Vision robotique

Travaux pratiques séance 1

Yuhan DOU

I. Objectifs

La problématique abordée dans ce TP est celui du redimensionnement d'image à l'aide de la technique du "Seam Carving". Cette technique prend en considération du contenu de l'image. Elle permet de réduire la taille de l'image en préservant au mieux la taille initiale des objets.

L'objectif de ce TP est d'écrire un programme utilisant cette technique.

II. "Seam carving" basique

L'algorithme de "Seam carving" consiste à supprimer itérativement des chemins de pixels d'énergie minimum jusqu'à atteindre la taille de l'image cible. Le processus entier est divisé en les 3 étapes suivantes :

- Calcul de l'énergie d'image
- Construction du graph correspondant à l'image
- "S-T Graph-Cut" selon "maxflow" (min cut)

1/ Calcul de l'énergie d'image

En général, la norme L^2 ou L^1 du gradient de l'image est un choix raisonnable pour représenter l'énergie d'image. Selon l'article [RSA08], on choisit la norme L^1 comme ci-dessous :

$$E_1(i,j) = \partial x(i,j) + \partial y(i,j)$$

$$\partial x(i,j) = \left| I(i,j+1) - I(i,j) \right|$$

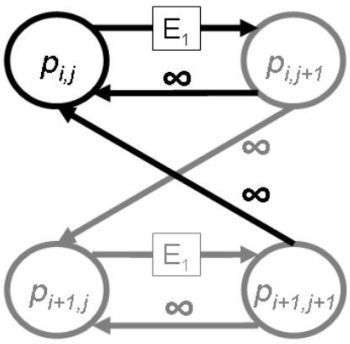
$$\partial y(i,j) = \left| I(i+1,j) - I(i,j) \right|$$

Selon les 3 formulations, on peut écrire la fonction E1 pour calculer l'énergie d'image.

```
function f = E1(img)
    lin=size(img,1);
    col=size(img,2);

    img_X=[img(:,2:col),zeros([lin,1])]; %deplacement d'image a gauche pour calculer le gradient horizontal,
                                         %et remplissage la colonne la plus droite à 0
    X=img_X-img;  %gradient horizontal

    img_Y=[img(2:lin,:);zeros([1,col])]; %deplacement d'image vers haut pour calculer le gradient vertical,
                                         %et remplissage la colonne la plus basse à 0
    Y=img_Y-img;  %gradient vertical

    f=abs(X)+abs(Y); %gradient total, qui est E1 dans l'article
    f=double(f);
end
```

2/ Construction du graph correspondant à l'image

Pour construire un graph, il faut ajouter 2 noeuds : S(source) et T(destination). Il y aura un côté à partir du noeud S vers chaque pixel dans la première colonne d'image, et il y aura un côté à partir de chaque pixel dans la dernière colonne d'image vers le noeud T. Les poids de tous ces côtés sont infini. En plus, les côté entre les pixels d'image sont construits selon la figure 4(c) dans l'article [RSA08] :



On peut utiliser la fonction "digraph" dans Matlab. Au début, on a écrit une fonction pour calculer le numéro de pixel dans l'image, afin de simplifier la construction du vecteur de tous les amonts et de tous les avals:

```
function num = num_node_in_graph(img,i,j)
        lin=size(img,1);  %nb de ligne d'image

        num=(j-1)*lin+i+2;%l'index du pixel img(i,j) dans le vecteur S ("source" pour graph)
                          %Ici, on fixe le noeud S et T à l'index 1 et 2 respectivement,
                          %donc l'index de pixel d'image commence a 3
end
```

Maintenant, on commence la construction du graph peu à peu avec des côtés à partir du noeud S (avec le numéro "1") vers la première colonne d'image :

```
E=E1(im_gray);
S(1:lin)=1; %les cotes a partir du noeud S vers la premiere colonne d'image,
            %donc les noeuds "source" tous sont S, son numéro est 1
T(1:lin)=num_node_in_graph(im_gray,1,1):num_node_in_graph(im_gray,lin,1);
            %les noeuds "destination" de ces cotes, ici ce sont la premiere colonne d'image
Weight=inf*ones([1,lin]);  %les couts (énergie) correspondants aux cotes ci-dessus
```

Ensuite, on traite sur les pixel de la première colonne d'image. Leur côtés sont un peu différents que ceux des pixels " au milieu" (de la deuxième colonne à l'avant-dernière

colonne).

```
%  ajouter la premiere colonne d'image dans le graph
S=[S,num_node_in_graph(im_gray,1,1):num_node_in_graph(im_gray,lin,1)];
T=[T,num_node_in_graph(im_gray,1,2):num_node_in_graph(im_gray,lin,2)];
Weight=[Weight,E(:,1)'];
```

Après, on prend en considération les pixels “au milieu” (de la deuxième colonne à l’avant-dernière colonne). Pour chaque colonne, il faut généralement construire 4 types de côté : ① “vers droite” [p(i,j) → p(i,j+1)], ② “vers gauche” [p(i,j) → p(i,j-1)], ③ “vers gauche-bas” [p(i,j)→p(i+1,j-1)], ④“vers gauche-haut” [p(i,j)→p(i-1,j-1)] :

```
%  les colonnes generales
for j=2:(col-1)

    %%vers droite
    %l'operation sur toute les lignnes a une fois (a l'aide de l'extension de matrice)
    S=[S,num_node_in_graph(im_gray,1,j):num_node_in_graph(im_gray,lin,j)];
    T=[T,num_node_in_graph(im_gray,1,j+1):num_node_in_graph(im_gray,lin,j+1)];
    Weight=[Weight,E(1:lin,j)'];

    %%vers gouche
    S=[S,num_node_in_graph(im_gray,1,j):num_node_in_graph(im_gray,lin,j)];
    T=[T,num_node_in_graph(im_gray,1,j-1):num_node_in_graph(im_gray,lin,j-1)];
    Weight=[Weight,inf*ones([1,lin])];

    %%vers gouche-bas
    S=[S,num_node_in_graph(im_gray,1,j):num_node_in_graph(im_gray,lin-1,j)];
    T=[T,num_node_in_graph(im_gray,2,j-1):num_node_in_graph(im_gray,lin,j-1)];
    Weight=[Weight,inf*ones([1,lin-1])];

    %%vers gouche-haut
    S=[S,num_node_in_graph(im_gray,2,j):num_node_in_graph(im_gray,lin,j)];
    T=[T,num_node_in_graph(im_gray,1,j-1):num_node_in_graph(im_gray,lin-1,j-1)];
    Weight=[Weight,inf*ones([1,lin-1])];
end
```

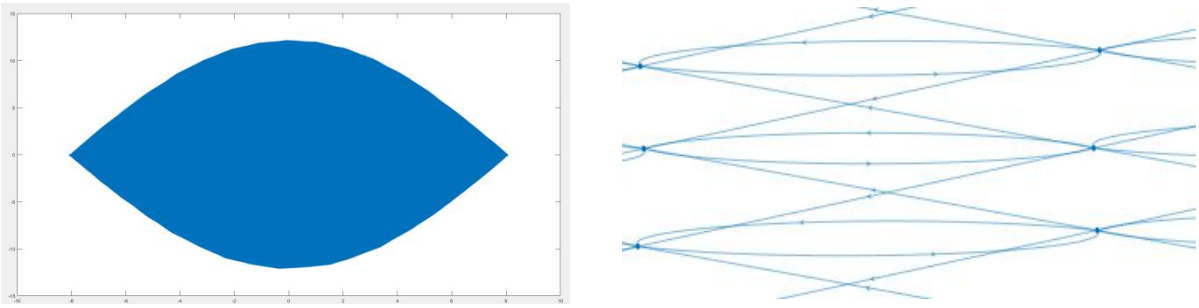
En fin, pour la dernière colonne d’image, les côtés des cas ②③④ sont les même comme le traitement sur les colonnes “au milieu”, mais il faut modifier la manière de construire les côtés “vers droite” (cas ①). Les avals dans ce cas tous sont le noeud T (avec le numéro “2”). Donc le code correspondant est représenté comme ci-dessous :

```
%%vers T (vers droite)
S=[S,num_node_in_graph(im_gray,1,j):num_node_in_graph(im_gray,lin,j)];
T=[T,2*ones([1,lin])]; %Les "destination" tous sont le noeud T (son numero est 2)
Weight=[Weight,inf*ones([1,lin])];
```

Jusqu'à maintenant, on a fini la construction des paramètres nécessaires pour l’utilisation de la fonction “digraph” dans Matlab, à savoir, le vecteur de numéro des amonts “S”, le vecteur de numéro des avals “T”, et les poids de chque côté correspondant. Donc, on peut utiliser cette fonction pour obtenir le graph final :

```
f=digraph(S,T,Weight); %graph final
```

Résultat de graph d'image "banc.jpg" :



graph d'image 'banc.jpg'

La figure à droite est un agrandissement partiel de la figure à gauche. On peut voir que les côtés sont les mêmes que le modèle vu dans l'article [RSA08].

3/ "S-T" Cut selon maxflow (min cut)

À partir du graph, on utilise la fonction "maxflow" dans Matlab pour exécuter le "S-T" Cut.

```
[m_f,v_f,S_f,T_f]=maxflow(G,1,2,'searchtrees');
```

On peut obtenir le vecteur S_f, dont les éléments sont tous les pixels associés à la partie avec le noeud "S". Ensuite, pour réduire la taille d'image, il faut supprimer sur chaque lignne quelques pixels. On a écrit une fonction "**cut=draw_cut(img, S_f, pixel_cut)**" pour touver les colonnes où il y a les pixels à supprimer lors de chaque itération.

"**img**" : l'image déjà transformée en gris;

"**S_f**" : le vecteur dont les éléments sont tous les pixels associés à la partie avec le noeud "S" (on dite ces pixels sont de type "S");

"**pixel_cut**" : le nombre de pixel dans chaque ligne à supprimer lors de chaque iteration. C'est aussi la lageur du chemin à supprimer.

"**cut**" : une matrice de même dimension d'image. Les premieres "pixel_cut" colonnes contiennent les pixels associés à la partie "T" (on dite ces pixels sont de type "T").

Les pixels de type"S" sont considés comme l'objet, et ceux de type "T" sont considés comme le fond. Donc pour réduire la taille d'image, il faut supprimer les pixels de type "T".

L'initialisation de la matrice "cut" est exécutée comme ci-dissous :

```
cut=zeros([size(img,1),size(img,2)]); %meme dimension d'image

%initialision
for i=1:size(S_f,1)
    if (S_f(i)>2)          %l'index de pixel d'image commence a 3
        cut(S_f(i)-2)=255;%au debut, on met tous les pixel d'image associes au noeud S a blanc(255),
                          %donc les pixels d'image associe au noeud T reste a noir(0)
    end
end
```

C'est-à-dire que tous les pixels dans la partie avec "S" sont mis à 255 dans la matrice "cut", et les pixels dans la partie avec "T" restent à 0.

Ensuite, on va chercher toutes les colonnes où il y a au moins un pixel de type “T”. Le code correspondant est comme ci-dessous :

```
test=sum(cut,1); %le vecteur dont l'element est la somme de chaque colonne de "cut"

p=find(test<255*size(img,1));
                    %chercher tous les colonnes ou il y a au moins un pixel associe au noeud T
                    % (Ces pixels associes au T doit etre supprimer)
```

Maintenant, la valeur dans la matrice “cut” ne peut prendre que 2 valeurs : 0 ou 255. Si la somme d’une colonne est égale à (255 * nombre de ligne), c’est-à-dire que tous les pixels dans cette colonne sont de type “S”. De même, si la somme est inférieure que (255 * nombre de ligne), il y aura au moins un pixel de type “T”. “p” est la vecteur dont chaque élément est l’index de chaque colonne qui contient les pixels de type “T”.

Pour trouver le chemin du “cut”, il faut trouver la première colonne qui contient les pixels de type “T”. Parce que maintenant dans le graph d’image, tous les pixels à gauche du chemin sont de type “S”, et ceux à droite du chemin sont de type “T”. Si on trouve la première colonne qui contient les pixels de type “T”, on suppose que on va commence à supprimer quelques colonnes à gauche de cette colonne.

En fait, le premier élément du vecteur “p” p(1) est l’index de la colonne qu’on cherche. On prend en compte deux cas suivants.

① Si p(1)>pixel_cut, c’est-à-dire que il y aura assez de colonnes à gauche pour les supprimer. Pour illustrer le chemin, on va mettre tous les pixels sur ce chemin à 255, mais les autres à 0. Au début, on déplace le "cut" original les "pixel_cut" colonnes à gauche, et remplit les dernieres colonnes avec 0. Ensuite, on soustrait le “cut” original de la matrice obtenue. On va obtenir le résultat souhaité. Le code correspondant est représenté ci-dessous :

```
if (p(1)>pixel_cut) %p(1) est l'index du premier element de "p",
                    %c'est l'index de la premiere colonne qui contient les pixels associé au noeud "T"
                    %On va commence le cut a cette colonne.

    cut_p=[cut(:,(1+pixel_cut):size(img,2)),zeros([size(img,1),pixel_cut])];
    %deplacer le "cut" original les "pixel_cut" colonnes a gauche, et remplir les dernieres colonnes avec 0
    cut=cut-cut_p; %apres la soustraction, seulement les pixels dans le chemin ont la valeur 255,
                   %autre pixels sont a 0
```

②Si p(1)≤pixel_cut, il n'y aura pas assez de colonnes a gquche pour les supprimer, donc on va deplacer le "cut" original à droite.

```
%Si p(1)<=pixel_cut, il n'y aura pas assez de colonnes a gquche pour les supprimer,
%donc, on va deplacer le "cut" original a droite
else
    cut_p=[255*ones([size(img,1),pixel_cut]),cut(:,1:size(img,2)-pixel_cut)];
    %deplacer le "cut" original les "pixel_cut" colonnes a droite, et remplir les premieres colonnes avec 0
    cut=cut_p-cut;
end
```

En fin, pour la matrice “cut”, seulement les pixels dans le chemin à supprimer sont mets à 255, les autres pixels sont mets à 0.

Donc dans chaque itération, on cherche sur chaque ligne les pixels qui ont la valeur 255 (c’est-à-dire que les pixels de type “T”, à supprimer). Après la boucle sur ligne, on va obtenir

le chemin complet supprimé (cut_lin).

```
%pour chaque ligne, on cherche les pixels associes au noeud "T" (a supprimer)
for i=1:size(im_gray,1)
    cut_lin=find(cut(i,:)==255);
    im2(i,:,:)=[im(i,1:(cut_lin(1)-1),:),im(i,(cut_lin(cut_step)+1):size(im_gray,2),:)];%suppression
end
%après la boucle interne, cut_lin est le chemin à supprimer
```

Résultat de "S-T" cut :

1. banc.jpg : 50 itération, pas = 1 pixel

image originale

image réduite

On peut voir que la lageur d'image est réduite, quelques colonnes du font sont supprimer. Mais quelques parties d'objet sont aussi réduites, c'est pas idéale. Suivantes sont les figures avec le chemin :

On peut voir dans la figure à droite que le chemin trouvé traverse l'objet, c'est pourquoi dans l'image réduite il y a des dentelures sur le bord droit du banc.

2. plage.jpg : 50 itération, pas = 1 pixel

image originale

image réduite

4/ Comparaison

Comme on peut voir dans les images traitées, quelques parties d'objet sont aussi supprimées. Ce n'est pas ce que l'on souhaite. Maintenant on essaie d'utiliser la fonction "gradient" dans Matlab au lieu de la méthode écrite dans l'article [RSA08]. (Cette méthode correspond à la fonction "E1" dans notre code.)

Le résultat de "banc.jpg" est comme ci-dessous :

image originale

image réduite

On peut voir que leffet de la réduction d'image est un peut amélioré qu'avant, mais les pieds de la chaise sont toujours perdus. Donc pour obtenir un meilleur résultat, il faut faire de pré-taitement.

III. Discussion

Pendant ce TP, on a appris l'algorithme "Seam Carving", et on a compris le graph cut. On a connâit les conceptions d'énergie d'image, du graph et du "max-flow".

On a compris l'application du "graph cut" dans le traitement d'images pratique. Les pixels de type "S" sont considés comme l'objet, et les pixels de type "T" sont considés comme le fond. Donc pour réduire la taille d'image, il faut supprimer les pixels de type "T".

En plus, on peut trouver les avantages et les inconvénients de cet algorithme. Il peut parfois trouver le chemin traversant l'objet dans l'image. Il faut alors effectuer les prétraitements correspondants pour éviter ses défauts. Par exemple, on peur tout d'abord faire la segmentation, ensuite effextuer le graph cut dans la classe d'objet.