Rectification d'image par homographie Application a la construction d'une mosaïque d'images

Abied Imad *Bigi Mohamed*

17 janvier 2022

Table des matières

1 Introduction

Le but du projet et de produire une mosaïque à partir d'une séquence d'images, et ceci en reliant les images deux par deux en utilisant la transformée géométrique *homographie*. Le but initiale sera d'abord de mettre en place une homographie correcte, capable de remplacer un quadrangle d'une image par n'importe quel autre quadrangle. Après, la prochaîne étape sera de créer une mosaîque, en commençant par une mosaîque de deux images, et terminant par une mosaîque de n images.

2 Mise en place de l'homographie

2.1 Recherche des coefficients de l'homographie

Analytiquement, on retrouve les coefficients de la matrice d'homographie en développant la relation

$$\begin{cases} x_2 = \dfrac{h_{11}x_1 + h_{12}y_1 + h_{13}}{h_{31}x_1 + h_{32}y_1 + h_{33}} \\ y_2 = \dfrac{h_{21}x_1 + h_{22}y_1 + h_{23}}{h_{31}x_1 + h_{32}y_1 + h_{33}} \end{cases} \tag{1}$$

et ceci pour les quatres points appariés des deux quadrangle. On obtient ainsi l'équation suivante

$$AX = B \tag{2}$$

avec X vecteur contenant les coefficients de la matrice d'homographie, on calcule ainsi les coefficients en calculant le produit matricielle

$$X = A^{-1}B \tag{3}$$

si A est inversible, et

$$X = (A^\intercal A)^{-1}A^\intercal B \tag{4}$$

Le calcul de la matrice en matlab est basé sur le calcul analytique précédant, et ceci en initialisant les matrices A et B avec les coordonnées prise en paramètre, et en calculant l'inverse.

2.2 Application de l'homographie

L'application de l'homographie est réalisé grâce aux deux fonctions ***getpx*** et ***homographic_function***. La première fait appel à ***homographic_matrix*** afin de calculer la matrice d'homographie correspondante, et ceci en passant en paramètre, quatre point du quadrangle source et les quatres points qui délimitent le quadrangle cible. Puis, elle fait appel à la fonction ***getpx***, qui applique l'homographie sur les coordonnées de chaque point de l'image d'origine, et ceci en appliquant la formule 1. La fonction ***getpx*** permet aussi de remplacer le quadrangle de l'image d'origine par le quadrangle cible, et ceci en comparant les coordonnées obtenues par la transformée avec les bordures du repère définit par l'image cible. Si les coordonnées sont incluses dans le repère, on remplace les pixels de l'image d'origine, définit par les coordonnées originaux, par les pixels de l'image cible, définit par les nouveaux coordonnées, résultats de la transformation, sinon, on garde les pixels de l'image d'origine. Les calculs sont réalisé après conversion des valeurs des pixels en *double* pour plus de précision, l'affichage est en *uint8*.

Les figures 1a, 1b,2a,2b montrent un exemple des points d'un quadrangle d'origine, La figure 3a montre le quadrangle cible, et finalement, la figure 6a, montre le résultat final de l'homographie.

(a) Premier point

(b) Deuxième point

(a) Troisième point

(b) Quatrième point

(a) Quadrangle cible

(a) Résultat final

3 Mise en place de la mosaïque

3.1 Etape1 : Division de l'image en trois composantes

Pour mettre en place une mosaîque, la première étape sera de diviser chaque image en trois composante. La première est l'image elle-même, la deuxième composante est le mask binaire, la troisième composante est la boîte englobante. La boîte englobante de l'image comme le cadre dans lequel se trouve l'image, c'est à dire, si notre image a pour longueur w et largeur h, notre boîte sera définie par les deux points de coordonnées *(0,w-1)* et *(0,h-1)*. Ces trois composantes sont mise en place dans la fonction ***image_to_mosaîque***

3.2 Etape2 : Transformation des trois composantes avec l'homographie

La deuxième étape consiste en l'application de l'homographie sur les trois composantes mentionnées précédemment. D'abord, on doit déterminer la matrice d'homographie à utiliser. Pour se faire, on

prend deux ensembles de quatres points communs entre les deux images qu'on va utiliser pour réaliser la mosaîque.

- **Transformation de la boîte englobante** : La matrice d'homographie obtenue, on l'appliquera sur chaque boîte englobante, *(ie, les coordonnées des quatres sommets)*, après, on prendera le minimum et le maximum de chaque coordonée, construisant ainsi une nouvelle boîte, qui englobe l'original.
- **Transforation du masque et de l'image** : Ces deux composantes sont d'abord initialisées par une matrice nulle, de même taille de celle de la boîte englobnate, la prochaine étape serait donc de positionner l'image au sein de la boîte, pour se faire, on applique la matrice d'homographie obtenue sur l'image initiale, on obtient ainsi notre image transformée, et on détermine après notre masque binaire

Ces transformations sont mise en place par la fonction ***transform_ mosaique***

3.3 Etape3 : Fusion des images pour obtenir la mosaîque

Les étapes décrites précédemment seront appliquées sur chaque image qu'on veut créer une mosaîque avec, ainsi, la dernière étape sera logiquement la fusion de tous ces images.

- **Fusion des boîtes englobantes** : La fusion des boîtes englobante est simple, on prend le minimum de chaque coordonées sur le premier sommet *(ie en haut à gauche)*, et le maximum de chaque coordonnées sur le dernier sommet *(ie en bas à droite)*, notre nouvelle boîte est celle délimitée par les nouveaux coordonnées.
- **Fusion des images et masques** : La fusion des deux images revient à déterminer les positions des pixels des deux images dans la nouvelle boîte englobante. Pour y positionner les deux images correctement, on doit effectuer un changement de repère. Pour chaque image, le repère d'origine étant sa boîte englobante avant la fusion, on doit donc déterminer un vecteur de changement de repère, avec le nouveau repère est la nouvelle boîte englobante après fusion. Pour la première image, après avoir déterminer le vecteur de mouvement, on remplace les pixels *(noirs à priori)* de la boîte englobante par celles de l'image. Pour la deuxième image, après avoir déterminer le vecteur de mouvement, on somme les valeurs des pixels de l'image avec celles qui correspondent à leurs positions dans la nouvelle boîte englobante, car quelques positions *(ie les parties en communs entre les deux images)* sont déjâ popularisées par la première image. Le même traitement est réalisé pour les masques binaires. Une dernière est de normaliser les valeurs des pixels de la mosaîque, chaque valeur de la nouvelle image est ainsi divisée par le coefficient *(new_ mosaique.masque + (new_ mosaique.masque == 0))*, ceci permet d'avoir trois cas
  - **Position popularisée par aucune image** : La valeur du coefficient est égale à 1, on garde ainsi la valeur originale.
  - **Position popularisée par une unique image** : La valeur du coefficient est encore égale à 1, on garde ainsi la valeur originale.
  - **Position popularisée par deux images** : Dans ce cas, la valeur du coefficient est égale à 2, la valeur du pixel dans ce cas étant la somme des valeurs dans les images originales, diviser par deux permet de normaliser correctement les intensités.

Cette étape de fusion est mise en place par la fonction ***merge_ two_ mosaique***

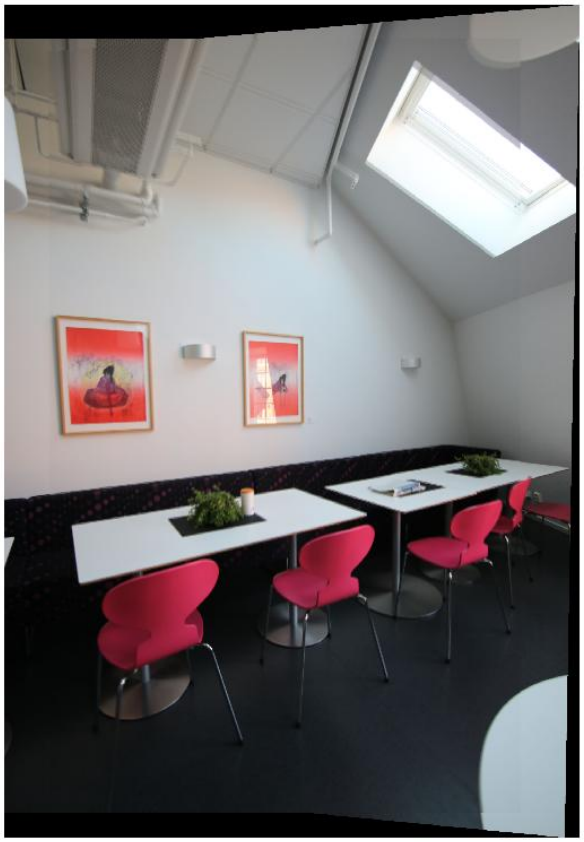
4 Exemples de mosaique d'images

Pour traiter les images en couleur, on a ajouter une troisième dimension pour la composante ***image*** dans nos fonctions.

4.1 Mosaîque de deux images

(a) Première image

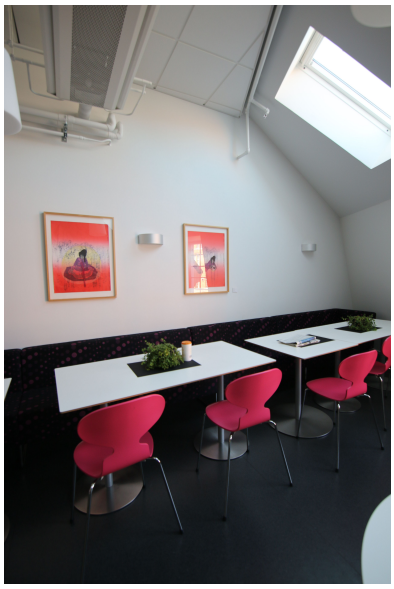
(b) Deuxième image



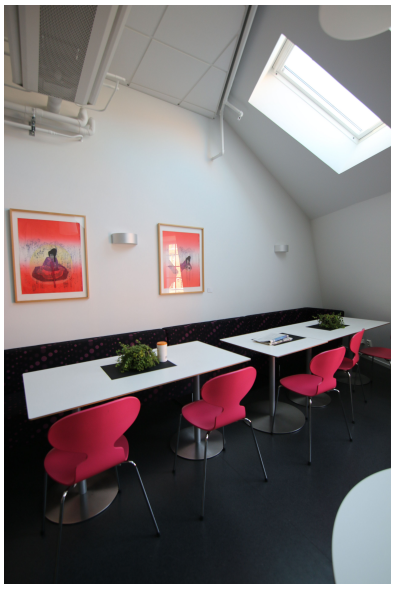
(a) Résultat final

FIGURE 6 – Mosaîque avec 2 images

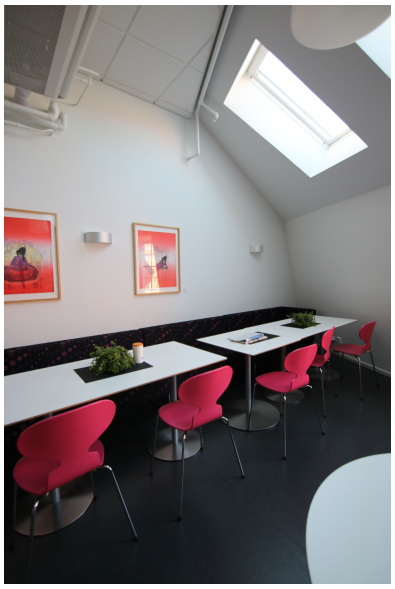
4.2 Mosaîque de trois images



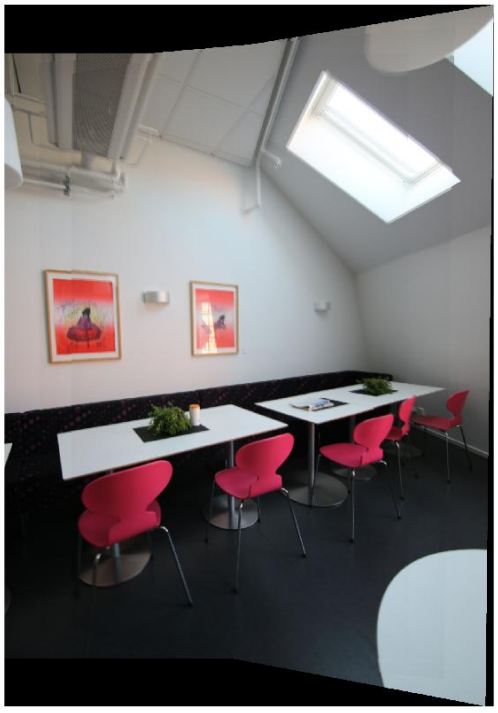
(a) Première image



(b) Deuxième image



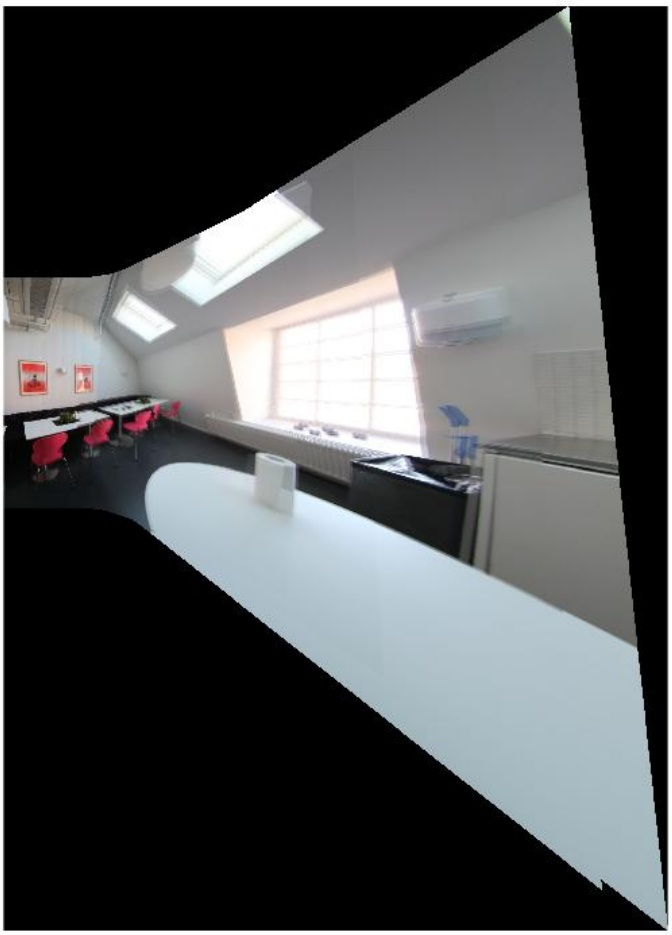
(c) Troisième image



(a) Résultat final

FIGURE 8 – Mosaîque avec 3 images

4.3 Mosaîque avec 10 images



(a) Résultat final

FIGURE 9 – Mosaîque avec 10 images

4.4 Principe de mosaîque de n images

Le principe de la mosaîque à n images se base sur la création d'une mosaîque par deux images. En effet, on commance par la création d'une première mosaîque avec deux image. Une fois la mosaîque crée, on recommance la procédure, avec la mosaîque crée comme image d'origine, et la procchaîne image comme l'image cible.

— Le script qui lance la mosaîque de deux images est : *mosaique_2_image_test.m*
— Le script qui lance la mosaîque de n images est : *mosaique_n_image_test.m*

5 Conclusion

L'homographie est une transformation essentielle dans le monde du traitement d'image, puisqu'elle permet la projection géométrique des images, d'un espace à l'autre. Elle possède donc plusieurs appli-

cations, tel que le remplacement d'un quadrangle d'une image par un autre appartenant à une autre, ou bien la création d'une mosaïque, à partir de différentes images.

6 Annexe : Code Matlab

6.1 homography_test

```
clear all;
close all;
clc;

origine_path = "our_images/twin_centre.jpg";
target_path = "our_images/github_logo.jpg";

origine = imread(origine_path);
target = imread(target_path);

figure,
imshow(origine);
pts_o = ginput(4);
title(origine);

output = homographic_function(origine, target,pts_o);

figure,
imshow(output);
title("output");
```

6.2 homographic_matrix

```
function H = homographic_matrix(pts_o, pts_t)

    %
    % pts_o : [x1, y1; x2, y2; x3, y3; x3, y3]
    % pts_t : [x1, y1; x2, y2; x3, y3; x3, y3]
    %

    A = zeros(8, 8);
    B = zeros(8, 1);

    for pt_index=1:1:4
        A(2*(pt_index-1)+1, :) = [-pts_o(pt_index,1), -pts_o(pt_index
            ,2), -1, 0, 0, 0, pts_o(pt_index,1)*pts_t(pt_index,1),
            pts_t(pt_index,1)*pts_o(pt_index,2)];
        A(2*pt_index, :) = [0, 0, 0, -pts_o(pt_index,1), -pts_o(
            pt_index,2), -1, pts_o(pt_index,1)*pts_t(pt_index,2),
            pts_o(pt_index,2)*pts_t(pt_index,2)];

        B(2*(pt_index-1)+1, 1) = - pts_t(pt_index, 1);
        B(2*pt_index, 1) = - pts_t(pt_index, 2);
    end

    h_vect = A \ B;



    h_vect = [h_vect; 1];
    H = reshape(h_vect, 3, 3)';

end
```

6.3 homographic_function

```
function  output = homographic_function(origine,panel,pts_o)

[h_origine,w_origine,z_origine] = size(origine);
[h_panel,w_panel,~] = size(panel);

output = zeros(h_origine,w_origine,z_origine);

pts_t = [...
    1          , 1; ...
    w_panel    , 1; ...
    w_panel    , h_panel; ...
    1          , h_panel...
];

H = homographic_matrix(pts_o,pts_t);

output = double(output);
origine = double(origine);
panel = double(panel);

for i = 1:h_origine
    for j = 1:w_origine
        output(i,j,:) = getpx(H, [j, i], origine, panel);
    end
end

output = uint8(output);
end
```

6.4 getpx

```
function px = getpx(H,pt,origine,panel)

[h_panel,w_panel,~] = size(panel);


coor_homogene_origine = [pt(1);pt(2);1];
coor_homogene_panel = round([(H(1,1)*pt(1)+H(1,2)*pt(2)+H(1,3))/(H
    (3,1)*pt(1)+H(3,2)*pt(2)+H(3,3)),(H(2,1)*pt(1)+H(2,2)*pt(2)+H(2,3)
    )/(H(3,1)*pt(1)+H(3,2)*pt(2)+H(3,3))]);


isinside_panel = (...
    (coor_homogene_panel(1) > 0 && w_panel >= coor_homogene_panel(1))
        && ...
    (coor_homogene_panel(2) >0 && h_panel >= coor_homogene_panel(2))
        ...
);


if( isinside_panel)
      px = panel(coor_homogene_panel(2), coor_homogene_panel(1),:);
else
     px = origine(coor_homogene_origine(2), coor_homogene_origine(1)
         ,:);
end

end
```

6.5 homographic_get_pt

```
function new_pt = homographic_get_pt(H, pt)
    pt = double(pt);
    new_pt = round([(H(1,1)*pt(1)+H(1,2)*pt(2)+H(1,3))/(H(3,1)*pt(1)+
        H(3,2)*pt(2)+H(3,3)),(H(2,1)*pt(1)+H(2,2)*pt(2)+H(2,3))/(H
        (3,1)*pt(1)+H(3,2)*pt(2)+H(3,3))]);

end
```

6.6 mosaique_2_image_test

```
clear all;
close all;
clc;

%origine_path = "our_images/petites_imgs/img01.jpg.jpeg";
%target_path = "our_images/petites_imgs/img02.jpg.jpeg";

origine_path = "our_images/img01.jpg";
target_path = "our_images/img02.jpg";

origine = imread(origine_path);
target = imread(target_path);

origine = rgb2gray(origine);
target = rgb2gray(target);

% figure(1),
% subplot(1,2,1);
% imshow(origine);
% subplot(1,2,2);
% imshow(target);
% title("test");

figure,
imshow(origine);
pts_o = ginput(4);
title(origine_path);

figure,
imshow(target);
pts_t = ginput(4);
title(target_path);

H_for_boite = homographic_matrix(pts_t, pts_o);
H_for_browse = homographic_matrix(pts_o, pts_t);


mosaique_o = image_to_mosaique(origine);
mosaique_t = image_to_mosaique(target);

mosaique_transform  = transform_mosaique(mosaique_t, H_for_boite,
    H_for_browse);

figure,
imshow(uint8(mosaique_transform.image));
```

```
title("output_tr");

new_mosaique = merge_two_mosaique(mosaique_o ,mosaique_transform);

figure,
imshow(uint8(new_mosaique.image));
title("output");
```

6.7 mosaique_n_image_test

```
clear all;
close all;
clc;

% images_path = [ ...
%     "our_images/petites_imgs/img01.jpg.jpeg",    ...
%     "our_images/petites_imgs/img02.jpg.jpeg",    ...
%     "our_images/petites_imgs/img03.jpg.jpeg"     ...
% ];

images_path = [ ...
    "our_images/img01.jpg",    ...
    "our_images/img02.jpg",    ...
    "our_images/img03.jpg"     ...
    "our_images/img04.jpg"     ...
    "our_images/img05.jpg"     ...
    "our_images/img06.jpg"     ...
    "our_images/img07.jpg"     ...
    "our_images/img08.jpg"     ...
    "our_images/img09.jpg"     ...
    "our_images/img10.jpg"     ...
];

images_nbr = length(images_path);

%image = rgb2gray(imread(images_path(1)));
image = imread(images_path(1));
assembled_mosaique = image_to_mosaique(image);

mosaique_figure = figure;
mosaique_figure.Position(1:2) = [200  400];

ginput_figure = figure;
ginput_figure.Position(1:2) = [1000  400];

```

```
figure(mosaique_figure),
imshow(uint8(assembled_mosaique.image));
title('Mosaique');

for i=2:1:images_nbr

    %image = rgb2gray(imread(images_path(i)));
    image = imread(images_path(i));
    tmp_mosaique = image_to_mosaique(image);
    figure(ginput_figure),
    imshow(uint8(tmp_mosaique.image));
    title('image to insert : Select 4 points !!');
    pts_t = ginput(4);
    title('image to insert');

    figure(mosaique_figure),
    title('Mosique : Select 4 points !!');
    pts_o = ginput(4);
    pts_o(:,1) = pts_o(:,1) + assembled_mosaique.boite(1, 1);
    pts_o(:,2) = pts_o(:,2) + assembled_mosaique.boite(1, 2);
    title('Mosique');

    H_for_boite = homographic_matrix(pts_t, pts_o);
    H_for_browse = homographic_matrix(pts_o, pts_t);

    tmp_mosaique  = transform_mosaique(tmp_mosaique, H_for_boite,
        H_for_browse);

    assembled_mosaique = merge_two_mosaique(assembled_mosaique,
        tmp_mosaique);

    figure(mosaique_figure), imshow(uint8(assembled_mosaique.image));

    fprintf("%d,%d %d,%d\n", assembled_mosaique.boite(1,1),
        assembled_mosaique.boite(1,2), assembled_mosaique.boite(2,1),
        assembled_mosaique.boite(2,2));

end
```

6.8 image_to_mosaique

```
function mosaique = image_to_mosaique(image)

    [h, w, c] = size(image);

```

```
    if c ~= 3
        mosaique.image = double(image);
        mosaique.image(:, :, 2) = double(image);
        mosaique.image(:, :, 3) = double(image);
    else
        mosaique.image = double(image);
    end


    mosaique.masque = ones(h, w);
    mosaique.boite = [0, 0; w-1, h-1]; % Coordon es g ographique

end
```

6.9 transform_mosaique

```
function new_mosaique = transform_mosaique(mosaique, H_for_boite,
    H_for_browse)

    [h, w, c]= size(mosaique.image);

    if c ~= 3
        error("transform_mosaique : handle only images with 3 channel
            color.");
    end

    %% get boit

    corners = zeros(4, 2);
    corners(1,:) = homographic_get_pt(H_for_boite, [1, 1]);
    corners(2,:) = homographic_get_pt(H_for_boite, [w, 1]);
    corners(3,:) = homographic_get_pt(H_for_boite, [w, h]);
    corners(4,:) = homographic_get_pt(H_for_boite, [1, h]);

    x_min = min(corners(:, 1));
    x_max = max(corners(:, 1));
    y_min = min(corners(:, 2));
    y_max = max(corners(:, 2));

    new_mosaique.boite = [x_min-1, y_min-1; x_max-1, y_max-1];

    %% get masque & image

    new_h = y_max-y_min+1;
    new_w = x_max-x_min+1;
```

```
28
29      new_mosaique.masque = zeros(new_h, new_w);
30      new_mosaique.image = zeros(new_h, new_w, c);
31
32      for i=1:new_h
33          for j=1:new_w
34              new_pt = homographic_get_pt(H_for_browse, [j+new_mosaique
                    .boite(1, 1), i+new_mosaique.boite(1,2)]);
35              if 1 <= new_pt(1) && new_pt(1) <= w && 1 <= new_pt(2) &&
                    new_pt(2) <= h
36                  new_mosaique.image(i, j, :) = mosaique.image(new_pt
                        (2), new_pt(1), :);
37                  new_mosaique.masque(i, j) = 1;
38              else
39                  new_mosaique.image(i, j, :) = 0 * mosaique.image(1,
                        1, :); % pour garder le format d'image (rgb ou
                        niveau de gris)
40                  new_mosaique.masque(i, j) = 0;
41              end
42          end
43      end
44
45  end
```

6.10 merge_two_mosaique

```
1       function new_mosaique = merge_two_mosaique(mosaique1, mosaique2)
2
3       %% get boite
4       new_mosaique.boite = [ ...
5           min(mosaique1.boite(1, 1), mosaique2.boite(1, 1)), min(
                mosaique1.boite(1, 2), mosaique2.boite(1, 2)); ...
6           max(mosaique1.boite(2, 1), mosaique2.boite(2, 1)), max(
                mosaique1.boite(2, 2), mosaique2.boite(2, 2))
7       ];
8
9       %% init masque & image
10      new_h = new_mosaique.boite(2, 2) - new_mosaique.boite(1, 2) + 1;
11      new_w = new_mosaique.boite(2, 1) - new_mosaique.boite(1, 1) + 1;
12
13      new_mosaique.masque = zeros(new_h, new_w);
14      new_mosaique.image = zeros(new_h, new_w, 3);
15
16      %% Populate masque & image
17
```

```
18      % coordonn es matercielles
19
20      x_m1_min = mosaique1.boite(1,1) - new_mosaique.boite(1,1) + 1;
21      y_m1_min = mosaique1.boite(1,2) - new_mosaique.boite(1,2) + 1;
22      x_m1_max = mosaique1.boite(2,1) - new_mosaique.boite(1,1) + 1;
23      y_m1_max = mosaique1.boite(2,2) - new_mosaique.boite(1,2) + 1;
24
25      x_m2_min = mosaique2.boite(1,1) - new_mosaique.boite(1,1) + 1;
26      y_m2_min = mosaique2.boite(1,2) - new_mosaique.boite(1,2) + 1;
27      x_m2_max = mosaique2.boite(2,1) - new_mosaique.boite(1,1) + 1;
28      y_m2_max = mosaique2.boite(2,2) - new_mosaique.boite(1,2) + 1;
29
30      % populate with mosaique1
31      new_mosaique.masque(y_m1_min:y_m1_max, x_m1_min:x_m1_max) =
            mosaique1.masque;
32
33      new_mosaique.image(y_m1_min:y_m1_max, x_m1_min:x_m1_max, :) =
            mosaique1.image;
34
35      % populate with mosaique2
36      new_mosaique.masque(y_m2_min:y_m2_max, x_m2_min:x_m2_max) = ...
37      new_mosaique.masque(y_m2_min:y_m2_max, x_m2_min:x_m2_max) +
            mosaique2.masque;
38
39      new_mosaique.image(y_m2_min:y_m2_max, x_m2_min:x_m2_max, :) = ...
40      new_mosaique.image(y_m2_min:y_m2_max, x_m2_min:x_m2_max, :) +
            mosaique2.image;
41
42      % normalizing
43      new_mosaique.image(:,:,1) = new_mosaique.image(:,:,1) ./ (
            new_mosaique.masque + (new_mosaique.masque == 0));
44      new_mosaique.image(:,:,2) = new_mosaique.image(:,:,2) ./ (
            new_mosaique.masque + (new_mosaique.masque == 0));
45      new_mosaique.image(:,:,3) = new_mosaique.image(:,:,3) ./ (
            new_mosaique.masque + (new_mosaique.masque == 0));
46
47      new_mosaique.masque = new_mosaique.masque ./ (new_mosaique.masque
            + (new_mosaique.masque == 0));
48
49  end
```