

Calcul de Mosaïques



Guéron Marie et Favreau Jean-Dominique

VIM / Master SSTIM

Polytech' Nice Sophia Antipolis

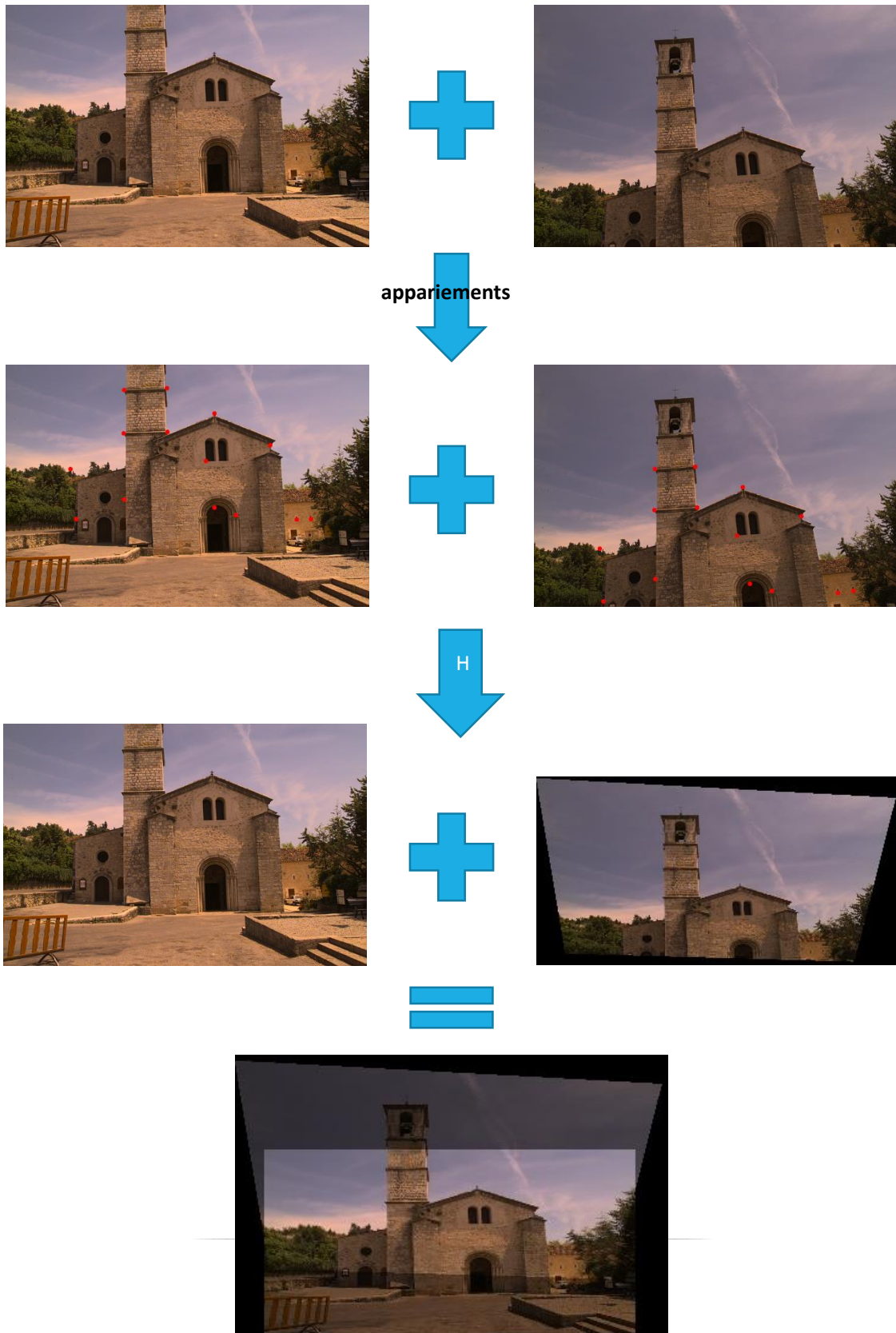
06/01/2014

Table des matières

Introduction.....	2
Calculs théorique de l'homographie	3
Passage à la pratique, avec un fichier	4
1. Lecture du fichier d'appariements.....	4
2. Calcul de la matrice d'homographie	6
3. Recollement des images	7
Passage à la pratique, sans fichier.....	9

Introduction

Le but est de calculer la déformation H qu'il a été nécessaire d'appliquer à une image pour que les points d'appariement qui ont été fournis coïncident sur les deux images. C'est cette matrice d'homographie H qui nous permettra de recoller les deux images ensemble grâce à leurs points d'appariement.



Calculs théorique de l'homographie

Considérons deux points P_1 et P_2 sur deux images différentes. On suppose que la deuxième image a subi une déformation H que nous cherchons à déterminer.

Nous posons :

$$P_1 = \begin{pmatrix} z \\ t \\ 1 \end{pmatrix}, P_2 = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \text{ et } H = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ \alpha_7 & \alpha_8 & \alpha_9 \end{pmatrix}$$

P_2 ayant subi une déformation H pour recoller à P_1 , nous avons :

$$P_1 = HP_2$$

Ce qui est équivalent à :

$$P_1 \wedge HP_2 = 0$$

Or :

$$\begin{aligned} P_1 \wedge HP_2 &= \begin{vmatrix} z & (HP_2)_1 & e_1 \\ t & (HP_2)_2 & e_2 \\ 1 & (HP_2)_3 & e_3 \end{vmatrix} \\ &= \begin{pmatrix} t(\alpha_7 x + \alpha_8 y + \alpha_9) - (\alpha_4 x + \alpha_5 y + \alpha_6) \\ (\alpha_1 x + \alpha_2 y + \alpha_3) - z(\alpha_7 x + \alpha_8 y + \alpha_9) \\ z(\alpha_4 x + \alpha_5 y + \alpha_6) - t(\alpha_1 x + \alpha_2 y + \alpha_3) \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 & 0 & -x & -y & -1 & tx & ty & t \\ x & y & 1 & 0 & 0 & 0 & -zx & -zy & -z \\ -tx & -ty & -t & zx & zy & z & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \\ \alpha_9 \end{pmatrix} \end{aligned}$$

Où $(HP_2)_i$ est la i -ème composante du vecteur HP_2 .

$$\text{Et on pose } A = \begin{pmatrix} 0 & 0 & 0 & -x & -y & -1 & tx & ty & t \\ x & y & 1 & 0 & 0 & 0 & -zx & -zy & -z \\ -tx & -ty & -t & zx & zy & z & 0 & 0 & 0 \end{pmatrix}$$

Nous avons donc maintenant à résoudre un problème qui s'écrit : $AX = 0$ où X est l'homographie que l'on cherche sous forme de colonne.

Ce qui n'est possible que si nous avons au moins 3 appariements. Et donc un jeu de neuf équations.

Passage à la pratique, avec un fichier

1. Lecture du fichier d'appariements

La liste des appariements est enregistrée dans un fichier qui a pour extension ".matches". Ce fichier a une syntaxe bien particulière qui nous permet de lui appliquer un algorithme qui extrait l'intégralité des informations qui nous sont utiles pour le traitement des images que nous aurons à faire par la suite.

```
1 #Legende
2 5
3 0 i.25.png 765x509
4 1 i.26.png 765x509
5 2 i.27.png 765x509
6 3 i.28.png 765x509
7 4 i.29.png 765x509
8 #
9 #no_image1 x y no_image2 x y ...
10 #...
11 1 183.415662629 422.47 0 687.11746988 428.485454545
12 1 216.135542147 351.672727273 0 682.048192771 353.060909091
13 1 32.489457846 346.351364155 0 497.710843549 347.739545044
14 1 7.37349395256 277.636364013 0 472.825301184 288.973181073
15 0 536.5 281.5 1 87.3298196315 274.628636755
16 0 560.463855959 210.239999233 1 123.275603302 203.137272351
17 0 634.5 168.5 1 204.844880376 175.836363144
18 0 734.963856007 169.194090762 1 286.875001443 191.106362998
19 0 733.5 271.5 1 274.662650198 280.181363078
20 0 703.938252627 318.004091687 1 246.090361714 319.744545674
21 0 476.5 159.5 1 34.5867475021 132.108636188
22 1 591.5 385.5 2 135.675903496 401.647272732
23 1 492.5 368.5 2 36.9000008299 386.839999826
24 3 47.8879511641 407.662724312 1 486.5 237.5 2 48.6975911839 239.461363259
25 3 139.145783133 384.2949971 1 558.5 210.5 2 136.601205885 220.489547731
```

Nombre d'images à traiter

Liste des images à traiter avec la syntaxe suivante :
n° de l'image / nom de l'image / largeur / hauteur

Liste des appariements

Chaque appariement est écrit de la manière suivante :
n° de l'image de référence / coordonnée en x / coordonnée en y / n° de l'image à recoller / coordonnée en x dans l'image à recoller / coordonnée en y dans l'image à recoller

Les 3 dernières étapes sont à répéter autant de fois qu'il y a d'images partageant ce point d'appariement.

```

template<typename T>
void mosaic<T>::lecture_appariement(std::string file)
{
    int nb_img;
    std::unique_ptr<Image<T>> ptr_image;
    std::fstream fichier(file.c_str());
    if(fichier)
    {
        std::string ligne;
        std::string id1,id2;

        std::getline(fichier, ligne);
        while(ligne[0]=='#')
        {
            std::getline(fichier, ligne);
            std::stringstream(ligne)>>nb_img;
            std::cout << nb_img << std::endl;
            for(int i=0; i<nb_img; i++)
            {
                std::getline(fichier, ligne);
                if(ligne[0]=='#' || ligne.size()==0)
                {
                    i--;
                    continue;
                }
                ptr_image = std::unique_ptr<Image<T>>(new Image<T>(ligne));
                std::cout << ptr_image->get_id() << std::endl;
                img[ptr_image->get_id()] = std::move(ptr_image);
            }
        }
        for(const auto& a : img)
        {
            std::cout << *a.second << std::endl;
        }

        std::vector<point_i<T>> vtmp;
        point_i<T> ptmp;
        while(getline(fichier, ligne))
        {
            if(ligne[0]=='#' || ligne.size()==0)
            {
                continue;
            }
            vtmp.clear();
            std::stringstream ss(ligne);

            while(ss.ignore())
            {
                ss >> ptmp.id >> ptmp.p.x >> ptmp.p.y;
                vtmp.push_back(ptmp);
            }
            const int size = vtmp.size();
            for(int i=0; i<size; i++)
            {
                int id = vtmp[i].id;
                for(int j=0; j<i; j++)
                {
                    img[id]->add_correspondance(vtmp[i],vtmp[j]);
                }
                for(int j=i+1; j<size; j++)
                {
                    img[id]->add_correspondance(vtmp[i],vtmp[j]);
                }
            }
        }
        for(const auto& a : img)
        {
            std::cout << *a.second << std::endl;
        }
        fichier.close();
    }
    else{
        std::cerr << "impossible d'ouvrir le fichier" << std::endl;
    }
};

```

On vérifie que le fichier passé en paramètre existe

On ignore les premières lignes qui sont en commentaire

On récupère le nombre d'images à traiter

On itère sur les lignes pour récupérer toutes les caractéristiques des images

On ignore les lignes qui sont en commentaire et on n'incrmente pas le compteur

On récupère les informations sur les images, que l'on garde en mémoire.

On ignore les lignes qui sont en commentaire ou les lignes vides

On ajoute les correspondances dans la mémoire te

On associe les correspondances à l'image dont on a récupéré l'id.

2. Calcul de la matrice d'homographie

Comme nous l'avons vu précédemment, nous cherchons à résoudre l'équation suivante :

$$AX = \begin{pmatrix} 0 & 0 & 0 & -x & -y & -1 & tx & ty & t \\ x & y & 1 & 0 & 0 & 0 & -zx & -zy & -z \\ -tx & -ty & -t & zx & zy & z & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \\ \alpha_7 \\ \alpha_8 \\ \alpha_9 \end{pmatrix} = 0$$

Où $H = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 \\ \alpha_4 & \alpha_5 & \alpha_6 \\ \alpha_7 & \alpha_8 & \alpha_9 \end{pmatrix}$.

Ce qui nous donne l'algorithme suivant :

```
template<typename T>
void mosaic<T>::compute_homographie(const std::vector<correspondance<T>>& c, cv::Mat_<T>& h)
{
    const size_t size = c.size();
    h = cv::Mat_<T>(9, 1);
    cv::Mat_<T> A(3*size, 9);
    T* ptr = A[0]-1;
    typename std::vector<correspondance<T>>::const_iterator it = c.begin();
    const typename std::vector<correspondance<T>>::const_iterator it_end = c.end();
    T x,y,z,t,tx,ty,zx,zy;
    for(;it!=it_end; it++)
    {
        x=it->p2.x; y=it->p2.y; z=it->p1.x; t=it->p1.y;
        tx=t*x; ty=t*y; zx=z*x; zy=z*y;
        *++ptr = 0; *++ptr = 0; *++ptr = 0;
        *++ptr = -x; *++ptr = -y; *++ptr = -1;
        *++ptr = tx; *++ptr = ty; *++ptr = t;
        *++ptr = x; *++ptr = y; *++ptr = 1;
        *++ptr = 0; *++ptr = 0; *++ptr = 0;
        *++ptr = -zx; *++ptr = -zy; *++ptr = -z;
        *++ptr = -tx; *++ptr = -ty; *++ptr = -t;
        *++ptr = zx; *++ptr = zy; *++ptr = z;
        *++ptr = 0; *++ptr = 0; *++ptr = 0;
    }
    A/=10000;
    ptr=A[0];
    cv::SVD::solveZ(A,h);
    h=h.reshape(0,3);
}
```

Initialisation des paramètres

On initialise la matrice A

Résolution de l'équation

Redimensionnement de H pour avoir une matrice 3x3

On divise A par 1000 pour éviter d'avoir des éléments d'ordre de grandeur trop différent dans A et améliorer ainsi la stabilité de notre algorithme.

3. Recollement des images

Pour effectuer le recollement des différentes images, il s'agit en fait de calculer l'homographie entre l'image de référence et l'image que l'on souhaite ajouter. En règle générale, l'image de référence est l'image dans laquelle sont déjà recollé certaines images. Si nous n'avons pas encore effectué de recollement, nous avons choisi de prendre comme première image de référence celle où il y a le plus de correspondance.

```
template<typename T>
void mosaic<T>::compute_next_mosaic()
{
    int indice = -1;
    if(!img.size())
    {
        return;
    }
    indice = result.get_id_closest_img();
    if(indice<0)
    {
        std::cerr << "erreur:\n\tfile: " << __FILE__ << "\n\tline: " << __LINE__ << std::endl;
        std::cerr << "0 appariement ????" << std::endl;
        img.clear();
        return;
    }
    std::unique_ptr<Image<T>>& ptr = img[indice];
    cv::Mat_<T> h;
    std::cerr << "start compute_homographie " << indice << " " << result.get_id() << std::endl;
    compute_homographie(result[indice],h);
    std::cerr << "end compute_homographie " << std::endl;
    image<T> tmp;
    memcpy(tmp.h, h[0], sizeof(T)*9);
    for(const image<T>& i : ptr->get_img())
    {
        tmp.name = i.name;
        tmp.rows = i.rows;
        tmp.cols = i.cols;
        for(int j=0; j<4; j++)
        {
            tmp[j] = h * i[j];
            std::cout << i[j] << " " << tmp[j] << std::endl;
        }
        result.add_tex(tmp);
    }
    correspondance<T> c2;
    result.erase(indice);
    for(const auto& a : ptr->get_assoc())
    {
        if(a.first == result.get_id())
            continue;
        img[a.first]->erase(indice);
        for(const correspondance<T>& c : a.second)
        {
            c2.p1 = h * c.p1;
            c2.p2 = c.p2;
            result[a.first].push_back(c2);
            (*img[a.first])[result.get_id()].push_back(correspondance<T>(c.p2,c.p1));
        }
    }
    img.erase(indice);
}
```

On récupère l'indice de l'image la plus "proche" de l'image de référence, c'est à l'dire l'image qui a le plus de correspondance avec notre référence

Si cet indice est nul, ce qui signifie que l'image de référence n'a pas d'appariements avec les images restantes. on envoie un message d'erreur






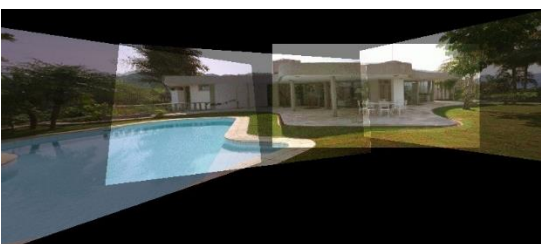
On calcule l'homographie entre l'image de référence et l'image la plus proche

On colle l'image la plus proche sur l'image de référence grâce à l'homographie

On calcule les nouvelles correspondances entre les images restantes et la nouvelle image de référence

On efface l'image que l'on vient de traiter des futures images à traiter.

Ce qui nous donne des résultats du type :

Image de référence	Image à recoller	résultat
		
		

Nous avons décidé de garder une légère transparence dans les images recollées pour pouvoir voir si le recollement se fait correctement. C'est pour cela qu'en regardant de près les images obtenues, nous pouvons constater l'apparition d'un léger flou aux endroits les plus éloignés des points d'appariement donnés. Ce qui est logique, puisque ce n'est pas à ces endroits que les deux images sont censées se correspondre parfaitement.

Passage à la pratique, sans fichier

Dans ce cas, nous n'utilisons plus de fichier donné par l'utilisateur et où est listée la liste des appariements entre les différentes images. Pour générer la liste des appariements, nous utilisons une interface qui permet à l'utilisateur de choisir les correspondances entre deux images en cliquant directement sur celles-ci.

Notre interface se construit comme suit :¹

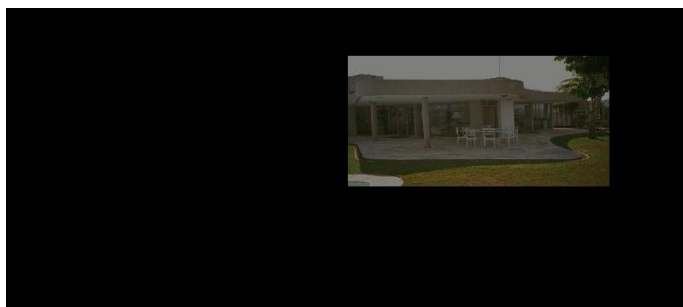
- Une première ligne où sont affichées les images en cours de traitement.
 - A gauche l'image de référence
 - A droite l'image à recoller.
- La deuxième ligne est la liste des images disponibles au recollement. Les images en surbrillance sont celles qui sont en cours de traitement.
- Et enfin, nous avons un dernier espace où s'affiche la liste des appariements sélectionnées entre les deux images en cours de traitement.



Une fois la liste des appariements créée, elle est enregistrée dans un fichier écrit de la même manière qu'expliqué [précédemment](#). A partir de là, la méthode utilisée pour calculer les recolllements est la même que celle vu dans la partie précédente. Nous avons obtenus les résultats suivants sur le set de photos de la piscine :

¹ Pour la liste des commandes, merci de se référer au README.md

Image de référence



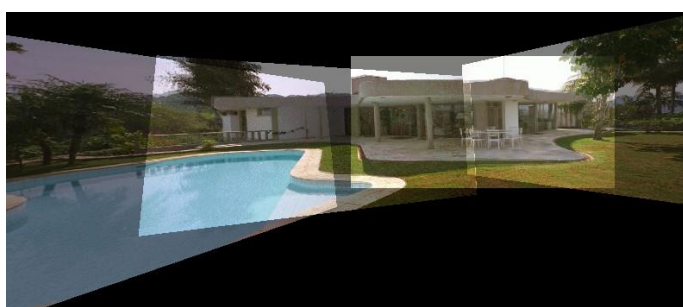
Recollement de la première image



Recollement de la deuxième image sur la mosaïque obtenue à l'étape précédente :



Image finale et recollement de la dernière image sur la mosaïque obtenue à l'étape précédente :



Nous pouvons constater un certain décalage au niveau du toit de la villa. Ceci est due au fait que les appariements que nous avons entré à la souris ne sont pas très précis et insère donc une part de bruit et d'erreur dans le calcul des matrices d'homographie et donc une inexactitude lors du recollement.