en une seule itération, et le champ de déplacement obtenu est le suivant (Figure ??) :



FIGURE 11 – Résultats obtenus par la méthode de Horn et Schunck avec  $\alpha = 0$

Ce résultat nous donne le champ de déplacement dans le sens du gradient d'intensité, comme annoncé en introduction, cette grandeur peut être calculée directement. Le champ de déplacement apparaît comme désorganisé, cela soulève le fait que le déplacement n'a aucune raison d'être dans le sens du gradient et nous pousse à introduire une seconde méthode ;

## 4 Amélioration de la méthode de Horn et Schunck

Jusqu'à présent, nous avions supposé que le champ de déplacement était lisse dans la direction du gradient d'intensité. Mais cela n'a pas de raison, à priori, d'être le cas, en effet pourquoi le déplacement d'un objet se ferraît dans la direction du gradient ?

Comme nous avons fait l'hypothèse fondamentale que l'intensité lumineuse d'un objet était constante au cours du temps, nous allons nous placer le long des lignes de niveau de l'intensité. Plutôt que d'approcher le champ de déplacement en se plaçant colinéairement au gradient d'intensité, nous alors suivre l'orthogonal de ce gradient.

Ainsi lors du déplacement de l'objet (que nous supposons toujours petit par rapport à la taille de l'image) nous suivrons le déplacement de la même ligne de niveau.

### 4.1 Le problème modifié

Nous projetons le gradient du déplacement, que nous notons abusivement par :  $\nabla h = \begin{pmatrix} \nabla u \\ \nabla v \end{pmatrix}$ , perpendiculairement au gradient d'intensité. C'est à dire sur le sous espace  $(\nabla I)^\perp$  :

$$(\nabla I)^\perp = Vect \left\{ I_0 := \begin{pmatrix} -I_y \\ I_x \end{pmatrix} \right\}$$

Nous appliquons les méthodes classiques de projeté orthogonal pour le produit scalaire considéré :

$$\begin{cases} (\nabla h)_{orth} = \frac{\langle \nabla h | I_0 \rangle}{\| I_0 \|^2} I_0 \\ = \frac{1}{I_x^2 + I_y^2} (-\nabla u I_y + \nabla v I_x) I_0 \end{cases} \quad (7)$$

L'idée est alors de remplacer le terme régularisant  $\int |\nabla u|^2 + |\nabla v|^2$  par la norme  $L^2$  de cette projection sur  $(\nabla I)^\perp$  :

$$\begin{aligned} \|(\nabla h)_{orth}\|_{L^2}^2 &= \int \left| \frac{1}{\|I_0\|^2} \langle \nabla h | I_0 \rangle I_0 \right|^2 d\mu \\ &= \int \left( \frac{1}{\|I_0\|} \right)^4 \langle \nabla h | I_0 \rangle^2 \|I_0\|^2 d\mu \\ &= \int \frac{1}{\|I_0\|^2} \langle \nabla h | I_0 \rangle^2 d\mu \\ &= \int \frac{1}{I_x^2 + I_y^2} Tr(\nabla h^T I_0)^2 d\mu \\ &= \int \frac{1}{I_x^2 + I_y^2} Tr(\nabla h^T I_0) Tr(\nabla h^T I_0) d\mu \\ &= \int \frac{1}{I_x^2 + I_y^2} Tr(\nabla h^T I_0) Tr(I_0^T \nabla h) d\mu \\ &= \int \frac{1}{I_x^2 + I_y^2} Tr(\nabla h^T I_0 I_0^T \nabla h) d\mu \\ &= \int Tr(\nabla h^T \tilde{W} \nabla h) d\mu \end{aligned}$$

Ou la dernière égalité vient du fait que si  $a, b \in \mathbb{R}$  alors  $\text{Tr}(a)\text{Tr}(b) = \text{Tr}(ab)$ . Et  $\tilde{W} = \frac{1}{\|I_0\|^2} \begin{pmatrix} I_y^2 & -I_x I_y \\ -I_x I_y & I_x^2 \end{pmatrix}$ .

En fait nous utiliserons une matrice pondérée pour éviter une division par zéro lorsque le gradient d'intensité est nul.

$$W = \frac{1}{\|I_0\|^2 + 2\gamma} \begin{pmatrix} I_y^2 + \gamma & -I_x I_y \\ -I_x I_y & I_x^2 + \gamma \end{pmatrix}$$

Nous introduisons alors la nouvelle fonctionnelle à minimiser :

$$J(u, v) = \int_{\Omega} (I_x u + I_y v + I_t)^2 + \alpha^2 \text{Tr} [\nabla h^T W \nabla h] d\mu \quad u, v \in H^1(\Omega)$$

(8)

Afin d'appliquer les équations d'Euler-Lagrange dans cette fonctionnelle, il nous faut expliciter la valeur de la trace. Un calcul direct, fastidieux mais facile, donne<sup>2</sup> :

$$\text{Tr} [\nabla h^T W \nabla h] = \frac{1}{I_x^2 + I_y^2 + 2\gamma} [(I_y^2 + \gamma)(\nabla u)^2 + (I_x^2 + \gamma)^2(\nabla v)^2 - 2I_x I_y (\nabla u \nabla v)]$$

Nous remarquons que lorsque  $\nabla I = 0$  nous retrouvons le facteur pénalisant de la méthode de Horn et Schunck.

Les équations d'Euler-Lagrange donne alors le système suivant à résoudre (avec  $A = \frac{1}{I_x^2 + I_y^2 + 2\gamma}$ ) :

$$\begin{cases} I_x(I_x u + I_y v + I_t) = A\alpha^2 [(I_y^2 + \gamma)\Delta u - I_x I_y \Delta v] \\ I_y(I_x u + I_y v + I_t) = A\alpha^2 [(I_x^2 + \gamma)\Delta v - I_x I_y \Delta u] \end{cases} \quad (9)$$

Ce système se réécrit sous la forme suivante (le calcul est donné à l'annexe ??) :

$$\begin{cases} (A\alpha^2 \gamma + I_x^2 + I_y^2)(u - \bar{u}) = -I_x(I_x \bar{u} + I_y \bar{v} + I_t) \\ (A\alpha^2 \gamma + I_x^2 + I_y^2)(v - \bar{v}) = -I_y(I_x \bar{u} + I_y \bar{v} + I_t) \end{cases} \quad (10)$$

Nous discrétisons alors comme précédemment pour obtenir :

$$\begin{cases} u^{n+1} = \bar{u} - I_x \frac{I_x \bar{u} + I_y \bar{v} + I_t}{A\alpha^2 \gamma + I_x^2 + I_y^2} \\ v^{n+1} = \bar{v} - I_y \frac{I_x \bar{u} + I_y \bar{v} + I_t}{A\alpha^2 \gamma + I_x^2 + I_y^2} \end{cases} \quad (11)$$

---

2. Afin de guider le lecteur curieux de mener ce calcul, nous précisons que  $\nabla h = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}$ .

## 4.2 Implémentation

Il est remarquable que malgré un changement radical de point de vue du problème (nous sommes passé d'une vue orthogonale à une vue tangentielle aux lignes de niveaux), les systèmes discréétisés des deux méthodes ne diffèrent que par une simple constante. L'implémentation de cette méthode améliorée est donc sensiblement identique à la précédente.

```
function [u, v] = HS_Amm(I1, I2, alpha, eps, niter)

% Pour éviter la division par 0
gamma = 1000;

[Ix, Iy, It] = deriv(I1, I2);

Ix2 = Ix.^2;
Iy2 = Iy.^2;

C = ((Ix2+Iy2+2*gamma).^-1)*alpha^2*gamma + Ix2 + Iy2;

u = zeros(size(I1));
v = u;

norme = 1;
k=0;

while norme > eps && k < niter
uAvg = reduc(u);
vAvg = reduc(v);

tmp = ((Ix.*uAvg)+(Iy.*vAvg)+It)./C;
u0 = uAvg - Ix.*tmp;
v0 = vAvg - Iy.*tmp;

k=k+1;
norme = trace((u-u0)'*(u-u0)) + trace((v-v0)'*(v-v0));
u = u0;
v = v0;
end
end
```

Code 5 – Implémentation de la méthode améliorée de Horn et Schunck

## 4.3 Mise en oeuvre de la seconde méthode

Afin de pouvoir comparer ces résultats aux précédents, nous utilisons la même paire d'images. Nous fixons  $\epsilon = 0.1$ ,  $\alpha = 15$  et  $\gamma = 1000$ . Nous avons alors une convergence en 1.63 secondes pour un total de 244 itérations.

Nous observons que les résultats sont semblables à ceux de la première méthode. Notamment dans les zones où l'intensité est uniforme en espace ( $\nabla I = 0$ ), nous observons bien des résultats identiques. Afin de mieux visualiser les différences, nous mettons en parallèle les deux résultats :



FIGURE 12 – Résultats avec la méthode 2



FIGURE 13 – Résultats obtenus par la méthode de Horn et Schunck



FIGURE 14 – Résultats avec la méthode 2

Cela montre que dans les zones d'intensité lumineuse constante, le champ de déplacement est identique. Nous pouvons aussi comparer les deux méthodes en normes :

$$\|u_1 - u_2\|_\infty + \|v_1 - v_2\|_\infty = 10.17$$

Nous voyons que sur cette paire d'image, la différence entre les deux méthodes est minime.

**Mini - conclusion :** Nous sommes en réalité surpris que les deux images donnent à ce point des résultats similaires. Cela se voit bien dans les schéma, c'est juste à une constate près que les schéma diffère, en fait en choisissant bien  $\alpha$  et  $\gamma$  nous pourrions obtenir exactement les même résultats. Cela nous laisse perplexe. C'est pourquoi nous utiliserons plutôt la première méthode dans le reste du rapport.



FIGURE 15 – Différence des deux méthodes

## 5 Application des méthodes sur des images bruitées

Jusque ici nous avons utilisé des images sans défauts, mais dans la vie réelle, les images issues de capteurs sont souvent abîmées, bruitées. Nous allons donc rapidement mettre en oeuvre une fonction de bruitage sur nos images et comparer les 2 méthodes, présentées ci-dessus, sur la paire d'images bruitées puis sur la même paire bruitée et filtrée par l'équation de la chaleur.

### 5.1 Bruitage & Lissage d'image

Commençons par présenter la fonction de bruitage. Celle-ci est très simple, elle applique un bruit uniforme sur chaque pixel de l'image.

Voici un exemple avec un bruit uniforme tiré dans  $[-50; 50]$ .



FIGURE 16 – Image bruitée

Voici le code de la fonction de bruitage :

```
function J = bruitage(I,range)
J = I + round(2*range*rand(size(I))-range);
end
```

Code 6 – Implémentation de la fonction de bruitage

Pour filtrer ces images bruitées, nous avons deux possibilités : par un filtre gaussien classique ou par l'équation de la chaleur. Il s'avère en fait que ces deux approches sont identiques<sup>3</sup> (cela s'observe sur les figures ?? et ??).

Voici une image filtrée par un filtre gaussien :



FIGURE 17 – Image filtrée par un filtre gaussien :  $\sigma = 2.5$

Et la même image à laquelle nous avons appliqué l'équation de la chaleur :



FIGURE 18 – Image filtrée par l'équation de la chaleur :  $T = 2$ ,  $dt = 0.2$

---

3. En fait cela traduit que le laplacien est un opérateur de diffusion.

Voici le code pour appliquer un filtre gaussien à une image :

```
function J = gauss(I,N,sigma)
%% Cr ation du noyau de convolution %%
[x y]=meshgrid(round(-N/2):round(N/2), round(-N/2):round(N/2));
f=exp(-x.^2/(2*sigma^2)-y.^2/(2*sigma^2));
f=f./sum(f(:));

%%% Application du filtre %%
J = conv2(I,f,'same');
end
```

Code 7 – Impl ementation du filtrage gaussien

et pour appliquer l' quation de la chaleur :

```
function J = chaleur(I,T,dt)
J = I;
t = 0:dt:T;
% pour la disc tisation du lalaci n
s = [0 1 0;1 0 1;0 1 0];

for k=1:length(t)-1
J = J + dt*(conv2(J,s,'same'))-4*J;
end
end
```

Code 8 – Impl ementation du filtrage par  quation de la chaleur

## 5.2 Les m thodes sur des images bruit es

Essayons d'appliquer les deux m thodes de Horn et Schunck pr sent es ci-dessus sur une paire d'images bruit es :



FIGURE 19 – M thode de Horn et Schunck 1 sur images bruit es



FIGURE 20 – M thode de Horn Shunck 2 sur images bruit es

Nous observons que les r sultats des deux m thodes ne sont gu re exploitables. A la rigueur, un oeil avis  pourrait deviner le champ de vecteur dans la figure ?? mais ce n'est vraiment

pas net. Cependant, remarquons qu'il n'y a pas tellement de sens à comparer les méthodes sur les images complètement bruitées, le champ de luminosité ne vérifiant pas l'hypothèse fondamentale de non-variation dans le temps.

Regardons ce qu'il se passe lorsque les deux images sont filtrées<sup>4</sup> :

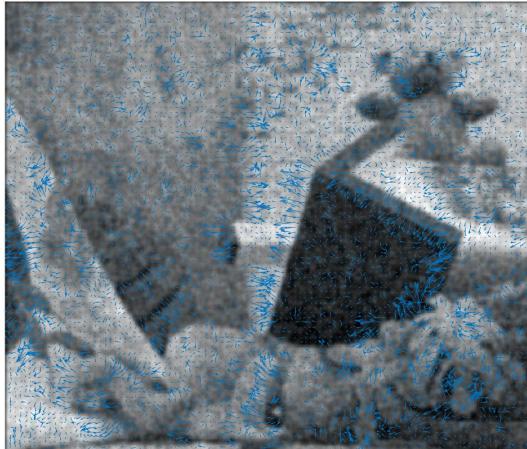


FIGURE 21 – Méthode de Horn et Schunck 1 sur images bruitées puis filtrées



FIGURE 22 – Méthode de Horn et Schunck 2 sur images bruitées puis filtrées

Nous remarquons que les résultats comportent l'information principal du mouvement, mais que les détails sont complètement annihilés par le filtrage.

Pour demander plus de régularité, nous relançons la seconde méthode avec  $\alpha = 30$ . Les résultats sont vraiment intéressant, la composante principale du mouvement est plus visible, néanmoins les petits détails disparaissent :



FIGURE 23 – Méthode de Horn et Schunck 2 sur images bruitées puis filtrées  $\alpha = 30$

---

4. Nous filtrons par l'équation de la chaleur par pur plaisir.

Comparons les deux méthodes sur les images filtrées :

$$\|u_{1bc} - u_{2bc}\|_\infty + \|v_{1bc} - v_{2bc}\|_\infty = 11.80$$

Nous retrouvons un ordre de grandeur de l'erreur de l'ordre du cas non bruité. Néanmoins pour l'erreur  $\epsilon$  imposée, c'est la seconde méthode qui converge le plus vite : en 102 itérations contre 125 itérations pour la première.

## 6 Prise en charges des grands mouvements

Nous avions supposé que les déplacements dans l'image était petit par rapport à la taille de l'image. Nous allons voir comment prendre en compte des déplacements importants par rapport à la taille de l'image.

L'idée est de réduire la taille de l'image, de calculer un champ de déplacement puis de revenir à l'image originale et d'affiner le champ précédemment calculé. En effet, en réduisant l'image, les déplacements qui sont grand par rapport à la taille de l'image seront réduit et nous pourrons donc appliquer l'un des deux algorithmes ci-dessus. Pour implémenter les fonctionnalités attendues, introduisons pour cela un outil fort pratique : l'interpolation. A cet effet, considérons que notre image est définie sur un domaine  $\Omega \subset \mathbb{R}^2$  par une fonction ne prenant ses valeurs qu'aux points entiers (ie.  $\in \mathbb{Z}^2 \cap \Omega$ ) (Figures ??, ??).

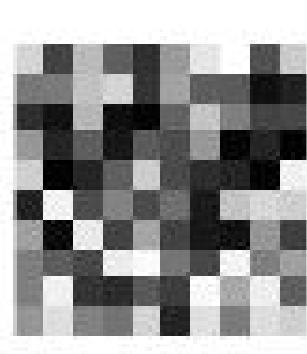


FIGURE 24 –  
Représentation classique  
d'une image

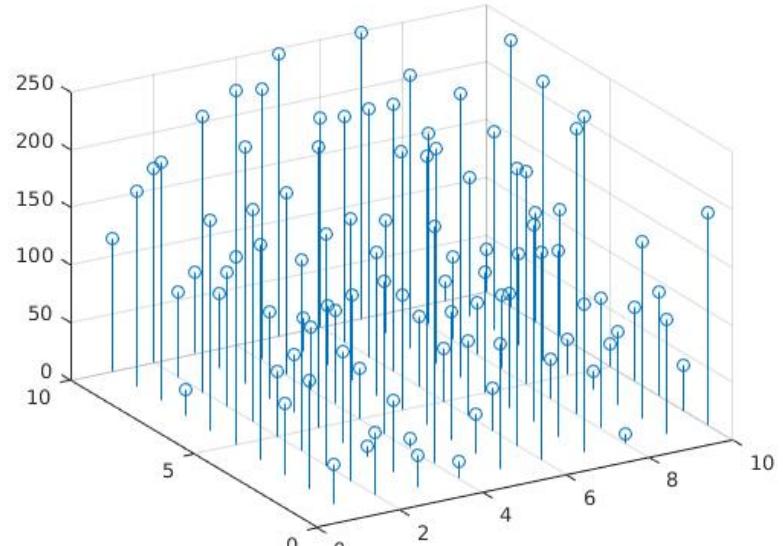


FIGURE 25 – Représentation de l'image sur son domaine continu

Mais en réduisant l'image, nous allons devoir attribuer des valeurs à la fonction *image* en des points non entiers. C'est là qu'intervient l'interpolation.

Afin de simplifier, nous implémentons des fonctions d'interpolation en une dimension. Nous appliquerons en premier lieu ces techniques sur les lignes de notre image puis sur les colonnes pour obtenir l'interpolation en deux dimensions.

Essayons de comprendre ce qu'il se passe en une dimension. Nous partons d'une image (en 1D) c'est à dire d'une fonction définie sur un ensemble finis :  $\llbracket 1 \dots 10 \rrbracket$  par exemple.

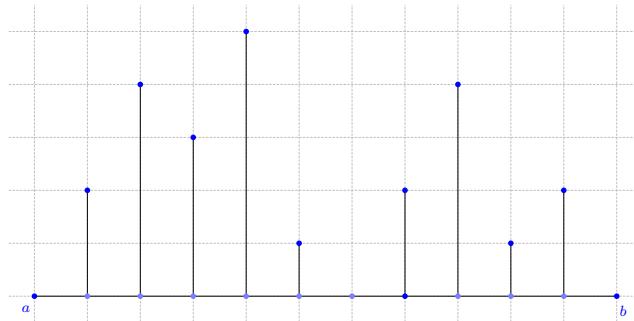


FIGURE 26 – Représentation 1D d'une image  $\Omega = [a, b]$

Appliquons un champ de déplacement à notre image. Cela revient à décaler les indices des points ou est évalué la fonction "image".

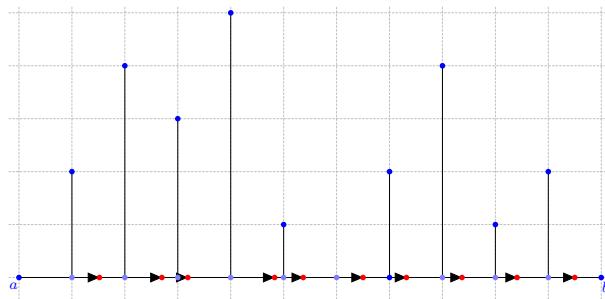


FIGURE 27 – Représentation 1D d'une image et de son champ de déplacement

Mais alors, comment attribuer une valeur à ces nouveaux points non entiers ? Remarquons, que le nombre de points ou la fonction image est évaluée n'a pas changé, le domaine de définition est le même.

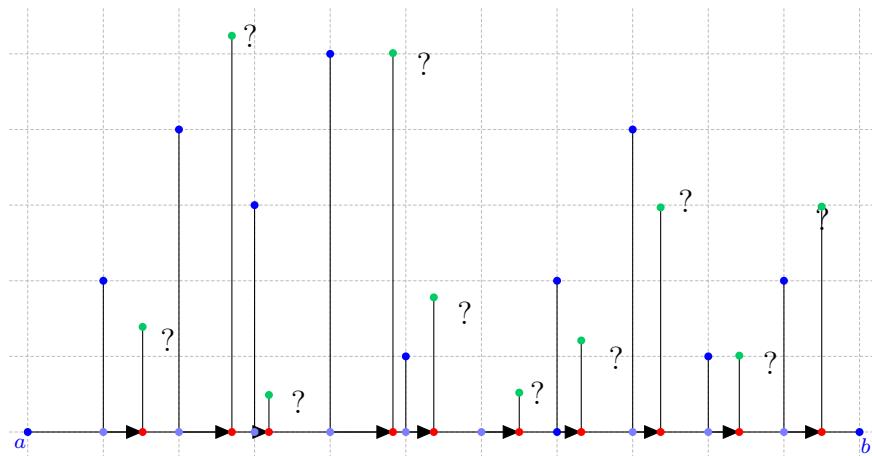


FIGURE 28 – Problème de l'interpolation

## 6.1 Interpolation linéaire

Pour donner une valeur à cette fonction "image" aux points non entier, une idée naturel et de relier deux valeurs successives connues de la fonction par des morceaux de droites. Ainsi la valeur de la fonction "image" aux points non entiers est donnée comme le barycentre de deux valeurs connues pondérées par un certain coefficient.

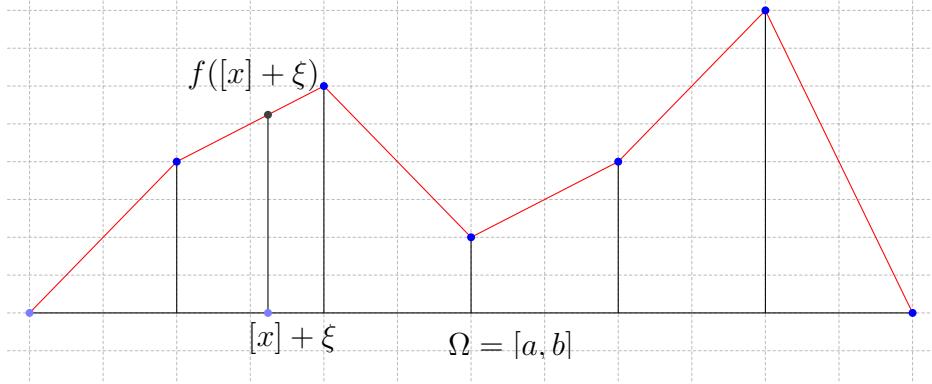


FIGURE 29 – Interpolation linéaire

La formule d'interpolation en un point  $x$  non entier est alors donnée par :

$$f(x) = f([x])(1 - \xi) + f([x] + 1)\xi$$

Où,  $[•]$  dénote la partie entière inférieure et  $\xi = x - [x]$ , ce qui implique  $\xi \in [0, 1[$ .

Voici le code qui donne l'interpolation linéaire en 1D. Cette fonction prend en entrée un vecteur  $H$ , qui donne les valeurs de la fonction "image" aux points entiers (naturellement numérotée de 1 à  $n$  en Matlab), et un vecteur  $u$  de même taille que  $H$  qui donne le déplacement des points entiers du support de  $H$ . La fonction retourne les valeurs de la fonction aux points  $\forall i \in \llbracket 1 \dots n \rrbracket$ ,  $i + u(i)$ .

**Remarque :** Une petite subtilité est que l'interpolation va systématiquement chercher une valeur à gauche et à droite, afin d'éviter tout problème, nous rajoutons deux points fictifs à  $H$  de valeur égale à la première et dernière valeur de  $H$ . Et nous imposons un déplacement nul sur ces points. Cela correspond à une condition de Neumann homogène au bord.

```

function Hu = linearInterp(H,u)
H = [H(1),H,H(end)];
m = size(H,2);
u = [0,u,0];
j = [1:m] + u;
p = floor(j);
Xi = j - p;

% Gestion des déplacements de capacités
% ...
% Fin Gestion des déplacements de capacités

Hu = H(p).*(1-Xi)+H(p+1).*Xi;
Hu = Hu(2:end-1);
end

```

Code 9 – Implémentation de la fonction d'interpolation linéaire

Voici le résultat que l'on obtient sur les données  $H = [0, 2, 2, 2, 1]$  et  $u$  demande l'évaluation en tout points de  $[0, 8 = 6]$ .

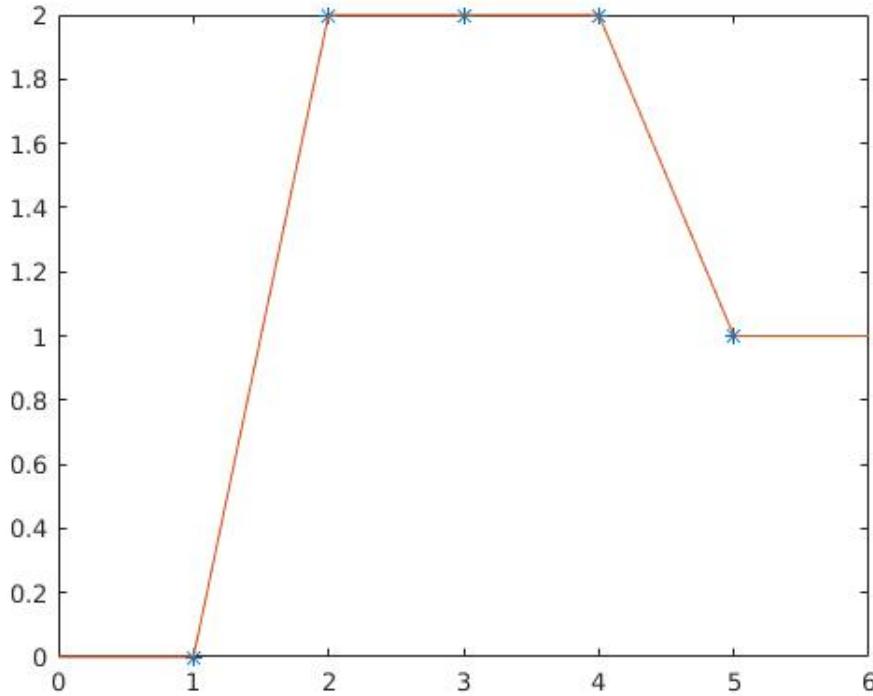


FIGURE 30 – Visualisation de l'interpolation linéaire

Afin de tester notre méthode sur des images, nous procédons comme suit :

1. Nous importons une paire d'images auquelles nous appliquons une des deux méthodes de Horn et Schunck
2. Nous "oublions" la deuxième image et nous effectuons l'interpolation linéaire de la première (d'abord selon les lignes avec le champ donné par la matrice  $u$ , puis sur les colonnes sur le champ donné par la matrice  $v$ ).
3. Nous comparons l'image interpolé avec la deuxième image de la paire, en théorie, les deux images devraient être semblables.

Nous voyons que le résultat (Figure ??) est assez concluant, la norme de l'erreur est de  $\|I_{interp} - I_2\|_\infty = 195$ .



FIGURE 31 – Interpolation linéaire

Voici le code utilisé pour générer cette exemple.

```

z = double(imread('data/Teddy/frame10.png'));
I2 = double(imread('data/Teddy/frame11.png'));

alpha = 15;
eps = 0.1;
niter = 2000;

tic;
[u v] = HS(z,I2,alpha,eps,niter);
toc;

n = size(z);
exagere = 1;
u = exagere*u;
v = exagere*v;

for i=1:n(1)
    z1(i,:) = linearInterp(z(i,:),u(i,:));
end
for i=1:n(2)
    z1(:,i) = linearInterp(z1(:,i)',v(:,i)')';
end

figure;
subplot(1,2,1)
imshow(I2,[0,255]);
title('Image 2 de la paire originale');
subplot(1,2,2)
imshow(z1,[0,255]);
title('Image 2 interpolée');

max(max(abs(I2-z1)))

```

Code 10 – Code de la fonction de test de l'interpolation linéaire

## 6.2 Interpolation par des Splines<sup>5</sup>

Afin d'améliorer les résultats de la section précédente, nous allons interpoler les données par une fonction lisse qui minimise la courbure. Nous posons alors le problème de minimisation :

$$\begin{cases} \mathcal{T} = \operatorname{argmin} \int_{\Omega} (\mathcal{T}''(t))^2 dt \\ \mathcal{T}(x_j) = dataT(j) \quad j = 1, \dots, m \end{cases} \quad (12)$$

Nous savons que c'est une fonction spline cubique qui réalise ce minimum (la preuve est donnée à l'annexe ??).

Donnons une base de  $\mathcal{S}^{3,\tau}$ , l'espace vectoriel des fonctions Splines.

Plusieurs observations sont à mettre en perspective :

1. Les points où les données sont connues sont les entiers de 1 à  $m$ . A partir de la définition des Splines, il est aisément de déduire que les fonctions de base s'obtiennent par translation d'une fonction de base fondamentale.
2. Nos Splines sont de degré 3 et donc leur support est de la forme  $[t_i, t_{i+4}] = [i, i + 4]$ .
3. La Spline fondamentale, que nous nommerons *mère Spline*, est celle pour laquelle son support est pour la première fois en contact avec les données, c'est à dire le minimum des indices  $i$  tels que  $i + 4 > 1$ .

Ainsi la *mère Spline* est définie sur  $[-2, 2]$ . Le calcul est détaillé à l'annexe ??, et nous obtenons finalement :

$$6B_{-2,3}(x) = (x + 2)^3 \mathbb{1}_{[-2,-1[} + (-3x^2(x + 2) + 4) \mathbb{1}_{[-1,0[} + (3(x - 2)x^2 + 4) \mathbb{1}_{[0,1[} + (2 - x^3) \mathbb{1}_{[1,2[} \quad (13)$$

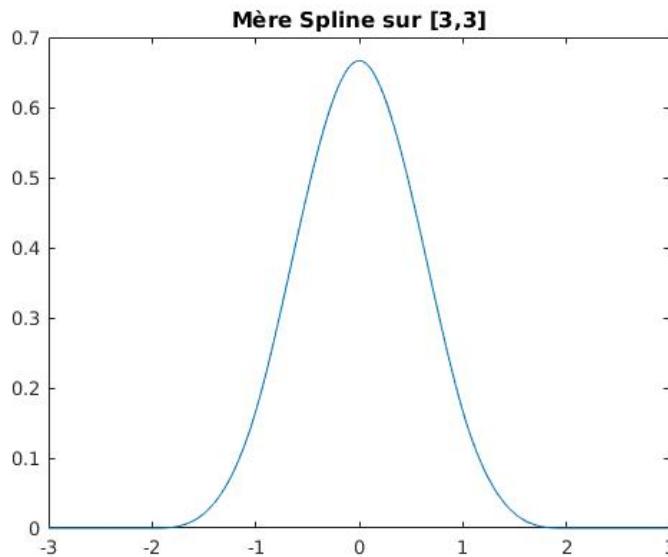


FIGURE 32 – Représentation graphique de la *mère Spline*

5. Nous supposons le lecteur familier avec les Splines et nous nous permettons de passer rapidement sur des propriétés élémentaires mais qui demandent un calcul fastidieux.

La fonction interpolée s'écrit comme une combinaison linéaire des fonctions de bases :

$$f(x) = \sum_{k=1}^m c_k B_k(x)$$

Où,  $B_k$  est la fonction de base qui s'obtient de la *mère Spline* par :  $B_k(x) = B_0(x - k)$ . La condition d'interpolation nous permet de calculer les coefficients  $c_k$ . Si  $data \in \mathbb{R}^m$  est le vecteur des données, les coefficients  $c_k$  s'obtiennent par le système :

$$B_m c = data \quad B_m = (B_k(j))_{j,k \in [1 \dots m]} = \begin{pmatrix} 4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 4 \end{pmatrix}$$

Avec les mêmes notation que précédemment, nous posons  $x = [x] + \xi$ . Avec des considérations de support, nous arrivons à la formule d'interpolation suivante :

$$f(x) = c_{[x]-2} B_0(\xi + 2) + c_{[x]-1} B_0(\xi + 1) + c_{[x]} B_0(\xi) c_{[x]+1} B_0(\xi - 1) c_{[x]+2} B_0(\xi - 2)$$

```
function Hu = splineInterp(H,u)

m = size(H,2);
B = full(gallery('tridiag',m,1,4,1));
c = (B\H)';

u = [1:m] + u;
p = round(u);
xi = u-p;

q = p-2;
r = p-1;
s = p+1;
t = p+2;

% Gestion des déplacements de capacités
% ...
% Fin gestion des déplacements de capacités

Hu = c(q).*mereSpline(xi+2)+c(r).*mereSpline(xi+1)+c(p).*mereSpline
    (xi)+c(s).*mereSpline(xi-1)+c(t).*mereSpline(xi-2);
end
```

Code 11 – Implémentation de la fonction d'interpolation Spline

Sur les mêmes données que précédemment  $H = [0, 2, 2, 2, 1]$ , voici l'interpolation Spline :

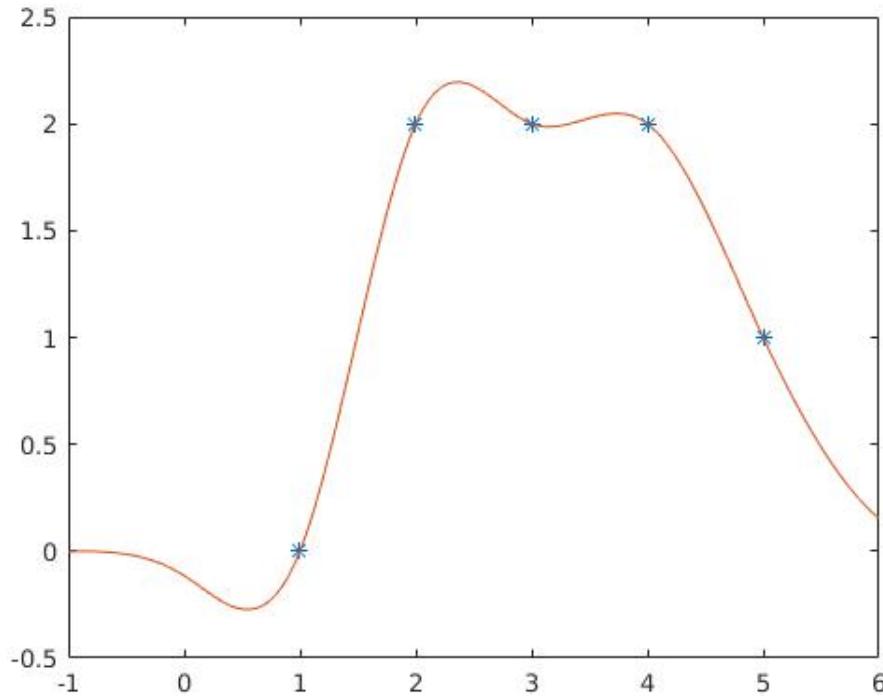


FIGURE 33 – Visualisation de l'interpolation Spline

Cela met en évidence un des inconvénients de l'interpolation Spline, des oscillations apparaissent lorsque les données sont stables.

Nous procémons de la même manière que précédemment pour tester notre méthode.

La norme de l'erreur est de  $\|I_{interp} - I_2\|_\infty = 195$ .



FIGURE 34 – Interpolation spline

Voici le code pour générer cette exemple.

```

z = double(imread('data/Teddy/frame10.png'));
I2 = double(imread('data/Teddy/frame11.png'));

alpha = 15;
eps = 0.1;
niter = 2000;

tic;
[u v] = HS(z,I2,alpha,eps,niter);
toc;

n = size(z);
exagere = 1;
tmpu = u;
tmpv = v;
u = exagere*u;
v = exagere*v;
z1 = z;
for i=1:n(1)
    z1(i,:) = splineInterp(z(i,:),u(i,:));
end
for i=1:n(2)
    z1(:,i) = splineInterp(z1(:,i)',v(:,i)')';
end

figure;
subplot(1,2,1)
imshow(I2,[0,255]);
title('Image 2 de la paire originale');
subplot(1,2,2)
imshow(z1,[0,255]);
title('Image 2 interpolée');

max(max(abs(I2-z1)))

```

Code 12 – Code la fonction de test de l'interpolation Spline

En guise de bonus, voici l'image que nous obtenons lorsque le champ de déplacement est exagéré d'un facteur 30 :



FIGURE 35 – Interpolation Spline exagérée

Cette petite expérience nous permet de mettre en évidence le phénomène suivant : les zones où le gradient d'intensité est constant (toit de la maison, fond à gauche) ne présentent quasiment pas de déformations. C'est bien cohérent avec les hypothèses de la méthode qui se base sur les variations d'intensité lumineuse pour déterminer un champ de déplacement. Les zones présentant une absence de variation d'intensité ne présentent donc pas de déformations.

### 6.3 Redimensionnement d'image

Nous allons appliquer les résultats précédents au redimensionnement d'image. Gardons à l'esprit ce que nous souhaitons obtenir : une image réduite ou les grands déplacements sont réduits à taille admissibles pour la méthode de Horn et Schunck.

Nous écrivons une fonction qui prend en entrée une image  $I$  de taille  $nm$  et un facteur de réduction  $\nu < 1$  ou d'agrandissement  $\nu > 1$  et qui retourne la même image de taille  $[\nu n][\nu m]$ .

Toute l'astuce consiste à trouver le bon champ de vecteur à apposer sur l'image puis de faire fonctionner les méthodes précédentes.

FIGURE 36 – Champ de vecteur pour la réduction d'une image d'un facteur  $\nu = 0.3$

Voici les résultats obtenus :



FIGURE 37 – Rétrécissement d'une image



FIGURE 38 – Agrandissement d'une image

Nous observons que dans le cas du rétrécissement l'image se retrouve pixélisé (en partie due à Matlab qui interpole les valeurs de l'image pour l'afficher à taille visualisable), cela confirme le fait de devoir lisser les légèrement les images lorsque nous faisons de la réduction.

Et le code implémentant la fonction de redimensionnement des images :

```
function I2 = redim(z,nu)

m = size(z);
row = m(1);
col = m(2);

if nu > 1
    new_size = round(nu*m);
    tmp = zeros(new_size);
    tmp(1:row,1:col) = z;
    z = tmp;
    m = size(z);
```

```

new_row = m(1);
new_col = m(2);
else
    new_row = round(nu*row);
    new_col = round(nu*col);
end

z1 = zeros(m);
u = zeros(m);
v = zeros(m);

for i=0:row-1
    for j=0:col-1
        u(i+1,j+1) = (j)*(1-nu)/nu;
        v(i+1,j+1) = (i)*(1-nu)/nu;
    end
end

for i=1:m(1)
    z1(i,:) = splineInterp(z(i,:),u(i,:));
end
for i=1:m(2)
    z1(:,i) = splineInterp(z1(:,i)',v(:,i)')';
end

I2 = z1(1:new_row,1:new_col);

end

```

Code 13 – Implémentation de la fonction de redimensionnement

## 6.4 Nouvelle méthode

Mettons en oeuvre la méthode pour prendre en charge les grandes variations. Voici l'algorithme que nous appliquons :

1. Nous prenons une paire d'image lissée par un filtre gaussien d'écart type  $\sigma = 0.8$ .
2. Nous réduisons les images d'un facteur  $\nu$
3. Nous appliquons une méthode de Horn et Schunck sur les images réduites. Nous obtenons un flot  $h_\nu$ .
4. Nous remettons le flot à la taille originale, nous obtenons un flot  $h$ .
5. Nous refaisons un coup d'une méthode de Horn et Schunck sur les images à taille originale  $I_1$  et  $I_2(\bullet - h)$ . Nous obtenons un flot  $k$
6. Nous retournons le flot affiné  $h + k$ .

Lors de la seconde étape, les grands déplacement se trouvent réduit à taille admissible et ceux qui était déjà admissible se retrouvent tellement petits que non pris en compte par la méthode. Puis lors de la cinquième étape, les grands mouvements se retrouvent très petits alors que les petit déplacement ne sont pas affectés. Ainsi le flot  $k$  caractérise les petits déplacements.

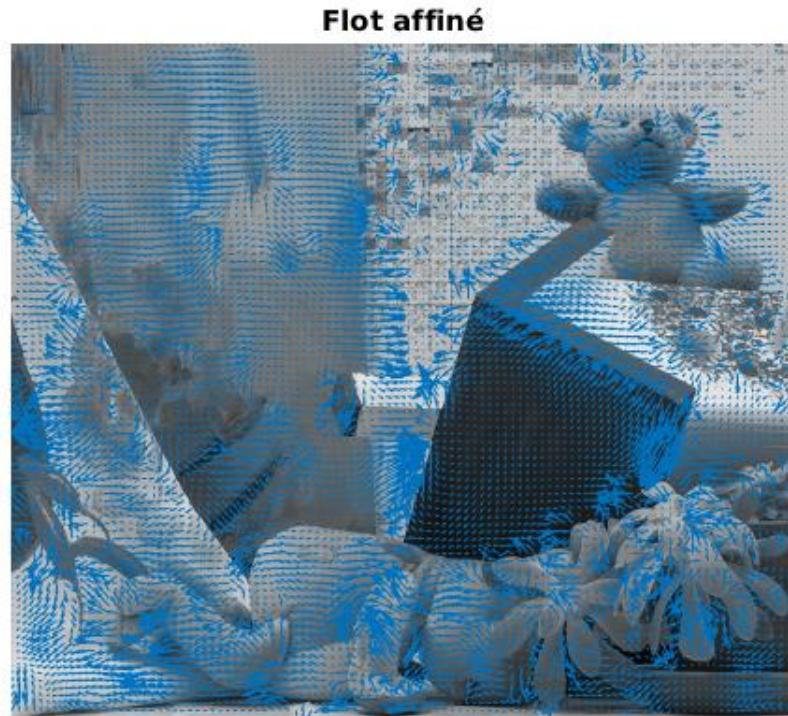


FIGURE 39 – Application de l'algorithme sur une paire d'images

Si nous comparons avec le flot initialement obtenu :

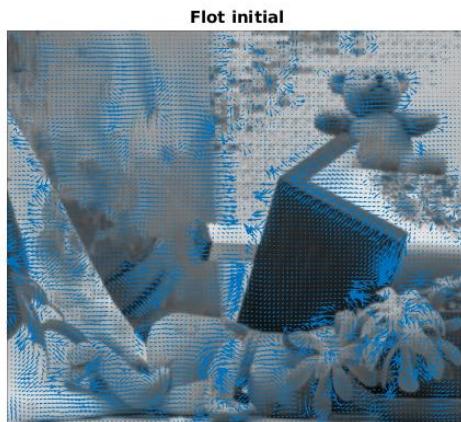


FIGURE 40 – Flot initial

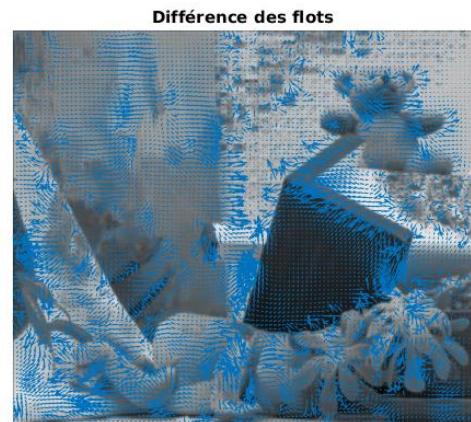


FIGURE 41 – Différence des deux flots

Nous observons que les différences sont nombreuses. Le flot affiné permet d'avoir des informations plus précises au niveau de l'intersection basse de la cheminée et du toit, du coin droit du toit, des motifs sur les tapisseries à gauche ...

```

function [u v] = HS_affine(I1,I2,N,sigma,nu,alpha,eps,niter)

n = size(I1);

% Filtre
I1 = gauss(I1,N,sigma);
I2 = gauss(I2,N,sigma);
[tmpu tmpv] = HS(I1,I2,alpha,eps,niter);

% Redimensionnement
I1r = redim(I1,nu);
I2r = redim(I2,nu);

% Calcul flot h réduit
[ur vr] = HS(I1r,I2r,alpha,eps,niter);
% Remise du flot à taille
uh = redim(ur,1/nu);
vh = redim(vr,1/nu);

% Précaution dans les erreurs d'arrondi
uh = uh(1:n(1),1:n(2));
vh = vh(1:n(1),1:n(2));

% Décalage de l'image2
I3 = I2;
for i=1:n(1)
    I3(i,:) = linearInterp(I2(i,:),-uh(i,:));
end
for i=1:n(2)
    I3(:,i) = linearInterp(I3(:,i)',-vh(:,i)')';
end

% Calcul de k
[uk vk] = HS(I1,I3,alpha,eps,niter);

u = uh + uk;
v = vh + vk;

plotFlow(u,v,I1,4);
title('Flot affiné');
plotFlow(tmpu,tmpv,I1,4);
title('Flot initial');
plotFlow(u-tmpu,v-tmpv,I1,4);
title('Différence des flots');
end

```

Code 14 – Implémentation de la méthode de Horn et schunck affinée

## 7 Conclusion

Ce projet a été une véritable source de motivation pour nous. De part, la variété des thèmes abordés et des concepts mis en jeux, nous avons pu mettre en oeuvre un certain nombre de compétences acquises dans notre formation et des compétences liées au monde de la recherche.

Il est satisfaisant de mettre en pratique nos connaissances théoriques sur un sujet qui a des débouchés concrets : voiture autonome, vision numérique, médical...



FIGURE 42 – Au revoir Teddy, en interpolation Spline

Néanmoins, nous sommes quelque peu frustré que la seconde méthode n'ait pas eu les résultats escomptées, et que nous n'ayons pas eu le temps d'implémenter cette méthode en éléments finis, comme initialement proposé dans le sujet.

Également, nous aurions bien aimé explorer l'ouverture proposée dans le projet et implémenter les méthodes de manière pyramidale , ainsi que d'explorer la possibilité de traiter des images couleurs.

## A Démonstration des équations d'Euler Lagrange

Démontrons les équations d'Euler-Lagrange (Théorème ??). Commençons par démontrer le

### Lemme :

Soit  $f \in C^0([a, b], \mathbb{R})$  telle que  $\forall h \in C^1([a, b], \mathbb{R})$  et  $h(a) = h(b) = 0$  :

$$\int_a^b f(t)h(t)dt = 0$$

Alors  $f$  est identiquement nulle sur  $[a, b]$ .

### Preuve :

Raisonnons par l'absurde.

Soit  $f$  satisfaisant les hypothèses. Supposons qu'il existe  $x \in [a, b]$  tel que  $f(x) > 0$  (Voir figure ??).

Alors par continuité,  $f$  est strictement positive sur un voisinage  $\mathcal{V}$  de  $x$ .

Prenons une fonction  $v \in C^1([a, b], \mathbb{R})$  qui est strictement positive sur  $\mathcal{V}$ . Par exemple,

$$v(x) = \begin{cases} (\beta - x)^2(x - \alpha)^2 & \text{si } x \in \mathcal{V} = [\alpha, \beta] \\ 0 & \text{sinon} \end{cases}$$

Alors la fonction produit  $fv$  est positive sur  $\mathcal{V}$  et nulle ailleurs donc son intégrale est strictement positive, ce qui est contraire à notre hypothèse.

Donc  $\forall x \in ]a, b[, f(x) \leq 0$ .

Le même argument en supposant l'existence d'un  $x$  tel que  $f(x) < 0$  appliqué à  $-f$  montre que  $\forall x \in ]a, b[, f(x) \geq 0$ .

Ainsi,  $f$  est identiquement nulle sur  $]a, b[$ , et donc sur  $[a, b]$  par continuité.  $\square$

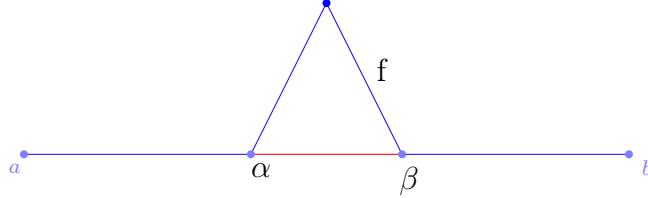


FIGURE 43 – Graphique de la fonction  $f$  absurde

Nous pouvons à présent nous attaquer à la preuve du théorème ??.

### Preuve :

Soit  $L : (q(t), \dot{q}(t), t) \mapsto L(q(t), \dot{q}(t), t)$ . Prenons  $q(t)$  tel que l'action  $A = \int_{t_0}^{t_1} L(q(t), \dot{q}(t))dt$  soit *minimale*. Montrons alors que  $L$  vérifie l'équation d'Euler-Lagrange.

Considérons une petite variation de  $q(t)$  :  $q_\epsilon(t) = q(t) + \epsilon\eta(t)$  (Voir Figure ??). Où,  $\eta : \mathbb{R} \rightarrow \mathbb{R}^n$  telle que :  $\eta \in C^1$  et  $\eta(t_0) = \eta(t_1) = 0$ . (Voir Figure )

Soit l'action perturbée :

$$A_\epsilon = \int_{t_0}^{t_1} L(q_\epsilon(t), \dot{q}_\epsilon(t), t) dt$$

Soit :

$$A_\epsilon = \int_{t_0}^{t_1} L(q(t) + \epsilon\eta(t), \dot{q}(t) + \epsilon\dot{\eta}(t), t) dt$$

Calculons la variation de l'action par rapport à la perturbation de  $q(t)$  :

$$\frac{dA_\epsilon}{d\epsilon} = \int_{t_0}^{t_1} \frac{\partial L}{\partial \epsilon}(q(t) + \epsilon\eta(t), \dot{q}(t) + \epsilon\dot{\eta}(t), t) dt$$

On applique la règle de la chaîne avec  $q_{\epsilon_i} = q_i(t) + \epsilon\eta_i(t)$

$$\begin{aligned} \frac{dA_\epsilon}{d\epsilon} &= \int_{t_0}^{t_1} \sum_{i=1}^n \left( \frac{\partial(q_i(t) + \epsilon\eta_i(t))}{\partial \epsilon} \frac{\partial L}{\partial q_{\epsilon_i}}(q_\epsilon(t), \dot{q}_\epsilon(t), t) + \frac{\partial(\dot{q}_i(t) + \epsilon\dot{\eta}_i(t))}{\partial \epsilon} \frac{\partial L}{\partial \dot{q}_{\epsilon_i}}(q_\epsilon(t), \dot{q}_\epsilon(t), t) \right) dt \\ &= \int_{t_0}^{t_1} \sum_{i=1}^n \left( \eta_i(t) \frac{\partial L}{\partial q_{\epsilon_i}}(q_\epsilon(t), \dot{q}_\epsilon(t), t) + \dot{\eta}_i(t) \frac{\partial L}{\partial \dot{q}_{\epsilon_i}}(q_\epsilon(t), \dot{q}_\epsilon(t), t) \right) dt \end{aligned}$$

Posons  $\epsilon = 0$ , alors comme  $x(t)$  réalise le minimum de  $A$  nous avons :  $\frac{dA_0}{d\epsilon} = 0$ . Après une intégration par partie sur le second membre, nous obtenons :

$$0 = \int_{t_0}^{t_1} \sum_{i=1}^n \left( \eta_i(t) \frac{\partial L}{\partial q_{\epsilon_i}}(q_\epsilon(t), \dot{q}_\epsilon(t), t) \right) dt + \sum_{i=1}^n \left( \left[ \eta_i(t) \frac{\partial L}{\partial \dot{q}_i} \right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \eta_i(t) \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i}(q(t), \dot{q}(t), t) dt \right)$$

Par hypothèse sur  $\eta$ , nous avons alors :

$$0 = \sum_{i=1}^n \left( \int_{t_0}^{t_1} \eta_i(t) \left( \frac{\partial L}{\partial q_i}(q(t), \dot{q}(t), t) - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i}(q(t), \dot{q}(t), t) \right) dt \right)$$

Appliquons alors le lemme ?? à une fonction  $\eta$  bien choisie :

$$\eta(t) = \begin{cases} \eta_j(t) = 0 & j \neq i \\ \eta_j(t) \neq 0 & j = i \end{cases}$$

Comme  $\eta_i(t)$  est quelconque et par hypothèse  $\eta(t) \in C^1$ , alors d'après notre lemme, la fonction entre parenthèse doit être identiquement nulle. Nous obtenons alors les équations d'Euler - Lagrange :

$$\forall i \in \llbracket 1; n \rrbracket \quad \frac{\partial L}{\partial q_i}(q(t), \dot{q}(t), t) - \frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i}(q(t), \dot{q}(t), t) = 0$$

□

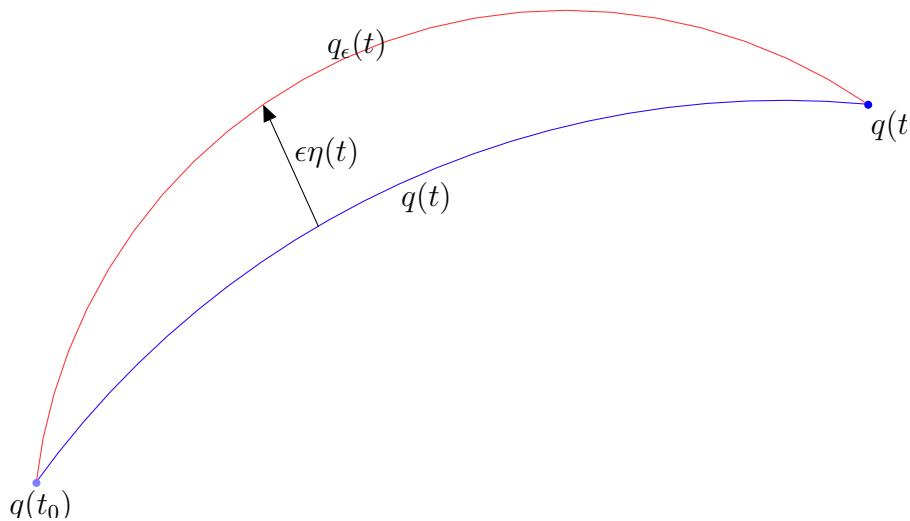


FIGURE 44 – Variation des variables

Nous résistons pas à l'envie de donner un exemple de la puissance de ces équations. Un exemple basique mais très instructif.

**Exemple :** Calculons la distance minimale entre deux point dans le plan. Plaçons en coordonnées curviligne et supposons par simplicité que l'instant de départ est 0 et l'instant d'arrivée est 1. L'action à minimiser est alors :

$$A = \int_0^1 ds$$

Avec  $ds = \sqrt{dx^2 + dy^2}$ . En écrivant  $y = y(x)$  nous en déduisons le lagrangien associé à cette action :

$$L(y(x), \dot{y}(x), x) = \sqrt{1 + \dot{y}(x)^2}$$

On applique alors les équation d'Euler - Lagrange pour en déduire :

$$\begin{aligned} \frac{\partial L}{\partial y}(y, \dot{y}, x) - \frac{d}{dx} \frac{\partial L}{\partial \dot{y}}(y, \dot{y}, x) &= 0 \\ \frac{d}{dx} \frac{2\dot{y}(x)}{2\sqrt{1 + \dot{y}(x)^2}} &= 0 \\ \frac{\dot{y}(x)}{\sqrt{1 + \dot{y}(x)^2}} &= \lambda \\ \dot{y}(x)^2 &= \lambda(1 + \dot{y}(x)^2) \\ \dot{y}(x) &= a \end{aligned}$$

On a alors :

$$y(x) = ax + b$$

Où les constantes  $a$  et  $b$  peuvent être déterminées avec des conditions initiales. Sans surprise, nous retrouvons que le plus court chemin dans le plan est une ligne droite.

## B Obtention du schéma de discréétisation du système de Horn et Schunck

Montrons comment est obtenu le système discréétisé des équations de Horn et Schunck (??). Nous partons du système d'équations continues obtenus par les équations d'Euler Lagrange (??). Nous notons par  $I_x, I_y$  et  $I_t$  les versions discréétisées des dérivées partielles de  $I$ . Enfin nous prenons la version discréétisé du Laplacien :  $\Delta f \approx \bar{f} - f$  avec  $\bar{f} =$ .

$$\begin{aligned} & \begin{cases} I_x(I_x u + I_y v + I_t) = \alpha^2(\Delta u) \\ I_y(I_x u + I_y v + I_t) = \alpha^2(\Delta v) \end{cases} \\ \iff & \begin{cases} I_x^2 u + I_x I_y v + I_x I_t = \alpha^2(\bar{u} - u) \\ I_y I_x u + I_y^2 v + I_y I_t = \alpha^2(\bar{v} - v) \end{cases} \\ \iff & \begin{cases} (\alpha^2 + I_x^2)u + I_x I_y v = \alpha^2 \bar{u} - I_x I_t \\ I_y I_x u + (\alpha^2 + I_y^2)v = \alpha^2 \bar{v} - I_y I_t \end{cases} \\ \iff & \begin{pmatrix} \alpha^2 + I_x^2 & I_x I_y \\ I_x I_y & \alpha^2 + I_y^2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \alpha^2 \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} - \begin{pmatrix} I_x \\ I_y \end{pmatrix} I_t \end{aligned}$$

Le déterminant de la matrice est non nul :

$$\begin{aligned} & \iff [(I_x^2 + \alpha^2)(I_y^2 + \alpha^2) - I_x^2 I_y^2] \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \alpha^2 + I_y^2 & -I_x I_y \\ -I_x I_y & \alpha^2 + I_x^2 \end{pmatrix} \left( \begin{pmatrix} \alpha^2 \bar{u} \\ \alpha^2 \bar{v} \end{pmatrix} - \begin{pmatrix} I_x \\ I_y \end{pmatrix} I_t \right) \\ & \iff [\alpha^2 I_x^2 + \alpha^4 + \alpha^2 I_y^2] \begin{pmatrix} u \\ v \end{pmatrix} = \alpha^2 \begin{pmatrix} (\alpha^2 + I_y^2)\bar{u} - I_x I_y \bar{v} \\ -I_x I_y \bar{u} + (\alpha^2 + I_x^2)\bar{v} \end{pmatrix} - \begin{pmatrix} (\alpha^2 + I_y^2)I_x - I_x I_y^2 \\ -I_x^2 I_y + (\alpha^2 + I_x^2)I_y \end{pmatrix} I_t \\ & \iff \alpha^2 [I_x^2 + \alpha^2 + I_y^2] \begin{pmatrix} u \\ v \end{pmatrix} = \alpha^2 \begin{pmatrix} (\alpha^2 + I_y^2)\bar{u} - I_x I_y \bar{v} \\ -I_x I_y \bar{u} + (\alpha^2 + I_x^2)\bar{v} \end{pmatrix} - \alpha^2 \begin{pmatrix} I_x \\ I_y \end{pmatrix} I_t \\ & \iff \begin{cases} (I_x^2 + \alpha^2 + I_y^2)u = (\alpha^2 + I_y^2)\bar{u} - I_x I_y \bar{v} - I_x I_t \\ (I_x^2 + \alpha^2 + I_y^2)v = (\alpha^2 + I_x^2)\bar{v} - I_x I_y \bar{u} - I_y I_t \end{cases} \\ & \iff \begin{cases} (I_x^2 + \alpha^2 + I_y^2)u = (I_x^2 + \alpha^2 + I_y^2)\bar{u} - I_x^2 \bar{u} - I_x I_y \bar{v} - I_x I_t \\ (I_x^2 + \alpha^2 + I_y^2)v = (I_x^2 + \alpha^2 + I_y^2)\bar{v} - I_y^2 \bar{v} - I_x I_y \bar{u} - I_y I_t \end{cases} \end{aligned}$$

Ce qui nous donne bien la forme annoncée :

$$\begin{cases} (\alpha^2 + I_x^2 + I_y^2)(u - \bar{u}) = -I_x(I_x \bar{u} + I_y \bar{v} + I_t) \\ (\alpha^2 + I_x^2 + I_y^2)(v - \bar{v}) = -I_y(I_x \bar{u} + I_y \bar{v} + I_t) \end{cases}$$

## C Obtention de la version améliorée du schéma de discréétisation du système de Horn et Schunck

Donnons la version discréétisé du système d'équations obtenues par Euler - Lagrange (Equations ??). Les calculs sont sensiblement les mêmes que précédemment, c'est pourquoi nous nous permettons d'aller un peu plus vite. Nous rappelons que  $A = \frac{1}{I_x^2 + I_y^2 + 2\gamma}$  et que le laplacien est approchée par  $\Delta f \approx \bar{f} - f$  avec  $\bar{f} =$ .

$$\begin{cases} I_x^2 u + I_x I_y v + I_x I_t = A\alpha^2(I_y^2 + \gamma)(\bar{u} - u) + A\alpha^2(\bar{v} - v) \\ I_x I_y u + I_y^2 v + I_y I_t = A\alpha^2(I_x^2 + \gamma)(\bar{v} - v) + A\alpha^2(\bar{u} - u) \end{cases} \iff \begin{cases} (I_x^2 + A\alpha^2(I_y^2 + \gamma))u + (I_x I_y - A\alpha^2 I_x I_y)v = A\alpha^2(I_y^2 + \gamma)\bar{u} - A\alpha^2 I_x I_y \bar{v} - I_x I_t \\ (I_y I_x - A\alpha^2 I_x I_y)u + (I_y^2 + A\alpha^2(I_x^2 + \gamma))v = A\alpha^2(I_x^2 + \gamma)\bar{v} - A\alpha^2 I_x I_y \bar{u} - I_y I_t \end{cases}$$

C'est à dire, sous forme matricielle :

$$\begin{pmatrix} I_x^2 + A\alpha^2(I_y^2 + \gamma) & I_x I_y - A\alpha^2 I_x I_y \\ I_y I_x - A\alpha^2 I_x I_y & I_y^2 + A\alpha^2(I_x^2 + \gamma) \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} A\alpha^2(I_y^2 + \gamma) & -A\alpha^2 I_x I_y \\ -A\alpha^2 I_x I_y & A\alpha^2(I_x^2 + \gamma) \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} - \begin{pmatrix} I_x \\ I_y \end{pmatrix} I_t$$

Le déterminant de la matrice de gauche est non nul vaut :  $A\alpha^2 [(I_x^2 + I_y^2 + \gamma)(A\alpha^2 + I_x^2 + I_y^2)]$ . Calculons le résultat du second membre :

$$\begin{pmatrix} I_y^2 + A\alpha^2(I_x^2 + \gamma) & -I_x I_y + A\alpha^2 I_x I_y \\ -I_y I_x + A\alpha^2 I_x I_y & I_x^2 + A\alpha^2(I_y^2 + \gamma) \end{pmatrix} \begin{pmatrix} A\alpha^2(I_y^2 + \gamma) & -A\alpha^2 I_x I_y \\ -A\alpha^2 I_x I_y & A\alpha^2(I_x^2 + \gamma) \end{pmatrix} = \begin{pmatrix} A\alpha^2(I_x^2 + I_y^2 + \gamma)(A\alpha^2\gamma + I_y^2) & -A\alpha^2 I_x I_y (I_x^2 + I_y^2 + \gamma) \\ -2A2 I_x I_y (I_x^2 + I_y^2 + \gamma) & A\alpha^2(I_x^2 + I_y^2 + \gamma)(A\alpha^2 + I_x^2) \end{pmatrix}$$

$$\begin{pmatrix} I_y^2 + A\alpha^2(I_x^2 + \gamma) & -I_x I_y + A\alpha^2 I_x I_y \\ -I_y I_x + A\alpha^2 I_x I_y & I_x^2 + A\alpha^2(I_y^2 + \gamma) \end{pmatrix} \begin{pmatrix} I_x \\ I_y \end{pmatrix} = \begin{pmatrix} A\alpha^2 I_x (I_x^2 + I_y^2 + \gamma) \\ A\alpha^2 I_y (I_x^2 + I_y^2 + \gamma) \end{pmatrix}$$

Somme toute, nous obtenons les système :

$$(A\alpha^2\gamma + I_x^2 + I_y^2) \begin{cases} u = (A\alpha^2\gamma + I_y^2)\bar{u} - I_x I_y \bar{v} - I_x I_t \\ v = -I_x I_y \bar{u} + (A\alpha^2\gamma + I_x^2)\bar{v} - I_y I_t \end{cases}$$

$$(A\alpha^2\gamma + I_x^2 + I_y^2) \begin{cases} u = (A\alpha^2\gamma + I_y^2 + I_x^2)\bar{u} - I_x^2 \bar{u} - I_x I_y \bar{v} - I_x I_t \\ v = -I_x I_y \bar{u} + (A\alpha^2\gamma + I_x^2 + I_y^2)\bar{v} - I_y^2 \bar{v} - I_y I_t \end{cases}$$

$$\iff \begin{cases} (A\alpha^2\gamma + I_x^2 + I_y^2)(u - \bar{u}) = -I_x(I_x \bar{u} + I_y \bar{v} + I_t) \\ (A\alpha^2\gamma + I_x^2 + I_y^2)(v - \bar{v}) = -I_y(I_x \bar{u} + I_y \bar{v} + I_t) \end{cases}$$

## D La spline comme interpolant de courbure minimale

La démonstration est celle donnée dans le cours d'Approximation et Design Géométrique. Elle repose sur le

**Lemme :**

Soit  $E$  l'ensemble des fonctions  $f \in \mathbb{C}^2([a, b])$  telles que :

$$\begin{cases} f(\tau_i) = y_i & \text{donné, pour } i \in \{1, \dots, l\} \\ f'(a) = \alpha & \text{donné} \\ f'(b) = \beta & \text{donné} \end{cases} \quad (14)$$

Alors pour  $\phi \in \mathbb{S}^{3,\tau} \cap E$ , nous posons  $e = f - \phi$  l'erreur dans l'approximation de  $f$  par  $\phi$ . Nous avons le résultat suivant :

$$\forall g \in \mathbb{S}^{1,\tau} \quad \int_a^b e''(x)g(x)dx = 0$$

**Preuve :**

$$\int_a^b e''(x)g(x)dx = \sum_{i=0}^{l-1} \int_{\tau_i}^{\tau_{i+1}} e''(x)g(x)dx = \underbrace{\sum_{i=0}^{l-1} [e'(x)g(x)]_{\tau_i}^{\tau_{i+1}}}_{0} - \sum_{i=0}^{l-1} \int_{\tau_i}^{\tau_{i+1}} e'(x)g'(x)dx$$

Où le premier terme est nul car c'est une somme télescopique et  $e'(\tau_0) = f'(a) - \phi'(a) = 0$  de même en  $\tau_l$ .

Or,  $g \in \mathbb{S}^{1,\tau}$  donc sur  $[\tau_i, \tau_{i+1}]$ ,  $g'(x) = \lambda_i$ . Ainsi,

$$\int_a^b e''(x)g(x)dx = - \sum_{i=0}^{l-1} \lambda_i \int_{\tau_i}^{\tau_{i+1}} e'(x)dx = - \sum_{i=0}^{l-1} \lambda_i (e(\tau_{i+1}) - e(\tau_i)) = 0$$

car  $e(\tau_i) = f(\tau_i) - \phi(\tau_i) = 0$ . □

On en déduit alors la proposition annoncée :

**Proposition D.1 :**

Si  $\phi \in \mathbb{S}^{3,\tau} \cap E$  alors nous avons :

$$\int_a^b (\phi''(t))^2 dt = \min_{f \in E} \int_a^b (f''(t))^2 dt$$

**Preuve :**

On a  $f = \phi + e$ .

$$\int_a^b (f''(t))^2 dt = \int_a^b (\phi''(t))^2 dt + \underbrace{\int_a^b e''(t)\phi''(t)dt}_{0 \text{ car } \phi \in \mathbb{S}^{3,\tau} \text{ donc } \phi'' \in \mathbb{S}^{1,\tau}} + \int_a^b (e''(t))^2 dt$$

D'où l'inégalité,

$$\int_a^b (f''(t))^2 dt \geq \int_a^b (\phi''(t))^2 dt$$

Soit,

$$\int_a^b (\phi''(t))^2 dt = \min_{f \in E} \int_a^b (f''(t))^2 dt$$

Le minimum est atteint quand  $e'' = 0$  car  $e''$  est continue. Or,  $e = f - \phi \in C^2([a, b])$ . Nous avons de plus,  $e'(a) = e'(b) = 0$  et  $e(\tau_i) = 0, \forall i \in \{1, \dots, l\}$  donc  $e \equiv 0$ .  $\square$

## E Calcul de la *mère Spline*

Procèdons au calcul de la *mère Spline*, nous partons de la définitions de nos noeuds ( $\forall i, \tau_i = i$ ) et de la formule donnant les B-Splines.

$$\begin{aligned}
B_{-2,3} &= \frac{x - \tau_i}{\tau_{i+3} - \tau_i} B_{-2,2} + \frac{\tau_{i+4} - x}{\tau_{i+4} - \tau_{i+1}} B_{-1,2} \\
&= \frac{x - \tau_i}{3} \left( \frac{x - \tau_i}{\tau_{i+2} - \tau_i} B_{-2,1} + \frac{\tau_{i+3} - x}{\tau_{i+3} - \tau_{i+1}} B_{-1,1} \right) + \\
&\quad \frac{\tau_{i+4} - x}{3} \left( \frac{x - \tau_{i+1}}{\tau_{i+3} - \tau_{i+1}} B_{-1,1} + \frac{\tau_{i+4} - x}{\tau_{i+4} - \tau_{i+2}} B_{0,1} \right) \\
&= \frac{x - \tau_i}{3} \left( \frac{x - \tau_i}{2} ((x - \tau_{-2}) \mathbb{1}_{[\tau_{-2}, \tau_{-1}[} + (\tau_0 - x) \mathbb{1}_{[\tau_{-1}, \tau_0[}] \right) + \\
&\quad \frac{\tau_{i+3} - x}{2} ((x - \tau_{-1}) \mathbb{1}_{[\tau_{-1}, \tau_0[} + (\tau_1 - x) \mathbb{1}_{[\tau_0, \tau_2[}] \right) + \\
&\quad \frac{\tau_{i+4} - x}{3} \left( \frac{x - \tau_{i+1}}{2} ((x - \tau_{-1}) \mathbb{1}_{[\tau_{-1}, \tau_0[} + (\tau_1 - x) \mathbb{1}_{[\tau_0, \tau_2[}] \right) + \\
&\quad \frac{\tau_{i+4} - x}{2} ((x - \tau_0) \mathbb{1}_{[\tau_0, \tau_1[} + (\tau_2 - x) \mathbb{1}_{[\tau_1, \tau_2[}] \right) \\
&= \frac{(x+2)^3}{6} \mathbb{1}_{[-2, -1[} + \frac{1}{6} (-x(x+2)^2 + (x+2)(1-x)(x+1)) \mathbb{1}_{[-1, 0[} + \\
&\quad \frac{1}{6} ((x+2)(1+x)^2 + (2-x)(1-x)(1+x) + (2-x)^2 x) \mathbb{1}_{[0, 1[} + \frac{(2-x)^3}{6} \mathbb{1}_{[1, 2[}
\end{aligned}$$

Ce qui nous donne bien la formule annoncée :

$$\begin{aligned}
6B_{-2,3}(x) &= (x+2)^3 \mathbb{1}_{[-2, -1[} + (-3x^2(x+2) + 4) \mathbb{1}_{[-1, 0[} \\
&\quad + (3(x-2)x^2 + 4) \mathbb{1}_{[0, 1[} + (2-x^3) \mathbb{1}_{[1, 2[}
\end{aligned}$$

## F Bonus : l'équation de la chaleur appliquée sur une image par éléments finis avec FreeFem++

Il était proposé dans le sujet initial du projet d'implémenter la méthode améliorée de Horn et Schunck (section ??) en éléments finis avec le logiciel FreeFem++. Dans cette optique, que nous avons par la suite laissée de côté, nous avions commencé à prendre en main ce logiciel et implémenté un programme qui applique l'équation de la chaleur sur une image. Nos résultats sont présentés dans la suite.

La première étape est de construire un maillage sur une image. Pour varier, avec le reste du rapport, nous utilisons l'image suivante :



FIGURE 45 – Image utilisée pour les éléments finis

Le maillage est obtenu en deux étapes, la première est de générer un maillage uniforme sur le domaine, ce qui crée un maillage très dense : 226592 points. Puis nous appliquons la fonction *adaptmesh* qui ajuste le maillage à l'image. Nous obtenons alors le maillage suivant : Pour un total de 26445 triangles et 13 414 sommets.

Le code de maillage est le suivant :

```
string filename1="image1"; // Noms du fichier
textes contenant l'image

mesh Sh;
{
    Sh=triangulate(filename1); //On introduit un maillage
    uniforme sur l'image
}
```

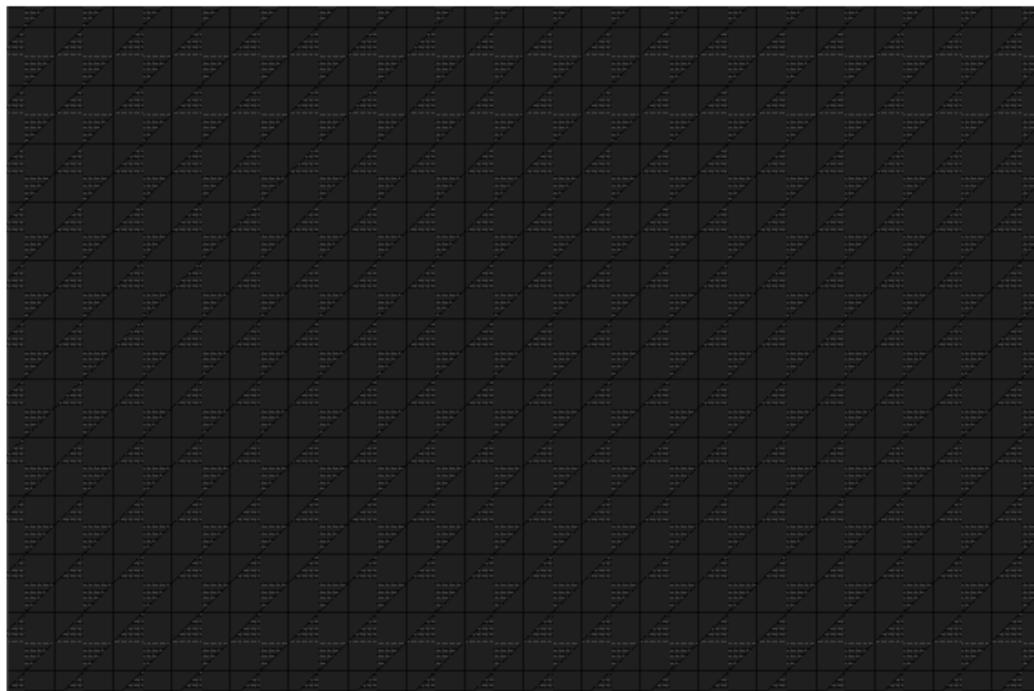


FIGURE 46 – Maillage uniforme sur l'image

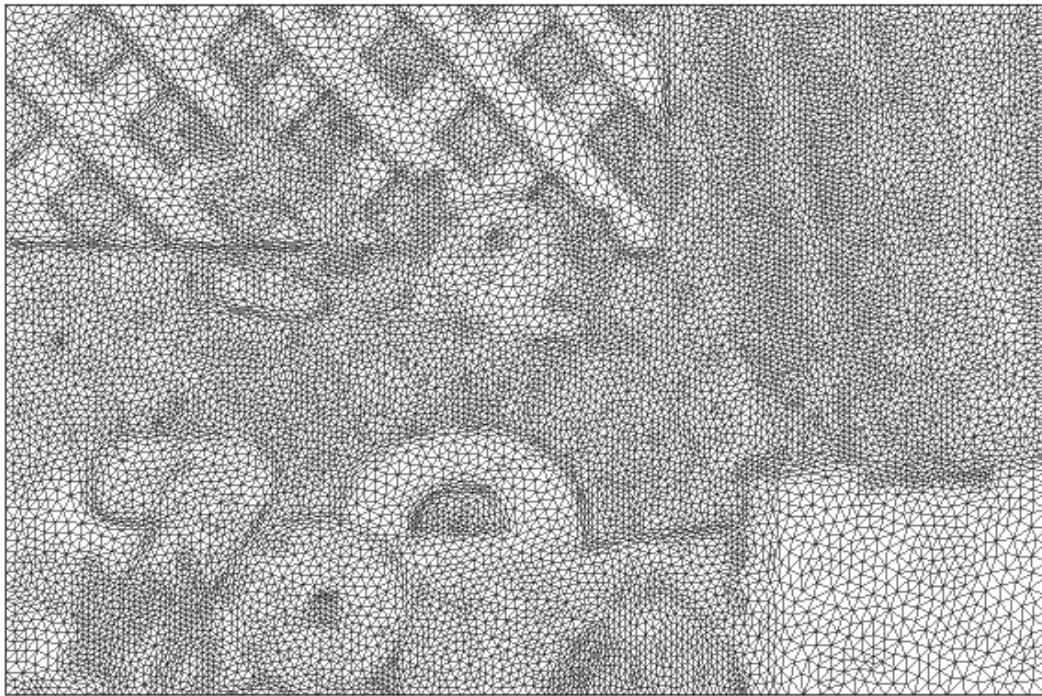


FIGURE 47 – Maillage adapté à l'image

```
fespace Vh(Sh,P1); // On souhaite travailler avec les éléments
// finis P1 sur Sh
Vh u1=0; // On définit l'image
{
    {
        // import de l' image 1
```

```

        ifstream file(filename1);
        real xx,yy;

        for(int i=0;i<u1.n;i++) {
            file >> xx >> yy >> u1[] [i];
        }
    }

mesh Th=adaptmesh(Sh,u1,err=1.e0,nbvx=10000000); // on adapte le
maillage à l'image
savemesh(Th,"img.msh");
plot(Th,wait=1); // On dessine le maillage obtenu.

```

Code 15 – Programme de création d'un maillage sur une image

Puis nous appliquons l'équation de la chaleur sur le maillage obtenu, avec comme condition initiale l'image elle-même.

```

string filename1="image1";

mesh Sh;
{
    Sh=triangulate(filename1);
    fespace Vh(Sh,P1);
    Vh u1=0;
    {
        {
            ifstream file(filename1);
            real xx,yy;
            cout << u1.n;

            for(int i=0;i<u1.n;i++) {
                file >> xx >> yy >> u1[] [i];
            }
        }
    }

    mesh Th=readmesh("img.msh");

    fespace Wh(Th,P1);
    Wh uh,vh,uh0=u1;
    macro Grad(u)[dx(u),dy(u)]//
    real dt = 0.5, Tf = 100;
    //Défiinition du problème
    problem chaleur(uh,vh) = int2d(Th)(vh*uh/dt) + int2d(Th)(
        Grad(uh)'*Grad(vh)) - int2d(Th)(uh0*vh/dt);

    int kk=0;
    for (real t=0.;t<Tf;t+=dt) {
        chaleur;
        uh0=uh;
        cout << kk;
    }
}

```

```

        if ( !(kk % 2) ) {
            plot(uh, grey=1, fill=1, OpaqueBorders=1,
                  ShowAxes=0, ps = "000"+kk+".png");
        }
        kk+=1;
}
    
```

Code 16 – Programme d'application de l'équation de la chaleur sur une image



FIGURE 48 – Image initiale  $T=0$



FIGURE 49 – image au temps  $T = 1\text{s}$



FIGURE 50 – Image au temps  $T=1\text{s}$

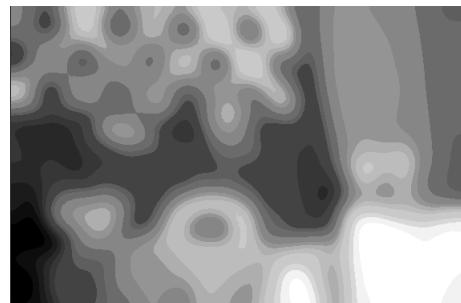


FIGURE 51 – image au temps  $T = 100\text{s}$

Nous constatons que le résultats est bien celui que nous attendons.  
 Néanmoins pour le lecteur enthousiaste de faire d'appliquer des méthodes éléments finis sur des images, nous souhaitons le mettre ne garde. en effet, ces méthodes ne nous apparaissent pas si pertinentes. Le principale inconvénient est la perte de l'information locale donné par le pixel, le fait d'utiliser des éléments finis, induit automatiquement une interpolation des des données sur les noeuds qui ne tombent pas pile sur les pixels de l'image (c'est à dire la grand majorité, sauf si on s'embête à faire un maillage rectangulaire régulier). Ce phénomène se constate sur la figure ??.

## Table des figures

## Liste des Codes