Table des matières

Chapitre 1

Extraction de segments à partir des contours

La toute première étape consiste à partir des images de départ, d'en extraire les contours. Ensuite à partir de ces contours on extrait des chaînes de pixels qui sont censées représenter les contours de l'image. Cette phase est importante car elle conditionne la *qualité* des segments qui seront extraits de l'image et donc la qualité de la reconstruction.

1.1 Gradient et maximum du gradient

Je ne reviens pas sur le filtrage antibruit appliqué à l'image, ni sur le calcul du gradient qui se font en fait simultanément par des convolutions des images de départ. La localisation précise se fait par recherche du *maximum du gradient dans la direction du gradient.*

1.2 Chainage des contours

Lors de l'extraction nous travaillons par hystérésis c'est à dire avec deux seuils. Le premier est celui qui conditionne l'ajout d'un point à une chaîne de contour, l'autre est la valeur maximale par laquelle doit passer une chaîne de pixels pour être considérée comme un contour valable :

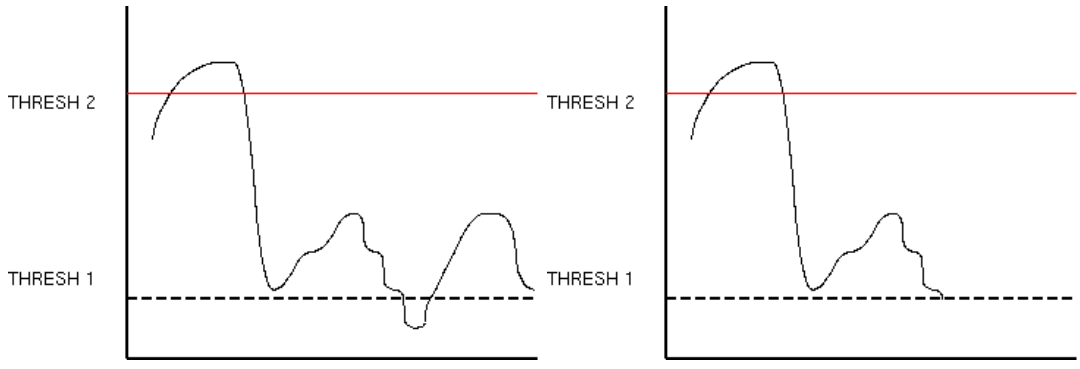

fig 1.1

1.3 Split/merge

Lorsque l'on dispose de ces chaines de pixels reste à en extraire des segments. C'est le travail des fonctions *split()* et *merge()* qui à partir d'approximations de *courbes* (ou ce qu'on peut assimiler

à des courbes) par des chaines de points, donnent des approximations de courbes par des lignes brisées.

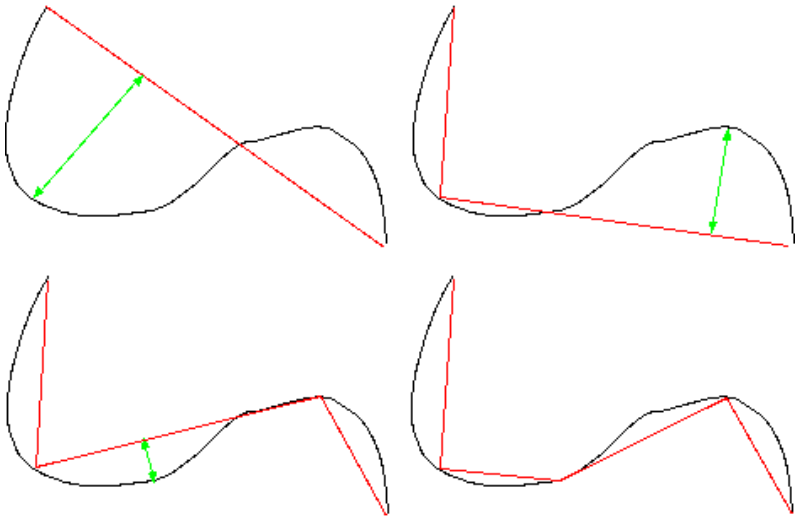

fig 1.2

La figure ci-dessous reproduit le processus de *splitage* des chaines de points. Ce processus étant suivi d'un processus opposé de *mergeage* des segments dont l'alignement est proche et qu'on pourrait réunir en un seul.

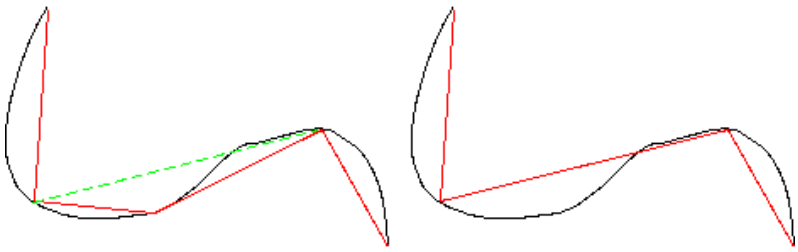

fig 1.3

L'alignement est défini par les différences d'angles entre les deux segments. Quelle est l'utilité de mergeage qui rend l'approximation plus grossière ? En fait, il ne sert à rien de découper les chaînes en tout petits segments qui approximeront parfaitement les petites variations des courbes de départ car on sait que la précision de la reconstruction 3D est d'autant plus grande que les segments à appareiller sont grands.

1.4 Découpage optimal

Ce découpage tel qu'il a été implémenté dans le projet n'est pas vraiment optimal car les segments obtenus dans l'une et l'autre vues sont indépendants les uns des autres, or les appariements dépendent de la qualité de ces découpages. J'ai préféré ne pas chercher à améliorer cette partie pour me consacrer au reste du projet étant donné que les images dont je disposais (cf dernier chapitre) étaient plutôt composés d'objets bien délimités aux arètes relativement droites. L'essentiel, me paraissait-il, était d'obtenir des segments en éliminant le plus possible les petites variations liées au bruit. Si j'avais eu plus de temps, j'aurais pu faire en sorte que ce découpage soit fait au niveau des appariements et non plus en amont, indépendamment du reste. Ainsi, j'aurais pu tenir compte des contraintes épipolaires lors du découpage ce qui aurait été l'idéal pour une image comportant de nombreux contours *courbes*.

Chapitre 2

Reconstruction 3D a partir d'un triplet de segments appareillés

Supposons que l'on ait les références de 3 segments 2D qui représentent la projection dans chaque image d'un segment de départ de la scène en 3D. En fait pour faire une reconstruction on n'a besoin que de deux segments. En effet, comme l'on connait la disposition exacte du plan de projection par rapport à la scène 3D, chaque extrémité d'un segment nous donne une "ligne de vue" qui matérialise en fait l'emplacement possible de l'extrémité dans l'espace. La connaissance d'une deuxième ligne de vue donnée par l'extrémité du segment dans l'autre image nous donne donc l'emplacement précis de ce point à l'intersection des deux droites.

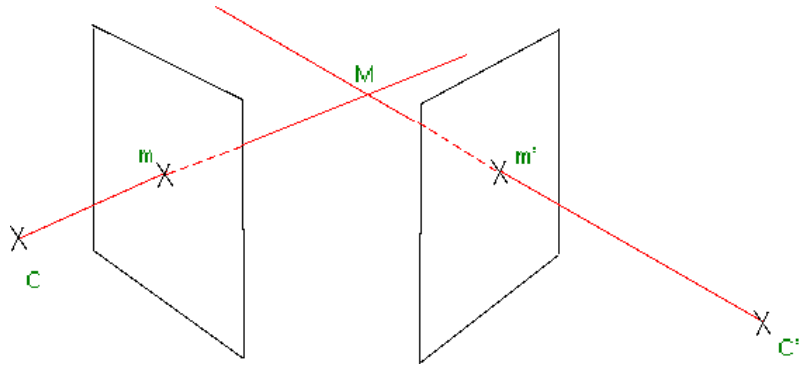

fig 2.1

Dans les faits les choses ne sont pas aussi simples. En effet, cela ne marche que si les points correspondent exactement ce qui est la plupart du temps faux, car à l'extraction les segments sont *tronqués*, divisés, légèrement déformés, pour des raisons diverses : le bruit, l'imprécision inévitable de la calibration de la prise de vue, ou le fonctionnement du processus d'extraction des segments. En fait, on ne peut pas se fier à la position précise des extrémités du segment, par contre on peut à peu près se fier à leur orientation. C'est ce qui est fait dans notre programme. Les segments avant d'être appariés sont *épipolairement* corrigés, de cela résulte que les extrémités appariés se correspondent parfaitement dans au moins deux vues, l'idéal étant qu'elles se correspondent dans les trois vues.

Une fois cela fait, chaque point 3D est représenté par 3 points 2D. Reste à retrouver les coordonnées 3D de ce point. On a un candidat potentiel qui est donné par la triangulation à l'aide de deux points sur les trois. A partir de là, on a mis au point une procédure qui minimise la distance sur chaque image entre le point projeté issu des calculs et le point projeté tel qu'on l'a localisé. Cette erreur est ensuite stockée pour chaque appariement possible et constituera un critère de choix des *meilleurs* appariements pour plus la suite.

Chapitre 3

Critères d'appariement des segments

La détection de contour et la reconstruction 3D, sont les parties du programme qui ne posent pas trop de cas de conscience. Il n'y a en effet pas trente-six manières différentes d'extraire des coordonnées 3D à partir de projections. Ce qui est plus délicat ce sont les critères d'appariement. Comment décider si un segment i s'appariera avec le segment j ou le segment k ? En fait il n'y a pas 1 critère qui fonctionnerait dans tous les cas, mais différents critères qui doivent être pondérés pour arriver au résultat optimal.

3.1 La contrainte épipolaire

Il y a tout de même des segments qui ne pourront jamais s'apparier, l'un avec l'autre, ce sont ceux qui sont épipolairement incohérents. Cette contrainte est moins forte pour des segments qui ont une certaine étendue que pour de simples points. De plus, il faut rajouter qu'on a *corrigé* les extrémités pour qu'elles soient cohérentes du point de vue de la géométrie épipolaire dans au moins deux couples de vues sur trois.

On se sert de cette contrainte à deux niveaux : le premier c'est pour faire un tri rapide des segments pour réduire le nombre des appariements possibles. On prend le milieu d'un segment dans une vue et on trace son épipolaire dans les deux autres vues, on ne retiendra pour les appariements que les segments qui sont coupés par cette épipolaire. On peut faire de même pour les deux autres vues si on veut des appariements *stricts*.

Une fois *corrigés*, les segments sont épipolairement cohérents dans deux couples de vues : 1 avec 2, 1 avec 3. Par contre, il peut y avoir des divergences en ce qui concerne le dernier couple de vues, 2 avec 3. Cette divergence entraine, une erreur lors de la reconstruction telle qu'elle a été exposé au chapitre 2. Cette erreur peut servir de mesure de la *force* de l'appariement dans les trois images, ce qui n'aurait pas été possible si on n'avait eu que deux images.

3.2 La contrainte d'unicité

On ne veut pas que le segment i soit apparié avec deux segments différents dans la même image.

3.3 Les contraintes géométriques

On suppose que les trois vues *se ressemblent,* c'est à dire ne diffèrent l'une de l'autre que d'un léger mouvement de caméra, sans quoi la reconstruction est tout simplement impossible. On peut mesurer le *déplacement* d'un segment entre deux images : avec cette contrainte, ce déplacement doit être à peu de chose près identique entre deux segments choisis arbitrairement. S'il ne l'est pas, on décrète que les deux appariements sont *incompatibles.*

Suivant la géométrie de la scène, on peut aussi déclarer incompatibles des appariements tels que deux segments intervertissent leur position entre deux vues. Cette contrainte n'est pas adaptée aux scènes qui présentent des objets sur différents plans et qui peuvent changer de position relative à cause des mouvements parallaxes. Elle n'a donc pas été implémentée.

On peut aussi imposer des contraintes sur les angles des segments. Un segment vertical ne pouvant pas s'apparier avec un segment horizontal par exemple. Ou encore, un segment qui serait dans la direction d'une droite épipolaire censée le couper, devra être éliminé de la liste, pour des raisons évidentes.

3.4 Les contraintes liées au gradient

Pour extraire les contours des images on a calculé les gradients. On peut imaginer qu'on conserve pour chaque segment la valeur moyenne du gradient pour les points qui le composent. Et on n'apparierait que les segments pours lesquels cette valeur de gradient serait compatible. J'aurais bien voulu l'implémenter, je n'en ai malheureusement pas eu le temps. Néanmoins, ce point me parait moins critique que dans le cas d'appariements à deux vues où les autres critères sont moins restrictifs.

Chapitre 4

Choisir les meilleurs appariements

4.1 Le graphe maximal

On a défini ci-dessus la notion de *compatibilité* entre les appariements. On peut représenter ces relations de compatibilité sous forme d'un graphe, où les noeuds seraient les différents appariements possibles et où les liens entre les noeuds symboliseraient leur *compatibilité*. Rechercher le meilleur appariement général possible consisterait, dans cette optique, à extraire un sous-graphe maximal[1] tel que tous ses noeuds soient entièrement reliés entre eux.

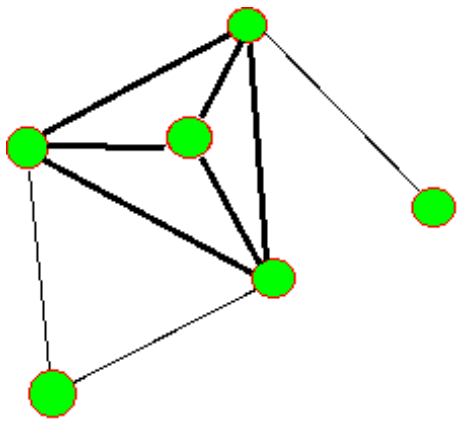

fig 4.1 : Graphe maximal en gras

Ce procédé, que j'ai commencé à mettre en oeuvre dans mon projet, manque de finesse, car il met toutes les incompatibilités sur le même plan ce qui est parfois gênant. Par ailleurs, LE graphe maximal n'existe pas la plupart du temps et donc, à moins de se contenter d'une solution intermédiaire, il faut mettre en oeuvre un autre processus de sélection. On pourrait peut-être s'en sortir en pondérant les liens et en cherchant à trouver un compromis entre la taille du sous-graphe et le poids de ses liens. Je n'ai pas eu le temps d'expérimenter (et je ne suis pas sûr de pouvoir arriver à quelquechose par cette voie là).

4.2 Relaxation

Soit un segment donné dans l'image 1, il est apparié à un certain nombre d'autres segments dans la deuxième et la troisième image. On voudrait qu'il ne soit plus apparié qu'à un seul segment dans la deuxième et un seul segment dans la troisième image. On suppose que le seul critère dont on dispose soit une mesure de la *force* de l'appariement sous la forme d'un coéfficient.

[1] maximal : tel qu'il n'existe pas de sous-graphe qui le comprenne strictement, et qui vérifie les mêmes conditions

Première solution, la plus simple : pour chaque segment, on choisit l'appariement au plus fort coéfficient. C'est rapide mais ça manque de précision, on a souvent des *ambiguités* : deux appariements possibles qui sont d'une force à peu près identique.

La solution choisie a été de classer les appariements pour chaque segment de la première image suivant la force de leur appariement. Ensuite on supprime les p% derniers appariements en s'assurant de laisser toujours au moins un appariement et en s'assurant également que le taux d'ambiguité (rapport entre les forces) entre l'élément enlevé et l'élément le mieux classé ne dépasse pas une certaine valeur. Puis on fait de même mais pour les segments de la deuxième image, et les segments de la troisième image. On itère le processus jusqu'à ce qu'il n'y ait plus d'appariements ambigus.

Chapitre 5

Les résultats

5.1 Les images de départ

Ce sont trois vues d'un même objet, avec des caméras dont les données étaient fournies sous forme des matrices de projection.

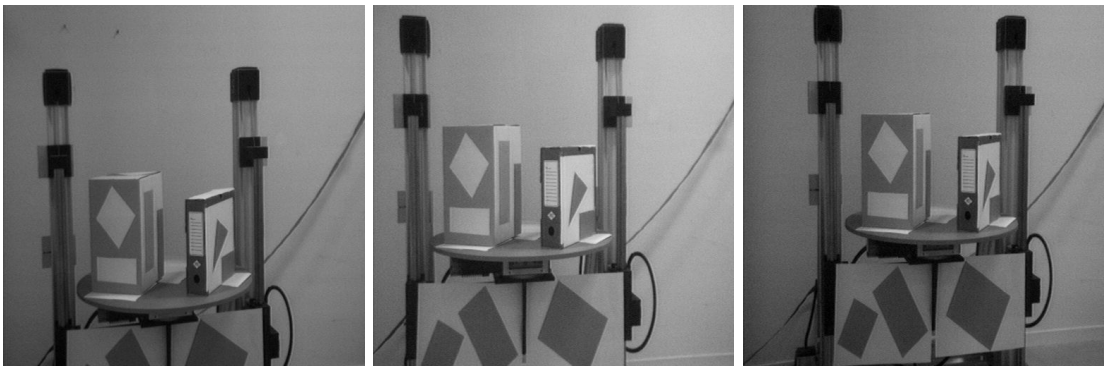

fig 5.1

5.2 Les résultats obtenus

Voici les images issues de l'extraction de contour :

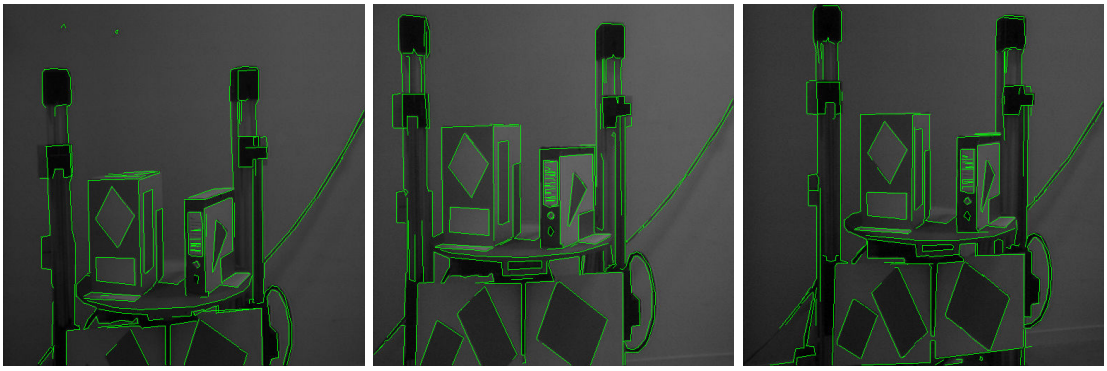

fig 5.2

Quand on lance le programme on obtient en sortie, un fichier geomview *output.off*, qui contient les coordonnées des segments 3D reconstitués.

Voici des photos d'écran de geomview montrant la scène reconstituée en 3D.

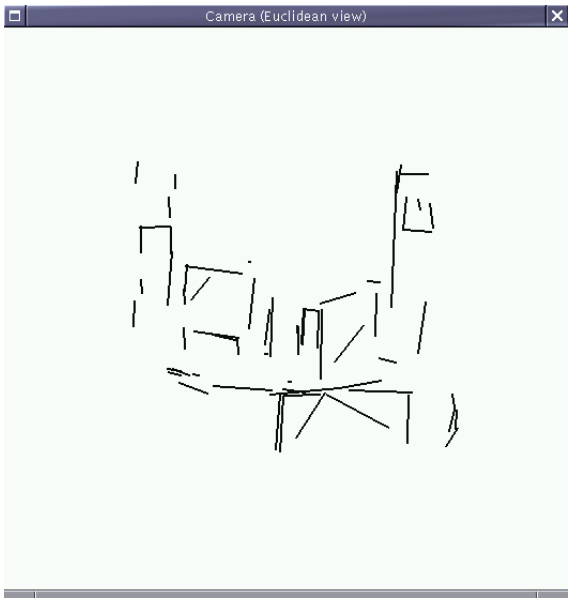

fig 5.3

On reconnait à peu près la scène sur cette vue-là. Par contre c'est beaucoup moins vrai si on change légèrement de point de vue.

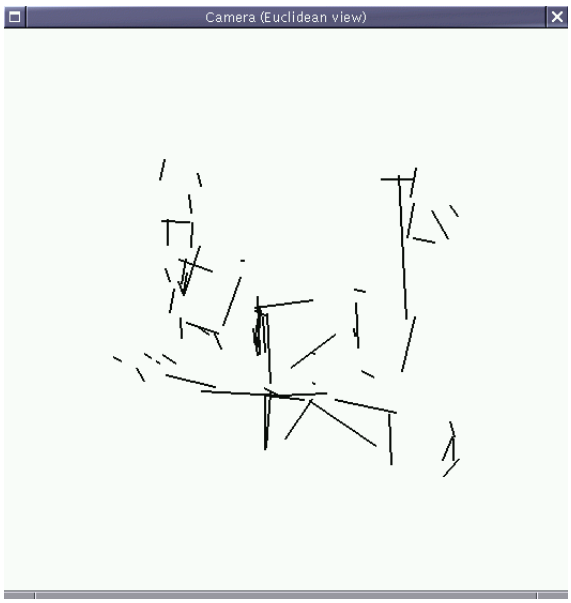

fig 5.4

Chapitre 6

Conclusion

Les résultats obtenus me semblent très décevants. J'avoue avoir passé un temps très important sur ce projet sans doute largement plus que ce qui était requis, mais l'implémentation de la moindre fonction me prenait un temps fou à programmer et surtout à déboguer étant donné qu'il s'agissait du premier projet que je réalisais en C++ : je ne rencontrai pas de difficulté conceptuelle mais l'écriture de la moindre ligne de code nécessitait un passage par la doc qui rallentissait tout. Le programme qui en résulte n'est donc pas très performant ni en temps de calcul ni en résultats, je n'ai pas eu le temps d'affiner ni mes procédures ni mes paramètres.

Un point positif tout de même, c'est que je crois que j'ai beaucoup appris pendant ce projet en lisant les docs et en mettant en oeuvre des idées et que si j'entamais désormais un autre projet en C++, je pourrais aller beaucoup plus vite sur la mise en oeuvre et me consacrer beaucoup plus au fond du programme.