Université Paris Descartes
UFR de Mathématiques et Informatique
45, rue des Saints-Pères 75270 Paris cedex 06

UNIVERSITÉ PARIS DESCARTES

MASTER 1

PROGRAMMATION

TP "Edition d'image par Equation de Poisson"

Mise en route
Depuis un terminal (placez vous dans le dossier de votre choix) tapez les commandes suivantes pour télécharger les fichiers nécessaires au projet :

```
wget http://w3.mi.parisdescartes.fr/~jdelon/enseignement/MA106/projetedition.zip
unzip projetedition.zip
cd im
```

Le dossier im contient plusieurs images qui vous serviront pour le projet.
Pour ouvrir et lire des images sous Scilab, on utilisera la toolbox SIVP qui est installée sur les machines. Commencez par charger la toolbox dans l'interface graphique de Scilab. Une fois la toolbox chargée, les lignes suivantes vous permettent d'ouvrir et d'afficher une image dans Scilab :

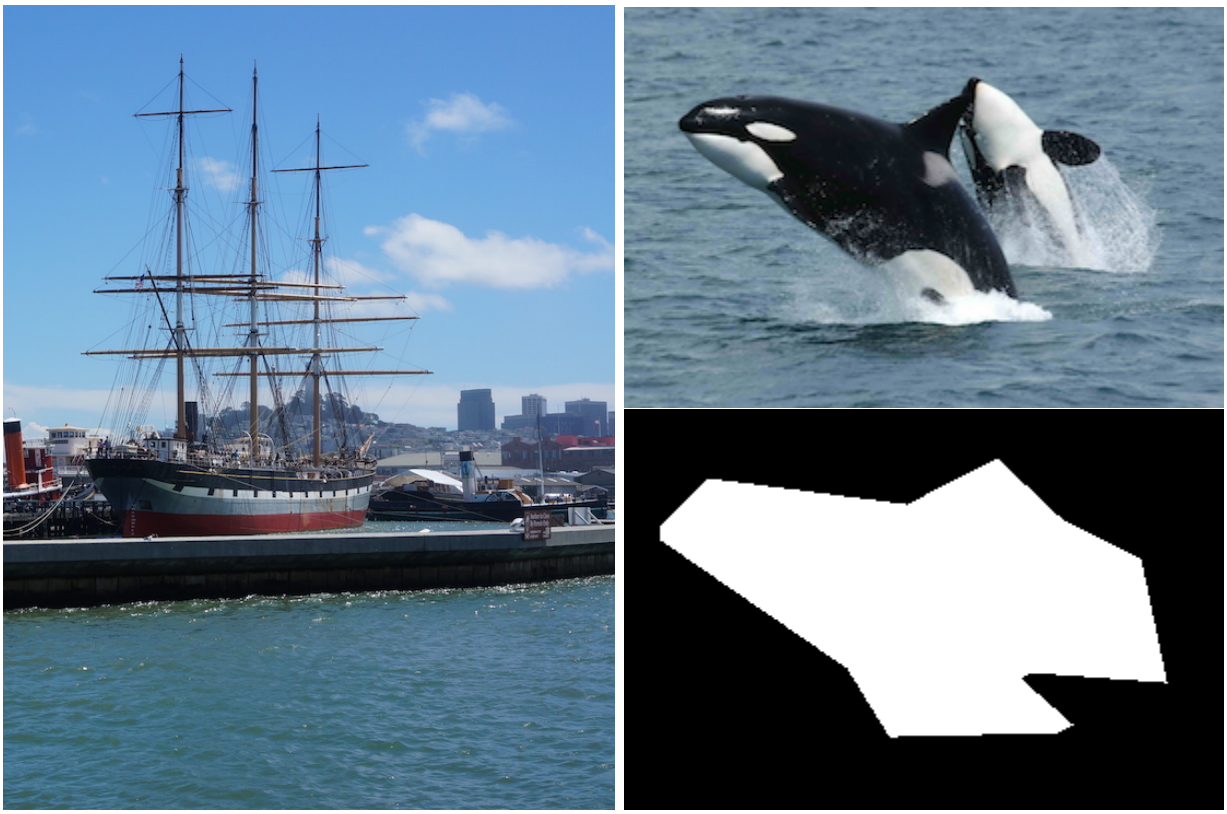
```
u0 =double(imread('sanfrancisco.jpg'));   // on ouvre l'image sanfrancisco.jpg
figure(1);imshow(u0/255);                 // on affiche l'image
```

1 Introduction

Ce projet porte sur l'édition d'images par équation de Poisson. Il est inspiré de l'article [1]. Dans ce qui suit, on considère des image discrètes prenant leurs valeurs dans $[0, 255]$. On rappelle qu'une image peut se représenter dans Scilab sous la forme d'une matrice.

Etant données deux images u et v, on souhaite importer un extrait de l'image v et coller cet extrait dans l'image u (voir la Figure 1 (a)). On commence par importer les deux images u et v dans Scilab. Pour coller un extrait de v dans u, il nous faut indiquer deux informations :

1. **Quel morceau de l'image v coller dans u ?** On le représente sous la forme d'un **masque binaire**. En Scilab, ce masque peut être représenté par une image, de la même taille que v, et constituée de 0 et de 1 (les pixels où le masque vaut 1 sont ceux qui vont être collés dans u). On peut voir l'exemple d'un tel masque binaire sur la Figure 1 (a)). Parmi les images fournies pour le projet, vous trouverez un certain nombre d'images en noir et blanc représentant ces masques. Pour créer les masques, il faut commencer par ouvrir ces images et par les transformer en matrices de booléens (qui valent T sur les points du masque et F en dehors).
2. **Où colle t'on ce morceau d'image dans u ?** On indique où l'on souhaite coller ce morceau d'image en donnant une position (x, y) représentant le pixel supérieur gauche de la région rectangulaire de u dans laquelle on va copier le morceau de v. Cette position (x, y) dans u correspond à la position $(0, 0)$ dans v.



(a) A gauche l'image u, prise à San Francisco. A droite : en haut l'image v, représentant des baleines, et en bas le masque qui indique quel morceau de l'image des baleines on veut coller dans u.

(b) A gauche, copie simple du morceau d'image. A droite, interpolation guidée par équation de Poisson.

FIGURE 1 – Exemple d'interpolation guidée. On colle un morceau de l'image des baleines dans l'image de San Francisco.

► **Question 1 :** *Soient u et v deux images en niveau de gris, x, y deux entiers et mask une image binaire de même taille que v. Ecrire une fonction* `[v]=copiecolle(u,v,mask,x,y)` *qui copie le morceau de l'image v qui est à l'intérieur du masque booléen dans l'image u en position (x, y). On pourra utiliser les lignes suivantes dans le code :*

```
[p,q]              = size(v);
ucrop              = u(x:x+p-1,y:y+q-1); // on extrait la zone de u concernée
ucrop(mask)        = v(mask);           // on copie les pixels du masque de v vers ucrop
u(x:x+p-1,y:y+q-1) = ucrop;
```

On supposera par simplicité que l'image v est plus petite que u et que la position (x, y) est choisie de manière à ce que la zone rectangulaire $[x : x + p - 1, y : y + q - 1]$ soit bien contenue dans l'image u. Faire une fonction similaire pour les images couleur, en appliquant la fonction `copiecolle` *séparément à chaque composante couleur.*

2 Interpolation guidée

Pour simplifier, on suppose dans ce qui suit qu'on a déjà extrait la région rectangulaire de u où l'on souhaite coller le morceau de v. Les images u et v sont donc supposées de même taille.

Le problème dans le résultat de la fonction précédente est que les images u et v ne se « recollent » pas bien sur les bords de la région modifiée (voir la Figure 1 (b) à gauche).

Appelons $\mathcal{S}$ le support de l'image u et Ω la région que l'on souhaite modifier (le masque est l'indicatrice de Ω, c'est-à-dire qu'il vaut 1 sur les points de Ω et 0 ailleurs). On considère que chaque pixel (i, j) a 4 voisins (droit, bas, gauche, haut) et on note $V(i, j)$ l'ensemble de ces 4 voisins. Notons $\partial\Omega$ le bord de Ω, c'est-à-dire l'ensemble des pixels de $\mathcal{S}$ qui ont un voisin dans Ω :

$$\partial\Omega = \{(i, j) \in \mathcal{S};\ V(i, j) \cap \Omega \neq \emptyset\}.$$

Au lieu de copier bêtement le morceau d'image, on propose d'interpoler l'image u sur le domaine Ω en suivant le mieux possible le champ des gradients de l'image v sur ce domaine et en conservant les valeurs de u sur le bord $\partial\Omega$ (voir la Figure 2). Pour cela, on va essayer de résoudre le problème de minimisation suivant :

$$\arg \min_{f_{|\partial\Omega} = u_{|\partial\Omega}} \int_\Omega \|\nabla f - \nabla v\|^2, \tag{1}$$

où ∇f est le gradient de f et où la contrainte $f_{|\partial\Omega} = u_{|\partial\Omega}$ signifie que f doit être égale à u sur le bord $\partial\Omega$.

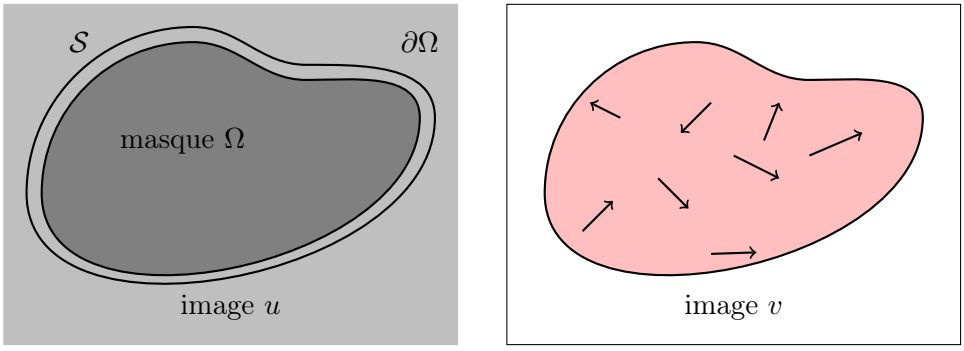


FIGURE 2 – On cherche à interpoler l'image u sur le domaine Ω en suivant le mieux possible le champ des gradients de l'image v sur ce domaine et en conservant les valeurs de u sur le bord $\partial\Omega$.

Si l'on réécrit se problème de manière discrète, on obtient

$$\min_{\forall (i,j)\in\partial\Omega,\ f(i,j)=u(i,j)} \sum_{(i,j)\in\Omega} \sum_{(k,l)\in V(i,j)\cap\Omega} |f(i, j) - f(k, l) - v(i, j) + v(k, l)|^2. \tag{2}$$

La solution f de ce problème de minimisation vérifie le système d'équations suivantes :

$$\begin{cases} \forall (i,j) \in \partial\Omega & f(i,j) = u(i,j) \\ \forall (i,j) \in \Omega, & \underbrace{4f(i,j) - \sum_{(k,l)\in V(i,j)} f(k,l)}_{-\Delta f(i,j)} = \underbrace{4v(i,j) - \sum_{(k,l)\in V(i,j)} v(k,l)}_{-\Delta v(i,j)} \end{cases}$$

Les termes de droite sont fixés par les images u et v (données), l'image f sur la région Ω est donc solution d'un système linéaire du type

$$Af = b.$$

► **Question 2 :** *Que valent la matrice A et le vecteur b dans le système précédent ?*
Afin de résoudre ce système et de trouver f, on propose d'utiliser la méthode itérative de Jacobi, qui consiste à partir d'un $f^{(0)}$ initial et d'itérer sur p

$$f^{(p+1)} = D^{-1}(Nf^{(p)} + b)$$

avec D la diagonale de la matrice A et $N = D - A$. Sous de bonnes conditions sur A, la suite $f^{(p)}$ converge vers la solution du système $Af = b$ lorsque p tend vers l'infini.

► **Question 3 :** *Supposons que l'on initialise la méthode en posant*

$$\begin{cases} \forall (i,j) \in \Omega, & f^{(0)}(i,j) = v(i,j) \\ \forall (i,j) \notin \Omega, & f^{(0)}(i,j) = u(i,j). \end{cases}$$

Expliquer pourquoi l'itération p de la méthode de Jacobi pour notre problème s'écrit alors

$$\forall (i,j) \in \Omega, \quad f^{(p+1)}(i,j) = \frac{1}{4}\left((K \star f^{(p)})(i,j) - \Delta v(i,j)\right), \tag{3}$$

où Δv est le laplacien discret de l'image v, $\star$ est une convolution discrète (fonction `conv2` *sous Scilab) et K est le noyau*

$$K = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

► **Question 4 :** *Ecrire une fonction* `[fx,fy,]=grad(f)` *qui prenne en entrée une image discrète f et renvoie deux images de même taille que f représentant le gradient ∇f de f : fx est la dérivée en x et fy la dérivée en y. On utilisera le schéma suivant :*

$$fx(i,j) = \begin{cases} f(i+1,j) - f(i,j) & si\ i < n \\ 0 & sinon \end{cases}, \quad fy(i,j) = \begin{cases} f(i,j+1) - f(i,j) & si\ j < m \\ 0 & sinon. \end{cases}$$

► **Question 5 :** *Ecrire une fonction* `d = div(gx,gy)` *qui étant donné un champ de vecteurs $g = [gx, gy]$ (g est une matrice $n \times m \times 2$) renvoie la divergence de ce champ de vecteurs. On pourra utiliser le schéma discret suivant :*

$$div(g)_{(i,j)} = \begin{cases} gx(i,j) - gx(i-1,j) & si\ 1 < i < n \\ gx(i,j) & si\ i = 1 \\ -gx(i-1,j) & si\ i = n \end{cases} + \begin{cases} gy(i,j) - gy(i,j-1) & si\ 1 < j < m \\ gy(i,j) & si\ j = 1 \\ -gy(i,j-1) & si\ j = m \end{cases}$$

► **Question 6 :** *Ecrire une fonction* `lap = laplacien(v)` *qui calcule le laplacien de l'image discrète v. On rappelle que $\Delta v = \mathrm{div}(\nabla v)$ donc vous pouvez l'écrire très simplement à partir des fonctions* ***grad*** *et* ***div*** *précédentes.*

► **Question 7 :** *Ecrire une nouvelle fonction* `[Z]=interpolationguidee(u,v,mask,niter)` *qui prend en entrée deux images de même taille, en niveau de gris, un masque binaire et un nombre d'itérations, et applique le schéma discret* (3) *pour interpoler l'image u sur la région définie par le masque grâce au gradient de v. Eviter au maximum les boucles for, en dehors de celle indispensable pour les itérations de la méthode de Jacobi. La fonction pourra avoir la structure suivante :*

```
function f = interpolationguidee(u, v, mask, niter)
  //initialisation
      A REMPLIR

 // itérations de la méthode de Jacobi
  for k = 1:niter
       A REMPLIR ;
  end
endfunction
```

A la fin de la fonction, seuillez l'image f obtenue pour vous assurer que le résultat est bien entre 0 et 255 grâce à la commande :

```
f =  max(min(f,255),0);
```

► **Question 8 :** *Utiliser la fonction de la question 1 pour étendre la fonction précédente au cas où u n'a pas la même taille que v et où l'on précise la position* (x, y)*. Etendre également la fonction aux images en couleur en l'appliquant séparément à chaque canal couleur.*

► **Question 9 :** *Tester la fonction précédente à l'aide des différents exemples d'images et de masques fournis dans le répertoire im. Le nombre d'itérations niter doit être choisi suffisamment grand (de l'ordre de quelques milliers). Construire vos propres exemples avec d'autres images et commentez les résultats.*

Références

[1] P.Perez, M. Gangnet et A. Blake, Poisson Image Editing, *ACM Transactions on Graphics*, 2003.