Calculs parallèles pour le traitement des images satellites

Sylvain Contassot-Vivier

N° d'ordre : 87
N° attribué par la bibliothèque : 98ENSL0087

ÉCOLE NORMALE SUPÉRIEURE DE LYON
Laboratoire de l'Informatique du Parallélisme

THÈSE

pour obtenir le grade de
Docteur de l'École normale supérieure de Lyon
spécialité : Informatique

au titre de la formation doctorale de : Informatique de Lyon

présentée et soutenue publiquement le 27/02/98

par Monsieur Sylvain CONTASSOT-VIVIER

Titre :

Calculs parallèles pour le traitement des images satellites

Directeur de thèse : MIGUET Serge

Devant la Commission d'examen formée de :

Monsieur Jean-Marc	CHASSERY,	Rapporteur
Monsieur Serge	MIGUET	
Monsieur Thierry	PRIOL,	Rapporteur
Monsieur Yves	ROBERT,	Président
Monsieur Eric	THÉRON	
Monsieur Gérard	VIDAL	

À mes parents,

À ma famille,

Remerciements

Qui dois-je remercier pour tout ce joli travail ?
Et bien Moi ! na ! voilà !!

Bon allez, sérieusement :

Je remercie tout d'abord mes parents et toute ma famille, sans qui j'aurais sûrement eu beaucoup de mal à arriver jusque là. Ils m'ont toujours supporté et ont montré un grand intérêt à mes études.

Concernant le laboratoire du LIP où j'ai commencé ma thèse, je remercie tous les gens que j'ai pu y côtoyer et qui ont contribué à faire régner un contexte sympathique et détendu. J'ai une pensée particulière pour les personnes avec qui j'ai partagé mon bureau pendant de longs mois, Olivier, Loïc, Harald et aussi Stéphane.

Bien sûr, je n'oublie pas non plus les personnes du groupe Image du LIP, Annick, Jacquot et les deux Jean-Marc, dont certains ont su exprimer tout leur talent au babyfoot et au foot en salle.

De même, je ne saurais oublier le petit groupe de ma promotion du DEA et du MIM, bien que les appels irrésistibles de Marianne aient dispersé une bonne partie d'entre nous aux quatre coins de notre beau pays. Mais pas moi °|°‿

Enfin, ayant émigré du LIP, en deuxième année, pour suivre mon chef jusqu'au laboratoire ERIC de l'Université Lyon 2. Je dois dire que j'ai beaucoup apprécié les personnes et l'ambiance familiale que nous y avons trouvé.

Là aussi, je remercie particulièrement Fabien et David qui ont dû me supporter dans leur bureau pendant les derniers mois de ma thèse. Période bien connue pour susciter chez les doctorants une folie psychédélique. Bravo à eux, mais je suis sûr qu'ils se rattraperont quand leur tour viendra.

Je remercie aussi Laure pour son aide face aux épreuves quasi insurmontables de l'administration, ainsi que Didier pour son support technique au pied levé.

Enfin, je remercie vivement Messieurs Jean-Marc Chassery et Thierry Priol pour avoir accepté d'être rapporteurs de ma thèse, ainsi que tous les autres membres du jury, Yves Robert, Eric Théron et Gérard Vidal.

J'ai une pensée particulière pour Gérard, géologue "informaticien" avec qui, la collaboration a été un réel plaisir.

Finalement, le dernier remerciement revient de droit à Serge Miguet, mon directeur de thèse. Il a eu le courage ou la folie de me faire confiance et m'a pris sous sa tutelle pendant ces trois dernières années. Il a toujours fait preuve d'une grande attention à l'avancement de mes travaux. Mais surtout, il a su développer un climat plus amical que hiérarchique qui a contribué fortement à ma motivation et à l'accomplissement de cette thèse.

Voili voilou, c'est terminou...

Résumé

L'étude réalisée dans cette thèse met en relation deux domaines scientifiques, a priori distincts, que sont la géologie et l'informatique. En effet, le contexte de ce travail est de concevoir une chaîne complète de traitements parallèles sur les images satellites allant de la reconstruction tridimensionnelle à la visualisation des terrains ainsi reconstitués. Ce travail a donc fait l'objet d'une coopération étroite avec le département de géologie de l'École Normale Supérieure de Lyon.

Nous proposons d'une part, la parallélisation d'un algorithme de reconstruction tridimensionnelle de relief à partir d'un couple d'images satellite, et d'autre part, un algorithme parallèle de visualisation de terrains avec texture. Ces travaux font donc appel à plusieurs domaines de l'informatique tels que le parallélisme, la vision stéréoscopique et la synthèse d'images. Une étude méthodologique plus générale sur les algorithmes de transformation géométrique des images est également présentée.

Au niveau séquentiel, nous proposons pour chacun des algorithmes abordés et lorsque cela est pertinent, différentes optimisations originales permettant des améliorations en termes de complexité et donc de temps de calculs, ainsi que des choix d'outils calculatoires pouvant améliorer la qualité des résultats, point très sensible dans un domaine comme la vision stéréoscopique. Dans le cadre du parallélisme, nous nous focalisons sur les stratégies de communications et d'équilibrage des charges pouvant être mises en œuvre pour tirer le meilleur parti des machines parallèles. En comparant nos problèmes avec ceux déjà traités dans la littérature, nous sommes arrivés à la conclusion qu'un équilibrage des charges dirigé par les données était préférable à toute autre technique. De plus, que l'on se place dans la partie vision ou synthèse, l'équilibrage des charges peut être abordé exactement de la même manière. On peut donc appliquer la même stratégie sur ces différents algorithmes. Enfin, une étude théorique de la complexité de l'algorithme parallèle de vision stéréoscopique nous permet de déduire les points clés influençant les performances et donc d'estimer a priori le nombre de processeurs nécessaires pour obtenir les meilleures performances absolues pour un ensemble connu de données.

Des expérimentations menées sur différentes machines parallèles, Volvox, Cray T3D ou Cray T3E nous permettent de vérifier le bon comportement de nos algorithmes parallèles et de confirmer leur efficacité.

Mots clés : Algorithmique parallèle, vision stéréoscopique, équilibrage de charge, analyse et traitement d'image, partitionnement rectilinéaire, synthèse d'image.

Abstract

The study presented in this Phd thesis puts in relation two a priori distinct scientific domains that are geology and computer science. Indeed, the context of this work is to design a complete processing line of parallel tools about satellite images, going from the three dimensional reconstruction to the visualization of the retrieved grounds. This work has then implied a strong cooperation with the department of geology at the École Normale Supérieure of Lyon.

We propose on one side, the parallelization of an algorithm for three dimensional reconstruction of the relief from a couple of satellite images, and on another side, a parallel algorithm to visualize textured grounds. These works are then related to different domains of computer science like parallelism, stereo vision and image synthesis. A methodological and more general study over geometrical image transformation algorithms is also presented.

Concerning the sequential level, we propose for each of the algorithms seen and when it is relevant, different optimizations allowing to reduce the complexity and then the computation times, and some choices for the computational tools to use for increasing the quality of the results, which is a major point in the domain of stereo vision. Concerning the parallelism, we focus on the communications and the load balancing strategies that could be used to take the most advantage of the parallel machines. By comparing our problems to those already studied in the literature, we have arrived at the conclusion that a data driven load balancing was the best suited technique. Moreover, whatever considering the vision or synthesis part, the load balancing problem can be seen exactly in the same way. We can then apply the same strategy over these different algorithms. Finally, a theoretical study of the complexity of our parallel stereo vision algorithm allows us to predict the number of processors necessary to get the best absolute performances for a given input data set.

Experimental results done on several parallel machines, Volvox, Cray T3D or Cray T3E allow us to verify the behavior of our parallel algorithms and to confirm their efficiency.

Keywords : Parallel algorithm, stereo vision, load balancing, image analysis and processing, rectilinear partitioning, image synthesis.

Table des matières

Table des figures

Liste des tableaux

Introduction

Depuis l'apparition d'une petite lueur d'intelligence dans le cerveau humain, celui-ci, mû par une curiosité sans borne, n'a cessé d'étudier toutes les choses qui l'entouraient. Parmi celles-ci, il en est une qui a une importance toute particulière sur notre vie, à la fois mère protectrice et corne d'abondance, sans doute le plus beau et le plus gigantesque vaisseau spatial que l'on peut imaginer, c'est la Terre. Celle-ci n'est pas une chose inerte, dire qu'elle vit serait sans doute exagéré, mais elle possède une activité interne et externe relativement importante, dont un exemple bien connu est les volcans. Cette activité a pour conséquence de modifier continuellement la structure de l'écorce terrestre sur laquelle nous marchons tous les jours et peut avoir des conséquences directes sur notre vie. C'est pourquoi, depuis bien longtemps, on essaie de comprendre sa structure et de prévoir son évolution, donnant ainsi naissance à la géologie.

Si l'on revient à une époque plus contemporaine, une autre science a vu le jour il n'y a pas si longtemps et surtout son avènement remonte à seulement un demi siècle : c'est l'informatique. C'est sans doute un des domaines qui a connu la progression la plus rapide. En effet, on est passé de machines complexes à utiliser et ne pouvant effectuer que quelques opérations de base au milieu de notre siècle, à des machines pouvant effectuer plusieurs millions de calculs à la seconde et possédant une interface Homme-machine évoluée et utilisable par le plus grand nombre, de nos jours. On assiste donc, à l'heure actuelle, à une course à la vitesse et à la taille des données traitées. On doit traiter toujours plus vite des ensembles de données toujours plus gros. Pour continuer cette évolution au-delà des contraintes physiques qui commencent à limiter la vitesse pure des processeurs, le principe du parallélisme, déjà utilisé depuis des millénaires, a été repris. En effet, dans tous les domaines où le travail à effectuer est important et les délais relativement courts, comme la construction de bâtiments, de bateaux, etc . . . , plusieurs personnes sont employées en même temps pour accélérer le processus. L'idée du parallélisme en informatique est similaire et consiste à faire travailler ensemble plusieurs processeurs séquentiels.

On peut finalement se demander quel peut être le lien entre ces deux domaines de recherche que sont la géologie et l'informatique parallèle. En fait, il est relativement simple, l'informatique fournit de nouveaux outils permettant aux géologues de faire des études plus précises et plus poussées. Même si l'on ne remplacera jamais les études sur le terrain, les outils informatiques permettent d'éviter des déplacements longs et coûteux que tous les chercheurs ne peuvent s'offrir.

L'étude que nous présentons ici, concerne donc un de ces outils. Plus précisément, il s'agit de reconstruire le relief d'un terrain observé par deux caméras à des points de vue légèrement

différents. Ces données sont une source d'information très recherchée par les géologues car elles permettent d'étudier la topographie du terrain en question à la fois dans sa globalité mais aussi sur des zones précises. Si l'on ajoute à cela, un programme pour visualiser ces terrains avec leur texture d'origine (déduite des images initiales), il est alors possible de faire des observations sous n'importe quel angle de vue. De plus, il n'est pas compliqué d'intégrer dans un tel programme des outils supplémentaires pour faire des calculs divers, pentes, courbes de niveaux, etc ...

Néanmoins, si l'on veut obtenir des terrains avec une bonne précision, il faut utiliser des images qui soient elles-mêmes très précises. Ce type d'images est fourni par plusieurs satellites dont SPOT qui permet d'obtenir des images avec une précision de 10 mètres. Une telle finesse a une conséquence directe sur la taille des images. En effet, si l'on observe un zone de 10 Km sur 10 Km, on aura une image de taille 1000×1000. Or, SPOT observe des zones d'environ 60 Km de côté, ce qui représente des images 6000×6000. On a donc un ensemble de données à traiter qui est très volumineux, qui requiert une capacité mémoire très importante et des calculs très longs.

Justement, le double intérêt du parallélisme et d'accélérer les temps de calcul mais aussi de permettre le traitement d'ensembles de données de plus grande taille. Mon travail de thèse a donc consisté à mettre en œuvre plusieurs algorithmes parallèles constituant une chaîne complète de traitement des images satellites. Cette chaîne comprend le pré-traitement des images brutes, la reconstitution tridimensionnelle du terrain puis sa visualisation.

Avant de commencer la description des ces algorithmes, nous rappelons brièvement les différents types de programmation parallèle qui existent ainsi que les types de machines.

Il y a principalement deux modèles de programmation parallèle : le parallélisme de tâches et le parallélisme de données. Le premier consiste à diviser le travail en tâches différentes et à les faire exécuter indépendemment les unes des autres en autorisant toutefois, lorsque cela est nécessaire, des échanges de données entre les tâches. Le second revient à faire exécuter le même travail par toutes les unités de calculs (processeurs) mais sur des données différentes.

Concernant les machines parallèles, il en existe aussi de grands types : les machines SIMD (Single Instruction Multiple Data) et les machines MIMD (Multiple Instruction Multiple Data). Sur les machines SIMD, tous les processeurs sont synchronisés pour exécuter la même instruction en même temps. Ce type de machine est particulièrement adapté au parallélisme de données. Au contraire, sur les machines MIMD, les processeurs fonctionnent de manière asynchrone indépendemment les uns des autres. Ces machines sont plus polyvalentes, et s'adaptent aussi bien au parallélisme de tâche qu'au parallélisme de données.

Enfin, il existe une autre méthode de programmation des machines MIMD qui combine les deux approches précédentes : SPMD (Single Program Multiple Data). Elle permet d'exécuter les mêmes tâches de manière asynchrone sur des données différentes.

Une dernière caractéristique importante des machines parallèles concerne la mémoire. Elle peut être soit distribuée sur les processeurs, soit partagée par ceux-ci. Dans le premier cas, chaque processeur possède sa propre mémoire à laquelle il est le seul à pouvoir accéder. Pour échanger des données, les processeurs doivent donc utiliser des passages de messages à travers un réseau de

communication. Dans le second cas, l'espace mémoire est commun à tous les processeurs et chacun d'eux peut accéder à tout cet espace. L'utilisation de messages n'est donc pas nécessaire.

Les algorithmes présentés dans cette thèse sont écrits selon le modèle SPMD, et sont testés sur des machines MIMD à mémoire distribuée (réellement ou de manière simulée).

Le plan de la thèse est le suivant :

Chapitre 1 : Étude de la littérature : dans ce chapitre, nous présentons une rétrospective des différentes techniques de stéréovision et plus particulièrement de reconstruction tridimensionnelle.

Chapitre 2 : Présentation de l'algorithme de Memier : l'algorithme séquentiel de M.Memier, que nous avons pris comme point de départ de notre étude sur la reconstruction tridimensionnelle est présenté en détails. Nous expliquons aussi pourquoi nous avons fait ce choix.

Chapitre 3 : Optimisations séquentielles et Parallélisation : nous passons en revue les modifications apportées à l'algorithme séquentiel de Memier ainsi que sa parallélisation. Des résultats expérimentaux sur machine Cray T3E permettent d'évaluer l'efficacité des nouveaux algorithmes ainsi obtenus.

Chapitre 4 : Algorithme parallèle de visualisation de terrains texturés : les différentes techniques parallèles de visualisation sont décrites dans un premier temps. Ensuite, nous décrivons un algorithme parallèle de Z-buffer qui a été développé au LIP et un algorithme séquentiel de visualisation de terrains que j'avais mis au point lors de mon stage de licence. Enfin, à l'aide de ces deux algorithmes, nous déduisons une version parallèle que nous testons sur les machines Volvox et Cray T3D.

Chapitre 5 : Algorithme parallèle de transformations géométriques d'images : Après une brève étude bibliographique, nous décrivons un schéma parallèle général pour effectuer des transformations d'images et nous étudions trois variantes de ce schéma selon la répartition des données et la méthode de calcul utilisée. La comparaison des résultats expérimentaux de ces trois variantes, sur Cray T3E, nous permet de mettre en évidence les stratégies les plus efficaces pour ce type d'algorithme.

Chapitre 6 : Conclusion et Perspectives: finalement, les résultats précédents vont nous permettre de conclure sur l'efficacité et la pertinence de l'utilisation du parallélisme pour ces types de problèmes. Nous dégageons aussi les paramètres importants qu'il faut prendre en compte et les compromis à faire lors de la conception d'un algorithme parallèle. Enfin, nous présentons les perspectives d'études que nous envisageons pour compléter ce travail.

CHAPITRE 1

Étude de la littérature

Une partie importante du travail présenté dans cette thèse repose sur l'algorithme séquentiel de reconstruction de terrains développé par M.Memier lors de son doctorat. Celui-ci sera d'ailleurs présenté en détails dans le chapitre 2. Pour bien comprendre les choix réalisés dans cet algorithme et les alternatives envisageables, nous présentons, dans ce chapitre, un tour d'horizon des travaux déjà effectués dans ce domaine.

Dès que les ordinateurs ont pu être équipés de capteurs tels que des caméras, permettant d'avoir une information visuelle sur le monde extérieur, le problème de la vision par ordinateur s'est posé et a suscité un engouement important. Il en résulte un grand nombre d'études liées à la vision par ordinateur. Cependant, lors de ce chapitre, nous nous cantonnerons aux principales techniques relatives à notre problème particulier de stéréovision. Nous faisons ressortir les caractéristiques propres à chacune d'elles dans le but d'en faire une classification. Pour plus d'informations, le lecteur peut se référer à plusieurs études bibliographiques plus complètes telles que [BF82, DA89, Bro92, Zha93].

Le problème majeur de la stéréovision revient à faire une mise en correspondance d'images (*image registration* en anglais). Celle-ci peut être décomposée en quatre grandes classes en fonction du type de problème traité:

- Mise en correspondance multimodale,
- Mise en correspondance de motifs,
- Mise en correspondance de points de vue,
- Mise en correspondance temporelle.

La première classe s'applique sur des images issues de différents types de capteurs, elle sert notamment à la segmentation. La deuxième consiste à trouver la localisation d'un motif dans une image et est utilisée régulièrement pour l'interprétation des éléments d'une image. La troisième est utilisée sur des images d'une même scène prises de points du vue différents, elle sert notamment à la reconstruction de l'information tridimensionnelle. C'est principalement dans cette classes de techniques que nous allons travailler. Enfin, la dernière classe correspond à des images d'une même scène prises à différents moments où sous différentes conditions, par exemple pour déterminer les

changements de l'organisation du territoire ou la croissance d'éléments naturels (forêts, ...).

Pour notre problème, nous allons travailler sur deux images satellites prises à des points de vues légèrement décalés. Nous nous cantonnerons donc aux techniques dites binoculaires. Malgré tout, pour avoir un aperçu global du domaine et bien positionner notre problème dans celui-ci, nous distinguons les différentes techniques en fonction du nombre de points de vue utilisés. Il existe les techniques basées sur une seule image (paragraphe 1.1), celles utilisant deux images, qui sont les plus courantes et donnent des résultats plus précis (paragraphe 1.2), et enfin, des techniques affinant encore la fiabilité des mises en correspondance, qui font appel à plus de deux images (paragraphe 1.3).

1.1 Techniques monoculaires

Il est possible de retrouver le relief d'une scène à partir d'une vue unique (*one-eyed stereo*). Les deux principales méthodes reposent sur la luminosité et sur la texture des objets observés.

1.1.1 Lumière

L'idée première des techniques basées sur la luminosité (*Shape from Shading*) remonte à Fesenkov en 1929 mais, le premier algorithme réel fût introduit en 1975 par Horn [Hor75] Le principe consiste à déduire les informations relatives à l'orientation de la surface par rapport à l'éclairage: une source lumineuse connue ou estimée éclaire des parties des objets et pas d'autres avec des intensités différentes. Ces travaux ont ensuite été complétés et modifiés par de nombreuses études, citons par exemple [VY93, WH97, SL97]. Chacun d'eux utilise différents types de modèles de lumière et fait des hypothèses (en général Lambertiennes) sur les surfaces observées. Justement, le problème majeur de ces techniques est qu'elles ne sont utilisables que dans certains cas particuliers et s'appuient sur des hypothèses très fortes sur la nature des matériaux observés ainsi que sur l'éclairage. De plus, elles ne produisent que des cartes d'orientation de surface et requièrent donc des points initiaux connus pour obtenir des valeurs absolues.

Ces techniques peuvent être améliorées en utilisant plusieurs images, toutes prises d'un même point de vue, mais avec des caractéristiques d'éclairage différentes. Par exemple une même prise de vue à des heures différentes de la journée donne un ensoleillement différent. Ces méthodes, dites photométriques, ont été définies au début des années 80 par Woodham [Woo80]. Tout un ensemble d'études ont aussi été menées sur cette technique [WIB91, Td91, KB91, TD92] en utilisant différents types de lumières. Si la méthode permet d'améliorer parfois la qualité des résultats elle entre dans le cadre de la vision active et n'est pas toujours applicable.

1.1.2 Texture

Toujours dans le même cadre d'idées, des techniques basées sur la texture des objets (*Shape from Texture*) ont été étudiées dès les années 50 [Gib50] et reprises ensuite, notamment par Kender [Ken80] qui explora minutieusement la théorie sous-jacente. Celle-ci s'appuie sur la déformation des textures lorsque l'on regarde depuis différents points de vue, l'aire du morceau d'image où il y a la texture considérée se modifie selon la distance et l'orientation. Cependant, ces techniques sont très restreintes de par leurs calculs coûteux.

Enfin, en partant de l'idée que les techniques de SFS et SFT prises seules n'étaient pas suffisantes pour retrouver la structure tridimensionnelle de scènes naturelles texturées à cause justement de la coexistence des informations de luminosité et de texture, des tentatives de combinaison des deux techniques telles que celle de Choe et Kashyap [CK91] ont été développées. Les auteurs utilisent un nouveau modèle tridimensionnel considérant l'image de la scène comme une superposition d'une texture aléatoire et de l'image éclairée lisse (non texturée).

1.1.3 Hiérarchie

De la même manière que précédemment, l'idée de l'insuffisance d'une seule technique a amené certains chercheurs à utiliser tout un ensemble de sous-algorithmes différents pour traiter chaque partie des images. Cela permet de prendre en compte les diverses possibilités contextuelles qui sont détectées seulement par certains algorithmes et pas par d'autres. Un exemple de cette technique est donné dans [Gra90].

1.2 Vision binoculaire

Le principe de la vision binoculaire est directement dérivé du système visuel humain [MP79] (et des animaux en général). L'idée est que l'on peut déduire une information tridimensionnelle à partir des différences de position (disparités) des éléments entre les deux images. Ces disparités sont dues à la projection perspective de la scène sur le plan de l'image. En fait, si l'on considère un point P(X,Y,Z) dans la scène, ce point va se projeter sur les deux images I et I' en p(x,y) et p'(x',y'). Faisons maintenant le raisonnement inverse, un point (ou pixel) dans une image et le centre optique de la caméra qui a généré l'image décrivent un rayon 3D qui traverse la scène. Ce rayon intersecte un objet dans la scène et la couleur de l'objet à cet endroit détermine la couleur du pixel considéré. Ainsi, si l'on sait que deux points p(x,y) et p'(x',y') dans les deux images représentent un même point 3D de la scène, et si l'on connaît la position relative des deux caméras, on peut retrouver les coordonnées du point 3D (dans un repère arbitraire, en général relatif aux caméras) correspondant qui se trouve à l'intersection des deux rayons formés par les pixels et les centres optiques de leurs images respectives. Le schéma de la figure 1.1 montre la géométrie du système de stéréovision binoculaire.

Notre problème de reconstruction tridimensionnelle revient donc à apparier entre eux des élé-

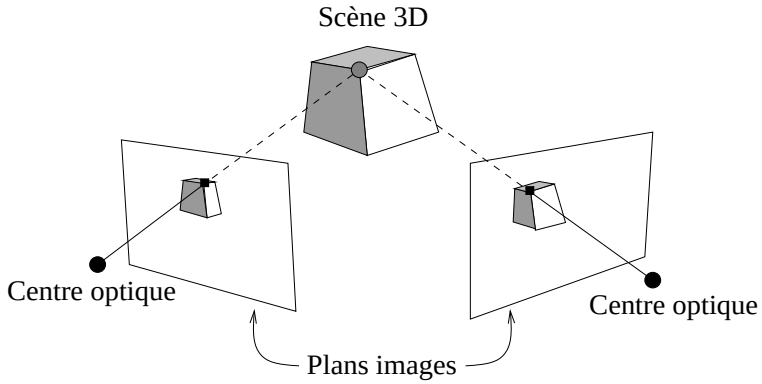

FIG. 1.1 - *Géométrie du système de vision avec deux caméras*

ments dans les deux images pour en déduire leur distance par rapport à celles-ci. On obtient donc une carte des profondeurs que l'on peut éventuellement repositionner dans un repère absolu si l'on connaît les coordonnées dans ce repère et dans les images de certains points de la scène.

1.2.1 Approches pour la Stéréovision

La mise en correspondance d'images est une tâche fondamentale en vision par ordinateur. Elle permet de résoudre des problèmes pratiques dans différents domaines tels que la vision par ordinateur, l'imagerie médicale, le traitement de données acquises à distance. Par exemple, elle est souvent utilisée pour intégrer des informations provenant de différents capteurs, trouver les changements entre deux images acquises à des moments différents sous des conditions différentes, retrouver l'information tridimensionnelle ou encore la reconnaissance d'objets.

Les techniques de mise en correspondance sont, dans leur forme la plus générale, composées de quatre éléments :

- L'espace des éléments à apparier,
- Une mesure de similarité,
- L'espace de recherche,
- La stratégie de recherche.

Les choix faits pour chacun de ces éléments définissent une technique particulière. L'étendue des problèmes traités a mené à une quantité de techniques qui rend difficile toute classification ou comparaison car, en général, ces méthodes ont été développées pour des applications particulières et non pas dans une optique globale liée à un type de problème.

1.2.2 Les composants du processus

Éléments à apparier

Le premier choix qui s'impose pour faire une mise en correspondance de deux images concerne la nature des éléments à apparier. Les différents types d'éléments peuvent être très divers. On citera parmi les plus utilisés, les valeurs brutes des pixels (intensités), les arêtes [BB91, KD94, YPZC95], les contours [HA89, SP90, CBA93], les coins, les intersections de lignes, les courbes [Nas92], les valeurs statistiques [GGB84, GL94, FHM+93, LM95], et d'autres éléments de plus haut niveau tels que des graphes représentant les relations entre primitives dans une image (méthodes structurelles [BL84, HOE94]). En général, le choix est fait en fonction des critères suivants :

- la minimisation de la sensibilité au bruit généré par le capteur ou d'autres déviations telles que les différences d'illumination,
- le choix des propriétés des images à apparier (par exemple les formes géométriques ou bien des informations sur les textures, ...),
- le coût des calculs.

De plus, la nature des primitives influe de manière significative la complexité des calculs et donc le temps du processus d'appariement. Le choix doit donc être fait en considérant tous ces aspects en plus du problème initial.

Mesure de similarité

Le deuxième élément à choisir dans le processus est la mesure de similarité. Elle est étroitement liée aux primitives à apparier puisqu'elle mesure la similarité entre celles-ci. Elle indique donc directement ce que l'on veut mettre en correspondance et symétriquement, ce qui ne nous intéresse pas.

Les mesures les plus courantes sont :

La corrélation croisée qui mesure la similarité entre deux motifs de mêmes tailles. Elle est souvent utilisée pour retrouver la position optimale d'une sous-image dans une image et donc pour mettre en correspondance des pixels en prenant en compte leur contexte de texture [GGB84, FHM+93]. Il existe plusieurs critères de corrélation, si l'on reprend les notations de [FHM+93], on considère les deux images I_1 et I_2, la fenêtre de corrélation de taille (2n+1)×(2m+1) et la disparité horizontale d (on se place dans le cas d'une géométrie épipolaire, sinon il faudrait simplement ajouter une disparité verticale). Le critère de base est défini par :

$$C(x,y,d) = \frac{\sum_{i=-n}^{n}\sum_{j=-m}^{m} I_1(x+i,y+j)I_2(x+d+i,y+j)}{\sqrt{\sum_{i=-n}^{n}\sum_{j=-m}^{m} I_2(x+d+i,y+j)^2}}$$

où le dénominateur sert à normaliser la corrélation de façon à ne pas être sensible à l'intensité locale relative de l'image. Toutefois, d'autres critères plus robustes, plus précis ou donnant des mesures absolues ont été définis, le plus répandu est sans doute le coefficient de corrélation défini par :

$$C(x,y,d) = \frac{Covariance(I_1(x,y), I_2(x+d,y))}{Variance(I_1(x,y)) \times Variance(I_2(x+d,y))}$$

soit

$$C(x,y,d) = \frac{\sum_{i,j}(I_1(x+i,y+j) - \mu_1(x,y))(I_2(x+d+i,y+j) - \mu_2(x,y))}{\sqrt{\sum_{i,j}(I_1(x+i,y+j) - \mu_1(x,y))^2 \times \sum_{i,j}(I_2(x+d+i,y+j) - \mu_2(x,y))^2}}$$

où $\mu_k(x,y)$ représente la moyenne des intensités sur la fenêtre de corrélation centrée en (x,y). Une propriété intéressante de cette mesure est qu'elle n'est pas modifiée si l'on remplace I_1 et I_2 respectivement par $a_1 I_1 + b_1$ et $a_2 I_2 + b_2$, autrement dit elle est invariante aux transformations affines des intensités des images pouvant apparaître lorsque les deux caméras sont légèrement différentes. Il existe encore d'autres mesures moins complexes, mais souvent aussi moins robustes. Lorsqu'il y a trop de bruit dans l'image, on peut utiliser un pré-filtrage pour enlever celui-ci (dans la mesure où l'on connaît sa nature et qu'on peut donc le modéliser). De plus, le pré-filtre et le motif peuvent être fusionnés pour obtenir un filtre réalisant les deux opérations à la fois. De la même manière, on peut utiliser un ensemble de filtres comme dans [JM91] où les auteurs utilisent des filtres linéaires d'orientations différentes et à des échelles différentes leur permettant une description assez riche des images. Le processus d'appariement travaillera ensuite sur ces descriptions. La somme des valeurs absolues des différences des intensités [BS72, Zha93] permet des calculs rapides et donne de bons résultats quand il n'y a pas de distorsions locales. Alors que la somme des valeurs absolues des différences des contours [YPZC95] est quand à elle, plus robuste à ces distorsions locales. La corrélation de phase basée sur la transformée de Fourier [YS89, Bro92, SN96] des images est bien adaptée en présence de bruit dépendant de la fréquence du signal. Le nombre de changements de signe lors des différences d'intensité entre pixels est une bonne mesure lorsque l'on a des images non similaires. Enfin, notons aussi les différences de contours/surfaces. Finalement, il existe aussi des mesures de plus haut niveau telles que la comparaison structurelle, basées par exemple sur des graphes [BL84].

Mais encore une fois, la mesure choisie dépend du type d'éléments que l'on veut apparier. Par exemple, si l'on utilise des courbes [Nas92], une mesure telle que la somme des carrés des différences entre les points les plus proches est plus adaptée. De même, avec les méthodes structurelles, on utilisera une mesure de similarité entre graphes ou arbres.

Enfin, il existe bien d'autres mesures qui peuvent être définies au cas par cas pour un problème particulier. De même, pour chaque type de mesure, il existe des variantes comme dans le cas de la corrélation croisée où l'on peut trouver un nombre important de mesures statistiques plus ou moins complexes à calculer. De plus, comme on l'a vu dans l'énumération précédente, chacune de ces mesures présente des avantages et des inconvénients. En effet, certaines sont moins sensibles à certains types de bruits ou distorsions du signal. L'important est donc de bien réfléchir à la mesure la plus précise pour apparier les primitives données tout en étant la moins sensible possible aux différents bruits dans les images.

Espace de recherche

L'espace de recherche joue aussi un rôle non négligeable dans le processus d'appariement car de lui dépend directement la quantité de calculs effectués (nombre de comparaisons à faire pour chaque primitive à apparier). En effet, pour chaque élément à apparier, on va balayer l'espace de recherche correspondant et tester chaque candidat dans cet ensemble. Il est donc directement lié aux éléments à apparier et à la stratégie de recherche. Il est généralement choisi de manière à être de taille minimale par rapport au problème posé. D'ailleurs, dans la plupart des cas, des contraintes géométriques sont utilisées pour réduire cet espace.

Par exemple une des contraintes les plus courantes est la géométrie épipolaire basée sur le système de vision [ZDFL95, PD96]. Cela permet de réduire considérablement l'espace de recherche car la géométrie épipolaire implique que deux points correspondants sont sur une même ligne épipolaire. En général, un précalcul est réalisé pour transformer les images dans cette géométrie de manière à ce que les lignes épipolaires correspondent aux lignes horizontales des nouvelles images.

De même, les contraintes d'unicité et de continuité des appariements définies par Marr et Poggio en 1979 [MP79], permettent de conditionner fortement la recherche. La contrainte d'unicité affirme que selon une direction fixée de vision (un rayon formé par un pixel et le centre optique de la caméra), on ne peut voir qu'un objet. En fait, cette hypothèse est clairement fausse dans le cas, par exemple, d'objets transparents mais pour la plupart des scènes courantes et dans le cas particulier des vues aériennes, elle reste valide et peut être reformulée en disant que chaque direction de vue n'admet qu'une correspondance dans l'autre image. De même, la contrainte de continuité suppose que les surfaces des objets observés sont continues. Donc, les distances des objets par rapport à la caméra (donc les disparités) varient continuellement selon les directions de vue sauf aux frontières des objets. Une autre contrainte souvent utilisée est la contrainte d'ordre qui empêche d'avoir des appariements croisés. Si l'on se place en géométrie épipolaire et que l'on considère deux points homologues $P_1(x_1, y)$ et $P'_1(x'_1, y)$, alors l'homologue d'un point $P_2(x_2, y)$ avec $x_2 > x_1$ sera de la forme $P'_2(x'_2, y)$ avec $x'_2 >= x'_1$. Toutefois cette contrainte ne doit être utilisée que dans certains cas précis car elle n'est pas toujours vérifiée. Par exemple, dans le cas d'objets avec des trous. Une autre contrainte que l'on trouve encore est la disparité maximale qui est définie par rapport à la géométrie du système visuel (position relative des caméras) et par le dénivelé maximal dans la scène. Cette contrainte est très utile car elle permet dans le cas épipolaire de ne pas avoir à comparer tous les points de la ligne de l'autre image mais de se restreindre à un intervalle donné autour d'une position privilégiée. Enfin, il existe d'autres contraintes particulières liées à la nature des éléments observés.

Stratégie de recherche

La stratégie utilisée pour rechercher les correspondances est primordiale à deux niveaux. D'une part, pour les temps de calculs nécessaires à sa mise en oeuvre, et d'autre part, pour la qualité des résultats.

Il est assez difficile de classifier les diverses stratégies existantes. D'une part parce qu'elles

sont nombreuses et variées. De plus, elles ont toutes des avantages et inconvénients donnant des résultats différents en fonction des autres composants du processus et de la nature des images traitées. Certaines sont limitées à un domaine particulier, d'autres peuvent être utilisées ensembles (combinaison) pour améliorer les résultats.

Enfin, certaines conventions sont utilisées pour faciliter le processus. Par exemple, lorsque l'on a deux ensembles de primitives issus chacun de l'une des images, une méthode couramment utilisée (mais pas toujours, selon la stratégie de recherche) est de prendre l'un des deux ensembles comme référence (la symétrie du problème nous assure qu'il n'y a pas de perte de généralité quant au choix de la référence) et de rechercher pour chacun des éléments de cet ensemble, son homologue dans l'autre ensemble.

En fait, la dépendance entre les différents composants du processus d'appariement intervient encore ici. En effet, la stratégie de recherche mise en oeuvre dépend principalement de la nature de l'espace de recherche. Parmi les techniques les plus courantes, citons :

- les techniques hiérarchiques ou de multirésolution [CVSG89, Gra90, YPZC95], appliquées pour accélérer les calculs en guidant les recherches dans une résolution donnée par rapport aux résultats obtenus avec une résolution moindre.

- La décision en séquence [BS72], utilise un seuillage sur l'accumulation des mesures de similarité sur l'espace de recherche et choisit l'appariement en prenant le candidat ayant le plus grand nombre de points testés avant de dépasser le seuil. Cela permet de réduire de manière significative la complexité des calculs sans perdre beaucoup au niveau de la qualité.

- La relaxation [LHB87, KDKA92, Nas92, ZDFL95], est une méthode itérative et localement parallèle. Elle prend en compte les similarités locales dans le voisinage de la primitive considérée et met à jour les mesures basées sur des probabilité de correspondance à chaque itération en tenant compte de la cohérence dans ce voisinage. Le problème essentiel de cette approche est réside dans le choix de la loi à appliquer pour mettre à jour les probabilités.

- Les transformées de Hough généralisées ont été développées pour apparier des formes à partir de leurs contours.

- Les méthodes stochastiques utilisées pour modéliser la carte des disparités à l'aide de modèles aléatoires, ainsi que des formulations bayesiennes pour les correspondances [Mat92], ou encore pour l'analyse des erreurs [RA90].

- La programmation linéaire est employée pour résoudre des systèmes d'inégalités linéaires, et permet de trouver la correspondance entre points ainsi que la zone d'erreur possible décrite par un polygone autour du point.

- La programmation dynamique [OK85a, OK85b, OTI87, LHB87, IB94], est un processus qui permet de résoudre un problème en utilisant les solutions de sous-problèmes. Le problème initial est résolu en utilisant les meilleures solutions aux sous-problèmes. On évite donc des calculs redondants mais cette méthode ne peut être appliquée que lorsqu'il y a une relation

d'ordre intrinsèque aux données (par exemple lorsque l'on veut faire une corrélation pixel à pixel, ceux-ci sont ordonnés).

- La recherche arborescente est une technique de parcours en profondeur d'un arbre décrit par les primitives des deux images. il s'agit d'assigner entre elles les primitives des deux images tant que l'assignation reste compatible avec celles déjà réalisées avant (relation binaire ou plus), sinon on rebrousse chemin, c.-à-d., on remonte dans l'arbre (*backtrack*). Le problème de cette technique est qu'elle peut être très coûteuse suivant le nombre de primitives à traiter.

- La comparaison de graphes [BL84], revient à trouver la clique maximale dans un graphe. Une primitive de la première image et une de la deuxième image forment un noeud du graphe si elle satisfont les contraintes d'appariement. Ensuite, deux noeuds sont connectés si ils sont compatibles par rapport aux contraintes binaires. La solution finale est le plus grand ensemble de noeuds connectés, soit la clique maximale. Là aussi, le problème du temps de calcul interdit l'utilisation de primitives trop nombreuses telles que les pixels de l'image.

- Les techniques structurelles utilisent aussi des graphes. Les primitives dans une image sont connectées entre elles par des relations binaires ou même ternaires (distances, angles, ...). Ainsi, pour chaque image on obtient un graphe et l'appariement consiste à retrouver le plus grand sous-graphe commun. Toutefois, cela n'est pas souvent possible à cause des bruits et déformations du signal, on utilise alors un appariement approché en déterminant un sous-graphe compatible.

- Il existe enfin, des approches pour la recherche basées sur des heuristiques particulières.

Finalement, en parcourant les quatre composantes nécessaires à un processus d'appariement, on se rend vite compte que ceux-ci sont étroitement liés les uns aux autres, d'une part, et d'autre part, les choix à faire pour chacun dépendent directement du problème posé. Nous n'avons montré ici que les techniques de base mais il existe un grand nombres de techniques issues de celles-ci avec certaines modifications apportées pour améliorer ou accélérer les calculs. Citons par exemple, le cas des méthodes de corrélation partielle utiles pour traiter le problème des occlusions (éléments cachés dans une des deux images). Dans [LM95], les auteurs utilisent un estimateur robuste pour trouver la bonne partie à corréler et font ensuite les calculs sur cette partie. De même, dans [GL94], des fenêtres de tailles adaptatives sont utilisées pour mieux tenir compte des différences de répartition spatiale des textures dans les images.

Enfin, une dernière façon d'améliorer la mise en correspondance est d'utiliser conjointement plusieurs techniques complémentaires. Plusieurs études ont été menées dans ce domaine de manière à obtenir des corrélateurs plus fiables et plus précis ainsi qu'à accélérer les calculs quand cela est possible. Dans le paragraphe suivant, nous décrivons rapidement quelques algorithmes coopératifs de mise en correspondance.

1.2.3 Algorithmes coopératifs

Lorsque l'on veut obtenir des cartes denses de disparités, on cherche a obtenir le plus grand nombre possible de couples appariés valides. Or, la validité est souvent contradictoire avec le nombre. C'est pour cela qu'un bon nombre d'études ont été menées dans le domaine des algorithmes coopératifs.

En fait, l'idée de base se fonde encore une fois sur le système visuel de l'homme pour lequel il est établi qu'il utilise plusieurs techniques différentes pour appréhender une scène. Le principe est donc d'utiliser les informations obtenues depuis une technique de base pour guider le travail d'une autre technique dans le but de compléter et d'affiner les résultats. On s'aperçoit avec cette définition, que l'on peut ainsi avoir un grand nombre de méthodes coopératives possibles.

Parmi ces méthodes, on peut citer par exemple les travaux de Hoff et Ahuja [HA89] où les auteurs utilisent conjointement la mise en correspondance d'éléments caractéristiques et la détection de contours. Okutomi et Kanade [OK93] utilisent quant à eux la méthode déjà citée où plusieurs images sont prises avec des déplacements latéraux, et les résultats de chaque couple sont fusionnés pour réduire les ambiguïtés. Li et Hu ont mis au point un algorithme en deux passes pour résoudre les problèmes de discontinuités et d'occlusions en se basant sur le gradient de disparité [LH96]. Dans le même genre, Olsen [Ols90] utilise aussi le gradient de disparité ainsi qu'une approche multirésolution avec des pyramides d'images Laplaciennes et Gaussiennes. La multirésolution est aussi utilisée par Kumar et Desai [KD94] en combinaison avec l'extraction de segments, la corrélation et l'interpolation dans un algorithme itératif. Dans le domaine itératif, on citera encore les travaux de Pankanti et Jain [PJ95] qui proposent un système intégrant multirésolution, organisation perceptuelle et SFS. Concernant les méthodes de SFS et SFT, l'approche de Choe et Kashyap [CK91] déjà abordée précédemment utilise les deux méthodes à la fois. Ahuja et al. combine l'intensité des pixels avec les contours et certaines contraintes telles que la régularité à l'intérieur de chaque région dans [WAH92]. Et Hsieh et al. [HMP92] utilisent une méthode régionale multirésolution en coopération avec une méthode d'extraction d'indices basée sur les intensités et les gradients le long des lignes épipolaires pour traiter des zones urbaines où le problème de la précision est très important.

Enfin, on ne peut omettre les techniques neuronales aussi appelés schémas coopératifs. Ces méthodes intègrent généralement une mesure d'erreur basée sur les deux contraintes d'unicité et de continuité et qui est minimisée dynamiquement par un réseau de neurones. Les différentes méthodes de ce domaine varient principalement selon l'organisation interne et le fonctionnement du réseau. Citons par exemple [MP76, JKMC94] et plus récemment [Hen97].

Dans le cadre de ma thèse, l'enjeu est de calculer le relief d'un terrain à partir de deux images issues du satellite SPOT. On va donc s'intéresser maintenant, à ce problème particulier du calcul de MNT, qui revient à calculer des cartes denses de profondeurs.

1.2.4 Algorithmes pour la reconstruction de terrains

Pour résoudre ce problème, on retrouve en fait un sous-ensemble des techniques énoncées précédemment. Dans la plupart des cas, les approches de mise en correspondance par régions sont utilisées. En effet, les approches basées sur la mise en correspondance d'indices géométriques (segments, courbes, ...) couramment utilisées en robotique produisent rapidement (temps réel) des cartes de profondeurs peu denses généralement suffisantes pour permettre à des robots de naviguer dans un environnement donné tout en évitant les obstacles potentiels. On ne reviendra pas sur les techniques déjà abordées, mais nous donnons ici quelques exemples de techniques développées plus particulièrement pour la construction de MNT.

Le premier exemple est l'algorithme assez connu de Otto et Chau [OC89] (*Region growing*) conçu pour travailler sur des images SPOT. Le principe est de partir d'un ensemble de points correspondants dont la validité est assurée, puis on prédit les correspondances dans le voisinage de ces points. Ces correspondances sont ensuite raffinées par l'algorithme adaptatif de corrélation aux moindres carrés de Gruen [Gru85] (minimisation de la somme des carrés des différences entre les deux fenêtres), et le processus est répété jusqu'à ce que tous les points soient mis en correspondance. Une évaluation intéressante de cet algorithme est donnée dans [DM89] tout en le comparant à celui de Barnard et Thompson [BT80] et à l'algorithme PMF de Pollard et al [PMF85] basé sur la mise en correspondance de segments. Toutefois, le problème de cette méthode est de *suivre des chemins*, or sur une topographie donnée, tous les chemins ne sont pas équivalents et des aberrations peuvent apparaître par accumulation d'erreurs. Ce problème est aussi connu dans la réalité où des cartographes en mesurant par géodésie des boucles ne se retrouvent pas à la même altitude sur leur point de départ après avoir parcouru quelques dizaines de kilomètres. De plus, le franchissement de falaises (et de tout autre anomalie topographique) est impossible sans idée a priori au delà de la région étudiée.

On trouve aussi des algorithmes mis au point pour d'autres engins de vision tels que la sonde Viking envoyée pour explorer Mars. Ainsi, Day et al. [DCM92] reprennent l'algorithme de Otto et Chau en le combinant à un algorithme de SFS.

Concernant les engins roulants, les études telles que celle de Faugeras et al. [FHM+93] montrent que l'on peut reconstruire un MNT en fusionnant les différentes reconstructions tridimensionnelles obtenues à l'aide du système visuel d'un robot le long de son déplacement sur un terrain.

Finalement, une des méthodes la plus employée reste celle de la corrélation de fenêtres d'images éventuellement combinée avec d'autres approches comme la multirésolution et la programmation dynamique comme c'est le cas des travaux de thèse de Memier [Mem91] portant sur des images SPOT. C'est cet algorithme qui a servi de base de départ à mes travaux de thèse. Une description détaillée est donnée dans le chapitre suivant.

Ces travaux ont aussi été repris par Jacquis et al. dans [Jac93, JKMC94] où les auteurs proposent notamment des extensions sur les contraintes de mise en correspondance utilisées par Memier telle que la cohérence entre lignes épipolaires (aussi utilisée par Ohta et Kanade [OK85a]). Ils présentent aussi d'autres approches dont une itérative du même style que celle de Otto et Chau et une basée

sur un réseau de neurones.

1.3 Vision avec plus de deux images

Dans le cadre de la vision multi-images, la géométrie du système de vision a une influence importante sur la méthode utilisée et réciproquement.

Une méthode courante est celle utilisant la variation de la distance entre les deux caméras du système [OK93, KS96] par déplacement latéral. De cette distance appelée ligne de base (*baseline*), dépendent les disparités qui vont apparaître entre les deux images. Si l'on dispose de plusieurs couples stéréo avec des lignes de base de différentes longueurs, on peut fusionner les informations obtenues dans chaque couple pour éliminer les ambiguïtés de correspondance et avoir un résultat final plus robuste et précis. Il est à noter que les disparités estimées sont plus précises lorsque la ligne de base est grande, mais d'un autre côté, cela implique de grande disparités donc un espace de recherche plus important et une probabilité d'erreur accrue. Il faut donc bien faire attention à ne pas dépasser un certain seuil et donc faire un compromis entre précision et densité des résultats. Des techniques plus générales que celle-ci travaillent sur au moins 3 caméras et ne se limitent pas à des déplacement latéraux. En vision trinoculaire, on trouve par exemple des positionnements en triangle rectangle [FHM+93].

Enfin, on peut aussi évoquer les techniques basées sur les flots d'images (*Structure from motion*) [LB95]. Le principe est de retrouver la structure 3D de manière itérative à partir d'une séquence d'images en se basant sur des équations de contrainte de mouvement.

Nous pourrions encore trouver, sans aucun doute, beaucoup d'autres techniques qui auraient leur place dans cette section. Cependant, comme nous l'avons indiqué précédemment, l'important ici est de donner un aperçu des méthodes les plus générales. Or, beaucoup des techniques existantes ne sont développées que pour des applications particulières et mettent souvent en jeu des variantes des approches vues ici.

Ainsi se termine notre revue sur les principaux algorithmes séquentiels de stéréovision et plus particulièrement de reconstruction de terrains. Les diverses approches mentionnées ici tentent de produire des résultats précis et denses mais un des gros problèmes qui reste à traiter est celui des temps de calculs. En effet, les méthodes de corrélation utilisées pour obtenir des cartes denses de disparités requièrent des temps de calcul souvent très élevés empêchant ainsi le traitement de grandes images telles que celles produites par SPOT avec une taille 6000×6000. Il y a bien eu des développements de cartes dédiées implantant un algorithme donné (par exemple [FHM+93]), mais sans remettre en cause l'intérêt d'une implantation électronique, nous nous plaçons ici dans un cadre plus général où l'idée est de concevoir une chaîne de traitement des images satellites allant de la reconstruction tridimensionnelle de terrains à la visualisation texturée de ceux-ci avec éventuellement la possibilité de calculs géologiques en temps interactif voire en temps réel, et qui soit utilisable sur plusieurs types de machines. Ainsi, le parallélisme nous paraît tout à fait indiqué pour traiter de grands ensembles de données dans des temps réduits, d'autant plus qu'il n'est a priori pas limité en puissance comme le seraient les architectures dédiées.

1.4 Approches parallèles

Notre but ici n'est pas de présenter le parallélisme en général mais simplement d'exposer différentes approches parallèles utilisées pour résoudre le problème de la stéréovision. Nous rappelons simplement qu'une machine parallèle regroupe plusieurs processeurs indépendants pouvant échanger des données soit par l'intermédiaire d'un réseau soit en ayant tous accès à une mémoire partagée. Les principaux schémas algorithmiques sont le SIMD (*Simple Instruction Multiple Data*) et le MIMD (*Multiple Instruction Multiple Data*). Il existe d'autres variantes de plus haut niveau telle que le SPMD (*Single Program Multiple Data*) ou encore l'approche en pipeline. Dans l'approche SIMD, tous les processeurs exécutent en même temps la même instruction mais sur des données différentes. Cette approche est aussi appelée parallélisme à grain fin. En MIMD, à un instant donné les processeurs exécutent des instructions potentiellement différentes sur des données différentes. En SPMD, un même programme est exécuté sur chaque processeur et toujours avec des données différentes. Cette approche est appelée parallélisme à gros grain. Enfin, l'approche du pipeline consiste à découper le traitement des données en sous-traitements mis en séquence. Ainsi, pour un élément de la chaîne, dès qu'une donnée a été traitée, le résultat est envoyé en entrée de l'élément suivant et une nouvelle donnée est récupérée en entrée. Cela génère un parallélisme implicite puisque plusieurs données se trouvent dans le pipeline à un instant précis.

Justement, concernant les approches pipeline, elles ne sont en fait pas très adaptées au problème de stéréovision car la chaîne va au mieux aussi vite que la partie la plus lente ce qui requiert des synchronisations et donc des latences. De plus, la division des tâches n'est pas facile et implique beaucoup de contraintes.

L'approche SIMD semble plus prometteuse. Le principe revient souvent à utiliser un processeur par pixel. On a donc une grille 2D de processeurs. Le problème majeur de ce genre d'algorithmes est qu'ils demandent beaucoup de processeurs et qu'ils ne sont pas très flexibles. Les processeurs dans ces machines sont souvent peu puissants en calculs flottants et n'ont pas beaucoup de mémoire. De plus, la grille 2D peut être trop petite pour certaines images. On peut alors envisager de découper les images initiales en sous-images mais cela pose des problèmes de continuité et de recouvrements des résultats. Enfin, l'efficacité est parfois réduite car pour certaines opérations, seuls certains pixels (processeurs) sont actifs. Parmi les études menées sur cette approche, on peut citer par exemple [LR91, HZM91, Chu96].

Concernant les approches MIMD ou SPMD, on trouvera plusieurs études telles que [FHM+93, WR94, CSWW97]. Dans ces algorithmes, l'espace de recherche est distribué sur les processeurs. Selon la nature de cet espace, les données sont elles aussi distribuées comme dans [FHM+93] où les images sont découpées en bandes horizontales et chaque processeur réalise une corrélation sur la bande qu'il possède.

Finalement, les approches neuronales peuvent aussi apparaître dans ce paragraphe puisqu'elles sont intrinsèquement parallèles.

Malgré l'intérêt du parallélisme pour résoudre le problème de la stéréovision, on remarque qu'il y a finalement assez peu d'études qui ont été menées dans cette optique en rapport à celles

réalisées dans le modèle séquentiel. Cela est sans doute du à la relative jeunesse du parallélisme ainsi qu'au nombre de machines parallèles encore très faible et à leur coût très élevé. Le développement de bibliothèques parallèles telles que PVM (*Parallel Virtual Machine*) et MPI (*Message Passing Interface*) permettant l'utilisation de réseaux de stations en tant que machine parallèle devrait faciliter l'accès au parallélisme au plus grand nombre de laboratoires.

1.5 Conclusion

En conclusion, on peut constater que la littérature sur le sujet est très importante et que cette abondance rend difficile toute revue et classification des méthodes existantes. On retiendra finalement la composition de tout algorithme de stéréovision en quatre parties : les éléments à apparier, la mesure de similarité, l'espace de recherche et la stratégie de recherche. De même, on peut noter la possibilité de combiner plusieurs techniques de base pour obtenir une mise en correspondance plus robuste et précise. Enfin, en parcourant les algorithmes dédiés à la reconstruction des terrains, on s'aperçoit que les problèmes principaux (outre ceux inhérents à la stéréovision) résident dans la taille des images que l'on doit traiter ainsi que dans les temps de calculs qui sont souvent un obstacle pour une exploitation réelle. Pour cette raison, des approches parallèles tentant d'exploiter ce formidable potentiel de calcul ont vu le jour avec plus ou moins de réussite. C'est pourquoi, nous avons voulu concevoir un algorithme parallèle de reconstruction de terrains fiable, précis, rapide et pouvant traiter des images entières (6000×6000) venant du satellite SPOT. Pour cela, nous nous sommes basés sur l'algorithme de Memier. D'une part, parce qu'une version opérationnelle de celui-ci était déjà utilisée par les géologues avec lesquels nous avons collaboré et surtout parce que ces mêmes géologues étaient très satisfaits de la précision des résultats obtenus par cet algorithme. Dans le chapitre suivant, nous en donnons une description détaillée.

CHAPITRE 2

Présentation de l'algorithme de Memier

Le but de ce chapitre est de présenter notre base de travail sur la vision stéréoscopique. Comme il a été mentionné précédemment, tout au long de cette thèse, nous avons collaboré avec le département des Sciences de la Terre de l'ENS Lyon. En effet, ce département détenait déjà un programme séquentiel de reconstruction de terrains, développé par Michel Memier dans le cadre de sa thèse. Ce programme séquentiel a donc servi de point de départ pour la conception d'un algorithme parallèle.

Ce second chapitre présente les différentes parties de l'algorithme de reconstruction tridimensionnelle, à partir d'un couple d'images stéréoscopiques, développé par Memier. Le lecteur peut se référer à [Mem91] pour une description plus détaillée.

2.1 Schéma algorithmique

Si l'on reprend la description d'un algorithme de mise en correspondance donnée dans le chapitre précédent, la méthode de Memier peut être définie par les quatre composantes de base : éléments à apparier, mesure de similarité, espace et stratégie de recherche.

2.1.1 Primitives à apparier

Puisque l'on veut obtenir une carte des profondeurs la plus dense et précise possible, les primitives qui semblent les plus adaptées sont les pixels. On s'aperçoit tout de suite de l'ampleur de la tâche impliquée par ce choix car de tous les types de primitives couramment utilisés, c'est sans doute le pixel qui génère le plus grand nombre de primitives à apparier. En particulier, les images SPOT brutes que l'on veut traiter ont une taille d'environ 6000×6000 pixels.

Pour apparier les pixels, il nous faut définir une mesure de similarité qui va nous permettre de les comparer.

2.1.2 Mesure de similarité

L'algorithme de Memier repose sur une corrélation basée sur la texture ou encore une corrélation de régions. Autrement dit, la technique consiste à comparer des régions d'images, correspondant ici aux fenêtres englobantes des pixels, plutôt que simplement les intensités de ceux-ci. Le principe est donc le suivant :

- prendre un point et sa fenêtre englobante dans l'image de référence,
- rechercher le point correspondant parmi les candidats possibles de l'autre image à l'aide de leurs fenêtres englobantes.

Pour trouver le correspondant, on utilise le coefficient de similarité (aussi appelé coefficient de corrélation), qui est calculé pour chaque couple de fenêtres entre le point gauche et les points droits. Cette mesure de similarité déjà définie dans le paragraphe 1.2.2, utilise les variances et covariances des fenêtres. Une fois les mesures calculées pour tous les candidats, il nous reste à trouver la meilleure correspondance.

2.1.3 Espace de recherche

Comme nous l'avons évoqué dans le chapitre précédent, une des conventions souvent utilisée en vision binoculaire est de prendre une des deux images comme image de référence puis de faire la mise en correspondance, c'est à dire la recherche des éléments homologues (les pixels ici), dans l'autre image. Dans notre cas, nous fixons l'image gauche comme étant la référence et nous notons que ce choix n'implique aucune perte de généralité sur l'algorithme.

De plus, étant donné que nous nous focalisons sur la partie de mise en correspondance, nous considérons que les images en entrées sont déjà redressées en géométrie épipolaire. Cette caractéristique est très importante car elle implique que pour chaque point d'une image, son homologue dans l'autre image se trouve sur une ligne donnée (ligne épipolaire). Si l'on redresse les images de manière à faire coïncider les lignes épipolaires avec les lignes horizontales des images tout en alignant les épipolaires correspondantes des deux images, on saura que le point homologue ne peut se trouver que sur la même ligne dans la seconde image. On constate donc que le domaine de recherche est ainsi considérablement réduit. On notera que la géométrie épipolaire n'existe pas réellement pour le capteur en ligne du satellite SPOT. Néanmoins, Memier utilise un redressement épipolaire approché en tenant compte de la géométrie du capteur et obtient des résultats très satisfaisant nous permettant de considérer ces images comme valides (du point de vue de la propriété épipolaire).

Donc, pour un point donné $I(x, y)$ dans l'image de référence, nous pourrions simplement balayer tous les pixels de la ligne y de l'autre image et mesurer pour chacun d'eux leur coefficient de similarité par rapport au pixel de référence, et finalement choisir celui de mesure maximale. Néanmoins, cela serait encore très et trop coûteux en temps de calcul et on sait que l'on peut réduire encore le domaine de recherche au sein d'une ligne jusqu'à quelques pixels autour du point $I'(x, y)$

dans la seconde image. On utilise pour cela la disparité maximale possible entre deux points correspondants. Cette disparité aussi appelée parallaxe maximale peut être estimée par rapport aux positions relatives des deux caméras, leur distance au sol et le dénivelé maximal observé sur la région traitée. La figure 2.1 décrit cette propriété géométrique. La parallaxe maximale donnant le

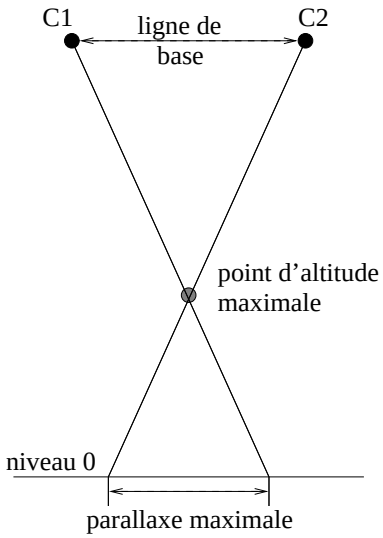

FIG. 2.1 - *Disparité ou parallaxe maximale par rapport à la géométrie du système*

nombre de candidats possibles pour un point donné, elle est aussi associée à la plage de recherche.

L'espace de recherche est donc défini pour chaque point de référence par la plage de recherche autour du point de mêmes coordonnées dans l'autre image. Cela implique que les points se trouvant dans une zone proche des bords des images ne pourront être traités. En effet, si la hauteur des fenêtres de corrélation est N, comme la fenêtre de corrélation doit être complètement disponible pour chaque point traité, les $\lfloor \frac{N}{2} \rfloor$ premières et dernières lignes et colonnes des images ne peuvent être corrélées. Enfin, concernant l'image de référence, si la plage de recherche est égale à P, pour que le calcul soit correct, il faut que les P candidats de chaque point de référence soient disponibles, donc que les fenêtres des $\lfloor \frac{P}{2} \rfloor$ points de part et d'autre du point de référence soient complètement disponibles. Ainsi, les $\lfloor \frac{N}{2} \rfloor + \lfloor \frac{P}{2} \rfloor$ premiers et derniers pixels de chaque ligne de l'image de référence ne peuvent être appariés.

Enfin, une fois que l'on a les mesures de similarité entre le point de référence considéré et ses candidats possibles, il ne reste plus qu'à trouver le bon candidat sur la courbe décrite par ces mesures. Le candidat choisi devrait logiquement être celui dont le voisinage (c.-à-d. la fenêtre) est le plus ressemblant à celui du point de référence. Cela se traduit par un maximum global sur la courbe des similarités. Néanmoins, celui-ci peut être plus ou moins bien marqué selon le contraste dans la zone considérée. La figure 2.2 montre quelques exemples de courbes selon l'organisation spatiale de la texture dans la zone de corrélation. On voit que plusieurs types de courbes sont possibles. Si la texture contient un motif répétitif, on aura alors plusieurs pics de corrélation. Si il y a de fortes disparités (radiométriques ou géométriques), la courbe sera lisse et le pic sera peu prononcé rendant difficile la localisation du maximum. Enfin, si la région est peu contrastée, on aura

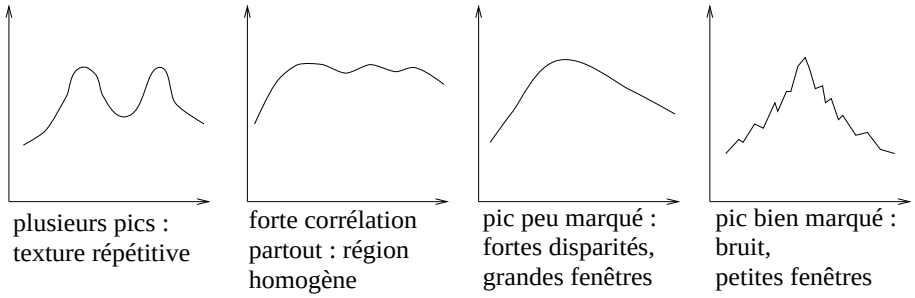

FIG. 2.2 - *Exemples de courbes de similarités en fonction de la région de corrélation*

une corrélation quasi constante sur tout l'espace de recherche. De la même manière, la taille de la fenêtre de corrélation modifie l'aspect de ces courbes. des fenêtres de petite taille donnent des pics assez bien prononcés mais bruités. Elles sont assez bien adaptées pour des variations locales assez fortes (falaises, ...). Par contre, les fenêtres plus importantes donnent des courbes moins sensibles au bruit mais avec un pic de corrélation plus aplati. Il est donc important de bien choisir la taille de la fenêtre de corrélation de façon à obtenir des pics de corrélation facilement exploitables sur l'ensemble de l'image.

La dernière composante de l'algorithme restant à définir est la stratégie de recherche des correspondances, même si l'espace de recherche défini ici la conditionne fortement. Nous montrons dans le paragraphe la technique utilisée par Memier.

2.1.4 Stratégie de recherche

En fait, la première idée qui vient à l'esprit pour trouver les correspondants est celle indiquée précédemment qui consiste à trouver le maximum global dans chaque courbe de similarités. Néanmoins, on a aussi vu les problèmes inhérents à ces courbes et donc les erreurs éventuelles impliquées par cette méthode. Pour pallier à ce problème de maximum global peu marqué, Memier repère tous les maxima locaux présents sur la courbe et utilise une pondération de leur mesure de similarité par la hauteur du pic prise sur un petit intervalle constant autour du sommet. Ainsi, le choix final ne se fera pas forcément sur le maximum global mais sur le maximum local ayant une similarité assez forte et un pic de corrélation mieux marqué réduisant ainsi la probabilité d'erreur sur l'appariement.

De plus, pour obtenir des appariements encore plus homogènes sur l'ensemble d'une ligne épipolaire, Memier utilise une contrainte supplémentaire sur les mises en correspondance, prenant en compte la cohérence entre les appariements voisins. Comme on l'a vu dans le chapitre précédent, il existe plusieurs contraintes de consistance qui permettent d'améliorer la corrélation à un plus haut niveau, en utilisant une mise en correspondance globale. Une de ces contraintes utilisée par Memier est la cohérence intra-ligne. C'est une contrainte d'ordre sur une ligne épipolaire donnée : Si $P_1(x, y)$ et $P'_1(x', y)$ sont des points correspondants dans les deux images I et I', alors la contrainte d'ordre nous dit que pour tout point $P_2(t, y)$ avec $t > x$, son homologue dans I' doit être de la forme $P'_2(t', y)$ avec $t' \geq x'$ comme indiqué dans la figure 2.3. Cela revient à dire que l'ordre des points doit être conservé entre les deux images. Comme on l'a vu dans le chapitre 1, cette contrainte n'est

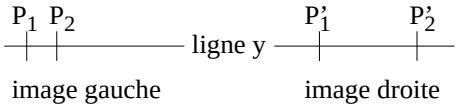

FIG. 2.3 - *Contrainte d'ordre intra-ligne*

malheureusement pas toujours vraie, en particulier lorsque l'on se trouve confronté à des structures verticales et étroites telles que des bâtiments. Néanmoins, elle reste bien adaptée à des reliefs naturels observés d'assez loin comme c'est le cas avec les images satellites. Finalement, plutôt que de trouver des couples de points indépendemment les uns des autres, Memier a opté pour une mise en correspondance tenant compte non seulement des mesures de similarité de chaque couple mais aussi du contexte global de la ligne épipolaire toute entière. Cela est réalisé en recourant à la programmation dynamique. Le principe est de trouver un chemin de coût minimum dans un graphe consistant en une grille 2D reliant entre eux tous les points appariables (pouvant être traités) des deux lignes épipolaires considérées. On peut noter que la parallaxe maximale conditionne assez fortement le chemin en le canalisant dans une bande précise du graphe orientée diagonalement (région grisée). La figure 2.4 montre ce graphe ainsi qu'un exemple schématique de chemin. Le calcul des coûts des

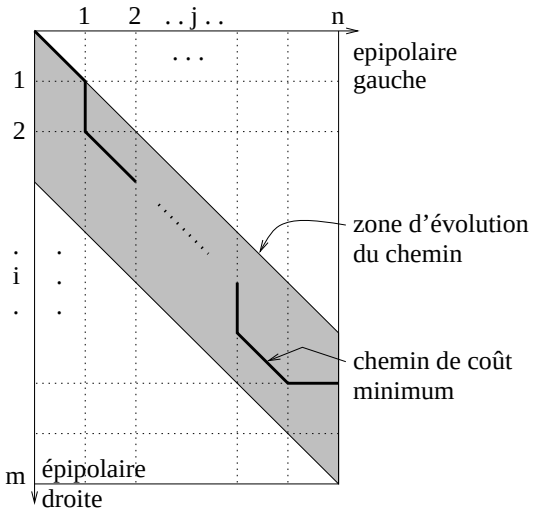

FIG. 2.4 - *Graphe 2D et chemin de coût minimum pour la mise en correspondance globale des lignes épipolaires*

chemins se fait donc colonne par colonne. Pour calculer les prolongements et les coûts des chemins possibles à la colonne j, on considère chaque point du domaine de recherche de la colonne j et les chemins déjà calculés jusqu'à la colonne j-1. On relie chaque point de la colonne j au chemin de coût minimum parmi ceux pouvant être prolongés par ce point sans violer la contrainte d'ordre. Cette contrainte d'ordre, citée plus haut, interdit de remonter dans le graphe, ce qui signifie qu'un chemin arrivant au point (i,j-1) ne pourra se prolonger en (i-k,j) avec k>0. La figure 2.5 indique l'ensemble des points pouvant prolonger un chemin donné tout en respectant cette contrainte. Le coût du nouveau chemin est obtenu simplement en additionnant le coût de l'ancien chemin et celui du point ajouté. Enfin, on stocke pour chaque point du graphe traversé par un chemin, le numéro

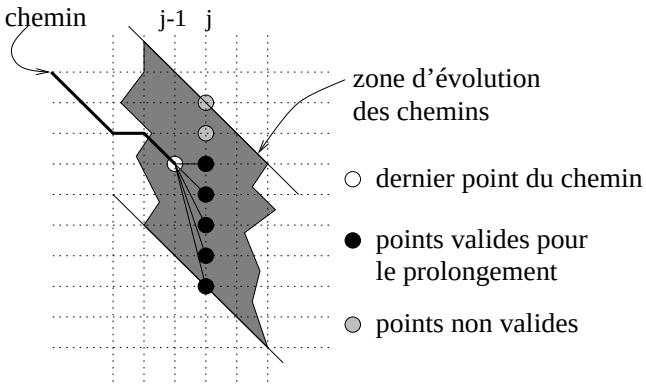

FIG. 2.5 - *Ensemble des points pouvant prolonger un chemin donné en respectant la contrainte d'ordre*

de ligne de son antécédent dans ce chemin. Ainsi, lorsque les coûts des différents chemins sont calculés dans la bande grise, il suffit de partir du coût minimum dans la dernière colonne du graphe puis de remonter en suivant les antécédents correspondants. De plus, Memier obtient une précision de disparité supérieure au pixel en affinant chaque point (appariement) du chemin minimal. Pour cela, il assimile le pic de corrélation au voisinage du point apparié à une parabole s'appuyant sur les similarités de ce point et de ses deux voisins, puis il calcule l'abscisse de dérivée nulle, comme l'indique la figure 2.6. S'il n'en trouve pas, il reste en précision entière. Malgré la réduction de la

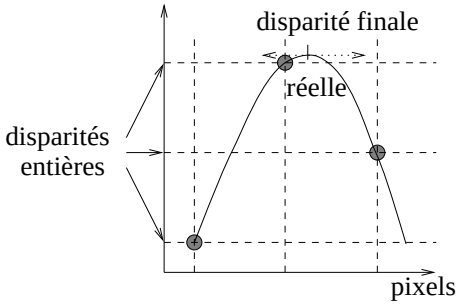

FIG. 2.6 - *Utilisation d'une parabole pour obtenir les disparités avec une précision supérieure au pixel*

plage de recherche décrite précédemment, l'algorithme obtenu reste très gourmand en temps de calculs puisque sa complexité est en $O(H.V.P.m.n)$ où H est la taille horizontale des images, V la taille verticale, P le nombre de candidats possibles pour un point de référence et $m.n$ est la taille des fenêtres englobantes (m colonnes et n lignes). Par conséquent et pour économiser du temps, Memier utilise un algorithme de corrélation en deux passes basé sur la multirésolution. Nous décrivons cette méthode dans la partie suivante.

2.2 Corrélation en deux passes

Une première corrélation est effectuée sur des images réduites en utilisant la programmation dynamique. Cela nous permet d'avoir une première approximation des disparités entre les points homologues des images gauche et droite. Les disparités obtenues sont valides à un coefficient multiplicateur près qui n'est autre que le facteur de réduction des images. Or, comme on sait que les disparités sont bornées par la parallaxe maximale lorsque l'on est à la résolution initiale, on en déduit que ces approximations doivent être conditionnées par un seuil égal au rapport de la parallaxe maximale sur le facteur de réduction.

Ainsi, en rééchantillonnant les disparités obtenues puis en les rééchelonnant par rapport au facteur de réduction, on peut retrouver des approximations de disparités en haute résolution. Pour cela, Memier utilise des splines cubiques naturelles. Enfin, une deuxième corrélation en haute résolution est alors nécessaire pour raffiner ces résultats. Néanmoins, puisque l'on sait que la disparité finale sera dans un voisinage proche (quelques pixels) de l'approximation, on peut donc réduire considérablement le domaine de recherche pour cette deuxième corrélation, d'où un gain de temps non négligeable. Pour réaliser cette deuxième corrélation, les pixels de l'image de droite sont déplacés par rapport à leur disparité estimée pour obtenir une image plus proche de l'image gauche et donc utiliser une plage de recherche plus petite.

De plus, une petite plage de recherche rend la probabilité de violation de la contrainte intra-ligne plus faible. C'est pourquoi la programmation dynamique n'est plus nécessaire à ce niveau et comme celle-ci est efficace mais assez gourmande en mémoire, il paraît logique de faire la seconde corrélation de manière indépendante pour chaque points de référence.

Finalement, nous obtenons une carte des disparités en combinant (ajoutant) les disparités approchées de la première corrélation et les disparités réduites de la seconde. Comme cette étape de calcul est très simple et rapide, elle a été intégrée au sein même de la deuxième corrélation. On en déduit une liste de points correspondants qui peuvent être reconstruits dans une base tridimensionnelle par rapport à la géométrie des caméras en tenant compte du modèle de déformation utilisé pour recaler les images en géométrie épipolaire. Les couples de points dont la disparité finale dépasse la parallaxe maximale sont invalidés. Il est à noter que dans notre cas, il est impératif de pouvoir recaler les points 3D dans un repère lié à la Terre. Il nous faudra donc connaître parfaitement et de manière précise quelques couples de points correspondants dans les deux images ainsi que leurs coordonnées 3D dans ce repère. Ces points sont aussi appelés points d'amer.

L'étape finale consiste, à partir des points 3D reconstruits, à calculer le MNT proprement dit, c'est à dire une grille régulière de points dont les altitudes sont connues.

2.3 Calcul du MNT

À partir de la carte des disparités, nous obtenons un ensemble de points 3D répartis de manière non régulière dans le plan 2D XY qui correspond à la vue par dessus de la scène 3D. La toute

dernière étape consiste à générer le Modèle Numérique de Terrain qui n'est autre qu'une grille régulière de points dont on connaît les altitudes. Un exemple schématique de MNT est montré dans la figure 2.7.

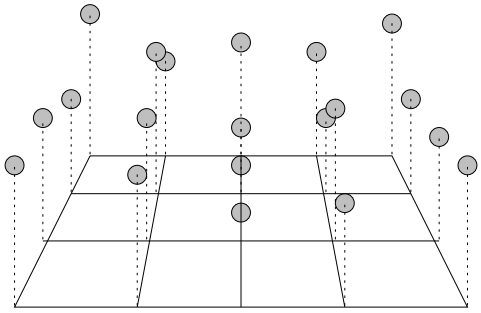

FIG. 2.7 - *Modèle Numérique de Terrain*

Pour obtenir cette structure, il suffit d'échantillonner nos points 3D sur une grille régulière 2D confondue avec le plan XY, en utilisant une taille donnée de cellules et un seuil permettant d'éliminer les points ayant une altitude trop divergente des autres points se trouvant dans la même cellule. Bien sûr, la taille des cellules conditionne la résolution et donc la précision du MNT final. De même, le seuil de divergence des altitudes influe sur la justesse et donc la qualité des résultats.

On peut d'ailleurs noter que cette dernière opération peut être appliquée sur un même ensemble de points 3D avec différents paramètres de taille de cellule et de seuil de divergence selon la résolution que l'on désir avoir.

2.4 Algorithme final

En rassemblant toutes les caractéristiques énoncées dans les paragraphes précédents, on obtient l'algorithme final utilisé par Memier pour traiter les images SPOT.

2.4.1 Algorithme

Pour rester le plus général possible, on considère que les deux étapes de corrélation peuvent être réalisées avec des fenêtres de tailles différentes, respectivement N_1 et N_2. Néanmoins, on utilise couramment des fenêtres carrées pour ne privilégier aucune direction particulière dans l'image.

L'algorithme est donc constitué des étapes suivantes :

- première corrélation sur les images réduites :
  - réduction des images épipolaires,
  - calcul des coefficients de similarité des candidats en utilisant la plage de recherche initiale et des fenêtres de taille N_1,
- mise en correspondance globale des lignes épipolaires en utilisant la programmation dynamique,
- interpolation des disparités approchées par des splines cubiques naturelles,
- rééchelonnement des disparités et repositionnement des pixels de l'image droite,
- deuxième corrélation sur les images haute résolution :
  - corrélation en utilisant l'image gauche initiale et l'image droite recalée, une plage de recherche réduite et des fenêtres de taille N_2,
  - calcul des disparités finales en additionnant les disparités approchées et résiduelles,
- reconstruction des points 3D,
- échantillonnage des points 3D sur la grille 2D du MNT.

Une fois que l'algorithme est conçu, il ne reste plus qu'à le programmer sur machine pour pouvoir le mettre en oeuvre.

2.4.2 Implantation

Le code de Memier a été réalisé étape par étape. C'est à dire que l'application dont nous disposons se présente sous la forme d'un ensemble de programmes écrits en langage Pascal. Chacun de ces programmes implantant une des étapes de l'algorithme donné dans le paragraphe 2.4.1, et prenant en entrée les données générées par l'étape précédente et stockées dans un ou plusieurs fichiers.

Les deux problèmes majeurs de cette application sont la taille mémoire qu'elle nécessite pour fonctionner et les temps de calculs. En effet, les géologues qui l'utilisent n'ont pu le faire fonctionner que sur une machine dotée de 128 Mo de mémoire vive pour traiter des images d'environ 2000 $\times$ 2000 pixels. Et pour ces mêmes images, il faut environ 5 à 6 heures de calculs sur une station SPARC serveur 1000, ce qui peut être un obstacle à une utilisation intensive. On peut aussi évoquer

la quantité de données produite, c'est à dire le nombre de points 3D, qui est parfois en deçà des prévisions, notamment à cause de zones difficiles à corréler car trop homogènes tels que des lacs. De même, Memier fait appel à des splines cubiques naturelles pour l'interpolation des disparités entre les deux corrélations. Cet outil d'interpolation n'est pas vraiment adapté à des données naturelles comme des altitudes ou des disparités. En effet, ces disparités entre points ne sont pas forcément régulières et peuvent même présenter des discontinuités en certains points particuliers. Par exemple, lorsqu'une occlusion apparaît entre les deux images. Or, les splines ont plutôt tendance à lisser les données. A ce niveau, il serait donc intéressant d'utiliser des fonctions d'interpolation locales plus flexibles telles que celles proposées par Catmull-Rom [CR74].

Dans la première partie du chapitre 3, nous revenons plus en détails sur ces différents problèmes tout en proposant différentes modifications.

2.5 Conclusion

Nous avons décrit l'algorithme de stéréovision de Memier. Celui-ci est divisé en plusieurs étapes et cela se retrouve au niveau du code de son application qui est constitué d'un ensemble de programmes distincts mettant en oeuvre ces différentes étapes.

Lorsque l'on compare l'algorithme de Memier aux différentes approches de reconstruction de terrains, on s'aperçoit que celui-ci est plutôt bien placé au niveau de la qualité. En effet, d'après l'avis des spécialistes géologues, les résultats d'un point de vue topologique sont plus précis que bon nombre d'autres approches. Le problème de la qualité des résultats est que l'on ne peut pas la vérifier réellement de manière informatique car les MNT produits ont une résolution bien supérieure à ceux qui sont disponibles dans le domaine publique ou commercial actuellement. On peut éventuellement avoir une estimation de la justesse en calculant des MNT à des résolutions plus faibles et en comparant avec ceux déjà existant. Néanmoins, toujours d'après l'avis des spécialistes géologues, ces MNT que l'on pourrait utiliser en référence ne sont pas forcément plus justes que ceux obtenus par l'algorithme de Memier. Donc, plutôt que de faire des comparaisons absolues avec des MNT plus ou moins fiables, on peut aussi évaluer la probabilité d'erreur pour chaque appariement en calculant par exemple l'écart moyen entre la similarité choisie et les autres dans le domaine de recherche. La probabilité d'erreur sera d'autant plus grande que l'écart sera petit.

L'approche de Memier intègre différentes techniques telles que la multirésolution, la programmation dynamique, et l'utilisation de contraintes géométriques et topographiques pour améliorer la robustesse du corrélateur tout en réduisant les temps de calculs. Néanmoins, il reste assez lent, les fonctions utilisées pour l'interpolation des disparités entre les deux corrélations ne sont pas vraiment adaptées, et il requiert une quantité de mémoire vive limitant considérablement le nombre de machines sur lesquelles il peut être utilisé.

Finalement, cette approche est un bon point de départ pour notre étude. Mais avant de se lancer dans sa parallélisation, il apparaît nécessaire d'y apporter quelques modifications pour optimiser les diverses phases de calculs afin de réduire les temps d'exécution séquentiels et tenter d'améliorer la qualité des résultats. Ces modifications sont explicitées en détails dans la première partie du

chapitre suivant. Ensuite, dans la deuxième partie du chapitre 3, nous étudions la mise au point d'un algorithme parallèle à partir de notre version séquentielle optimisée.

CHAPITRE 3

Optimisations séquentielles et Parallélisation

La description de l'algorithme de Memier donnée dans le chapitre 2, nous indique sa bonne qualité quant aux résultats fournis, mais aussi plusieurs défauts relativement importants. Il ne serait donc pas raisonnable de paralléliser directement cet algorithme tel quel. C'est pourquoi, la première phase de mon travail de thèse a consisté à étudier minutieusement chaque étape de l'algorithme pour dégager toutes les améliorations possibles sur chacune des caractéristiques importantes d'un algorithme. Après une première partie sur les notations utilisées tout au long de ce chapitre, nous présentons donc ces améliorations, et ensuite seulement, nous abordons dans une troisième partie la conception de l'algorithme parallèle à partir de notre nouvelle version séquentielle.

3.1 Notations

Par souci de clarté, nous fixons ici quelques notations importantes dont nous allons avoir besoin dans ce chapitre. Ces notations concernent principalement les différents paramètres de l'algorithme de reconstruction vu précédemment. Ainsi, la largeur et la hauteur des images épipolaires initiales seront notées respectivement L et H. Les fenêtres de corrélation étant carrées, une seule valeur n donnera la taille du côté dans un cadre général et nous reprenons les notations du chapitre précédent N_1 et N_2, pour les tailles effectivement utilisées dans les deux corrélations de l'algorithme. La taille de la parallaxe maximale, soit le nombre de candidats parcourus pour un pixel de référence sera notée P dans un cadre général, et P_1 et P_2 pour chacune des deux corrélations. Enfin, le coefficient de réduction des images lors de la première corrélation sera décrit par R.

3.2 Optimisations séquentielles

Il y a principalement quatre points qui peuvent être améliorer dans le code Pascal dont nous disposons :

- les temps de calcul,
- la qualité des résultats,

- la gestion mémoire,
- les entrées/sorties.

Les paragraphes suivants décrivent les améliorations apportées sur chacun de ces points. Toutefois, avant de plonger dans les tréfonds de l'algorithme, nous devons noter que bien que le code de Memier soit écrit en langage Pascal, nous avons opté pour le langage C, et ce pour deux raisons. La première est qu'il n'existe pas de bibliothèque parallèle pour le langage Pascal permettant le développement rapide et efficace d'un code multi-processeurs, alors qu'il y en existe plusieurs pour le C. La deuxième raison est que le langage C offre une gestion dynamique de la mémoire bien plus pratique et efficace que celle du Pascal, de même que pour les optimisations internes du code lors de la compilation.

3.2.1 Temps de calcul

Le défaut majeur de la technique de corrélation utilisée par Memier se trouve dans le recours à des calculs statistiques pour la mesure de similarité. En effet, ces calculs basés sur les variances et covariances des fenêtres de corrélation comme on l'a vu dans le paragraphe 1.2.2 sont très gourmands en temps. Et l'on se rend bien compte en considérant la taille des images à traiter (typiquement 2000×2000 et au delà) que l'algorithme en deux passes avec les images réduites (par des multiples de 2 en général) reste relativement lent.

Une première idée d'amélioration repose sur le fait que ces calculs de similarités sont réalisés indépendemment pour chaque fenêtre et que l'algorithme calcule ces similarités selon un ordre déterminé au sein d'une ligne épipolaire (de la gauche vers la droite), en utilisant une fenêtre de taille fixe. La figure 3.1 met en évidence le sens de parcours et les fenêtres de corrélation pour deux pixels voisins.

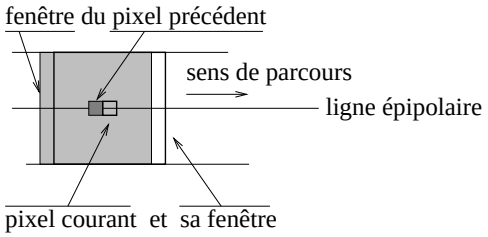

FIG. 3.1 - *Zones impliquées dans les calculs statistiques de deux pixels voisins avec une taille fixe de fenêtre*

Sur cette figure, on se rend bien compte que les zones de corrélation des deux pixels voisins se chevauchent. L'idéal serait alors de déduire les calculs statistiques du pixel courant à partir de ceux du pixel précédent. Or, si l'on regarde de plus près la formule du coefficient de similarité, on peut la reformuler dans un cadre général, de la manière suivante :

$$S = \frac{\sigma_{GD}^2}{\sqrt{\sigma_G^2 . \sigma_D^2}} \tag{3.1}$$

où σ^2_{GD} est la covariance des deux fenêtres F_G and F_D :

$$\begin{aligned}\sigma^2_{GD} &= \sum_{j=1}^{n}\sum_{i=1}^{n}\frac{(F_G(i,j)-\mu_G)(F_D(i,j)-\mu_D)}{n^2}\\ &= E((F_G - E(F_G))(F_D - E(F_D)))\end{aligned}$$

avec μ_k représentant l'intensité moyenne des pixels (c.-à-d. l'espérance mathématique $E(F_k)$) de la fenêtre k (Gauche ou Droite). Et σ^2_k représente la variance de la fenêtre k :

$$\begin{aligned}\sigma^2_k &= \sum_{j=1}^{n}\sum_{i=1}^{n}\frac{(F_k(i,j)-\mu_k)^2}{n^2}\\ &= E((F_k - E(F_k))^2)\end{aligned}$$

Or, on peut réécrire

$$\begin{aligned}\sigma^2_{GD} &= E((F_G - E(F_G))(F_D - E(F_D)))\\ &= E(F_G F_D - F_G E(F_D) - F_D E(F_G) + E(F_G)E(F_D))\end{aligned}$$

et comme $E(XE(Y)) = E(X)E(Y)$, on obtient

$$\begin{aligned}\sigma^2_{GD} &= E(F_G F_D) - E(F_G)E(F_D)\\ &= \frac{1}{n^4}\left[n^2\sum_{i,j}F_G(i,j)F_D(i,j) - \sum_{i,j}F_G(i,j)\sum_{i,j}F_D(i,j)\right]\end{aligned} \tag{3.2}$$

De même

$$\begin{aligned}\sigma^2_k &= E((F_k - E(F_k))^2)\\ &= E(F_k^2 - 2F_k E(F_k) + E(F_k)^2)\\ &= E(F_k^2) - E(F_k)^2\\ &= \frac{1}{n^4}\left[n^2\sum_{i,j}F_k(i,j)^2 - (\sum_{i,j}F_k(i,j))^2\right]\end{aligned} \tag{3.3}$$

Finalement, on s'aperçoit que les valeurs dont dépend notre coefficient de similarité sont :

$$S_{gd} = \sum_{i,j}F_G(i,j)F_D(i,j)\ , \tag{3.4}$$

$$S_g = \sum_{i,j}F_G(i,j)\ ,\ S_{g^2} = \sum_{i,j}F_G(i,j)^2\ , \tag{3.5}$$

$$S_d = \sum_{i,j}F_D(i,j)\ et\ S_{d^2} = \sum_{i,j}F_D(i,j)^2 \tag{3.6}$$

et en réintroduisant (3.2) et (3.3) dans (3.1) avec les appellations données dans (3.4), (3.5) et (3.6), puis en simplifiant par $\frac{1}{n^4}$, on arrive à

$$S = \frac{n^2 S_{gd} - S_g S_d}{\sqrt{(n^2 S_{g^2} - S_g^2)(n^2 S_{d^2} - S_d^2)}} \tag{3.7}$$

On constate aisément que toutes les sommes données en (3.4),(3.5) et (3.6) peuvent être calculées aussi bien en colonnes qu'en lignes ($\sum_{i,j}$ est équivalent à $\sum_{j,i}$). Comme les quatre dernières ne font intervenir qu'une seule image à la fois, il nous suffit de calculer indépendemment les sommes partielles sur chaque colonne (de hauteur n), puis d'additionner les n colonnes contiguës formant la fenêtre courante, comme indiqué sur la figure 3.2.

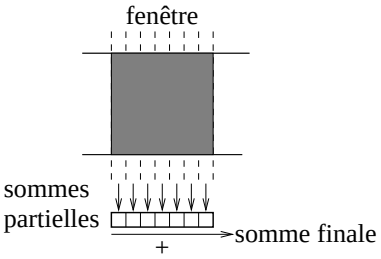

FIG. 3.2 - *Décomposition des sommes statistiques sur une fenêtre par leurs sommes partielles sur les colonnes*

Et comme on a vu dans la figure 3.1 que les $n-1$ *dernières* colonnes de la fenêtre associée à un pixel correspondent aux $n-1$ *premières* colonnes de la fenêtre du pixel suivant, on en déduit que l'on peut calculer les valeurs S_g, S_d, S_{g^2}, *et* S_{d^2} incrémentalement d'un pixel à son voisin. En effet, si l'on note C_j^i la $j^{\text{ème}}$ colonne ($0 \leq j \leq n$) de la fenêtre du $i^{\text{ème}}$ pixel de la ligne épipolaire, la relation qui lie les S_k (avec k représentant une des quatre sommes g, d, g^2 ou d^2) de deux pixels voisins est la suivante :

$$S_k^i = S_k^{i-1} - C_0^{i-1} + C_n^i \tag{3.8}$$

Et comme C_n^i représente la même colonne que C_{n+1}^{i-1}, on obtient une mise à jour incrémentale telle qu'indiquée dans le schéma 3.3.

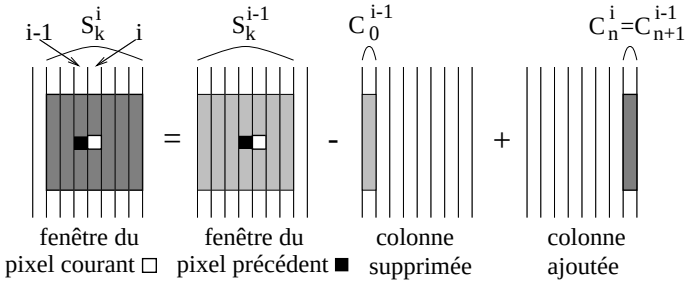

FIG. 3.3 - *Mise à jour incrémentale des valeurs S_g, S_d, S_{g^2} et S_{d^2} pour la fenêtre d'un pixel par rapport à celle du pixel précédent*

Enfin, les données à stocker pour calculer S_g, S_d, S_{g^2} et S_{d^2} sont les suivantes :

Concernant la fenêtre de référence, il nous suffit de conserver les valeurs S_g,S_{g^2} et leurs sommes partielles pour les n colonnes de la fenêtre. On utilise donc deux tableaux de $n+1$ cases, la case supplémentaire permettant de recevoir la somme de la nouvelle colonne lorsque l'on passe au pixel suivant. Une fois que les nouvelles valeurs statistiques S_g et S_{g^2} sont mises à jour, on fait une permutation circulaire des colonnes vers la gauche.

Concernant les P candidats, on doit stocker les valeurs de S_d et S_{d^2} pour chacun d'eux. Comme on utilise l'algorithme incrémental pour calculer ces P valeurs, on a seulement besoin des n sommes partielles (colonnes) de la dernière fenêtre parcourue. De même que pour la fenêtre de référence et pour la même raison, on ajoute une case supplémentaire à ces tableaux des sommes partielles. De plus, lorsque l'on change de pixel de référence, toutes les fenêtres des candidats se décalent de un pixel à droite, et l'on voit sur la figure 3.4 que les $P-1$ *derniers* candidats de la référence précédente correspondent aux $P-1$ *premiers* candidats de la nouvelle référence.

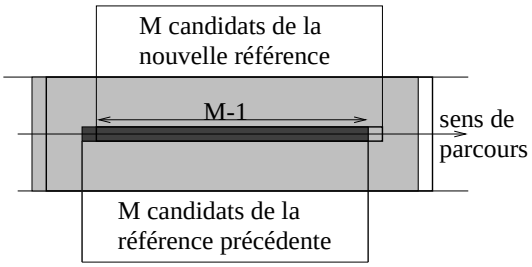

FIG. 3.4 - *Candidats communs à deux pixels de référence voisins*

Ainsi, lors des changements de référence, il nous suffit de décaler les S_d et S_{d^2} dont on dispose d'une position vers la gauche et de calculer ceux du seul candidat qui nous manque, c.-à-d. le dernier de la nouvelle référence. Or, comme on a conservé les sommes partielles de la dernière fenêtre parcourue, qui correspond au dernier candidat de la référence précédente et donc à l'avant dernier candidat de la nouvelle référence, on peut effectuer ces calculs de manière incrémentale. On a donc finalement deux tableaux de taille P pour les sommes totales et deux tableaux de taille $n+1$ pour les sommes partielles, et l'on utilise une permutation circulaire à gauche lors des changements de référence.

Enfin, le calcul de S_{gd} qui n'est autre que la corrélation proprement dite des deux fenêtres (référence et candidat), est un peu plus complexe. En effet, si l'on regarde la figure 3.5.a, on s'aperçoit que pour une fenêtre de référence, on peut décomposer la corrélation avec un candidat par les sommes partielles sur les colonnes, un peu comme les valeurs statistiques données en (3.5) et (3.6). Or, si l'on considère deux candidats voisins, on remarque que les sommes S^1_{gd} et S^2_{gd} correspondantes aux deux candidats n'ont aucun terme en commun (figure 3.5.b). On ne peut donc pas calculer la corrélation avec un candidat en fonction de celle avec le candidat précédent. On en déduit donc que pour chaque fenêtre de référence il faudra calculer indépendamment toutes les corrélations avec les fenêtres des candidats. Par contre, lorsque l'on passe d'un point de référence au suivant, toutes les fenêtres (celle de référence et celles des candidats) se décalent d'un pixel vers la droite. Si l'on considère $S^{i,j}_{gd}$ la corrélation des fenêtres de la $i^{\text{ème}}$ référence et de son $j^{\text{ème}}$ candidat, et que l'on note $C^{i,j}_p$ la corrélation des $p^{\text{ème}}$ colonnes de ces deux fenêtres. D'après (3.4), on s'aperçoit que les deux sommes $S^{i,j}_{gd}$ et $S^{i-1,j}_{gd}$ sont liés par la relation suivante :

$$S^{i,j}_{gd} = S^{i-1,j}_{gd} - C^{i-1,j}_0 + C^{i,j}_n \tag{3.9}$$

Elles ont donc une grande partie de leurs termes (par colonnes) en commun, comme on peut le voir sur la figure 3.6. On en déduit un processus incrémental de calcul des corrélations relatives à une

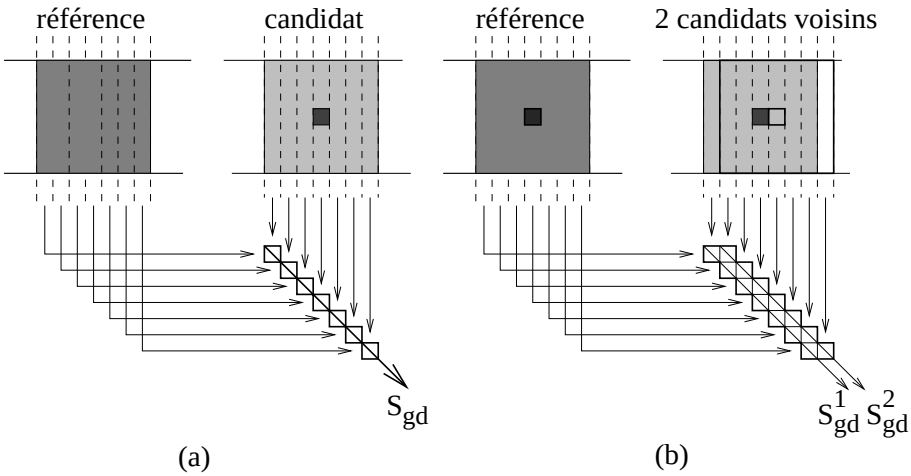

FIG. 3.5 - *Calcul des S_{gd} : (a) Décomposition en sommes partielles suivant les colonnes, (b) Indépendance des termes entre les corrélations de la fenêtre de référence avec les fenêtres de deux candidats voisins*

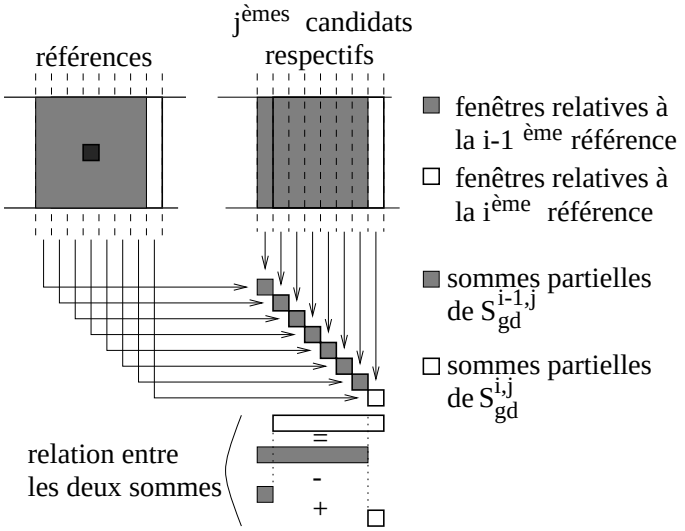

FIG. 3.6 - *Termes communs aux corrélations de deux points de référence voisins avec leur $j^{ème}$ candidat respectif*

référence à partir de celles de la référence précédente. Et ce processus est similaire à celui utilisé pour le calcul des autres sommes S_g, S_d, S_{g^2} et S_{d^2} (voir l'analogie entre la relation au bas de la figure 3.6 et la figure 3.3). Finalement, les données à stocker pour calculer ces S_{gd} sont :

Les valeurs des S_{gd} pour chaque candidat dans un tableau à P éléments ainsi que leurs n sommes partielles selon les colonnes. Comme le montre la figure 3.7, on réaligne les sommes partielles de manière à obtenir un tableau de P colonnes et n lignes. Étant donné que les sommes partielles liées à une corrélation entre la référence et un de ses candidats sont organisées verticalement dans ce tableau, il nous faut cette fois rajouter une ligne pour pouvoir stocker les nouvelles sommes partielles lors des changements de référence, et utiliser une permutation circulaire des lignes du tableau vers le haut.

Tout cela nous mène donc à un nouvel algorithme optimisé pour le calcul des coefficients de

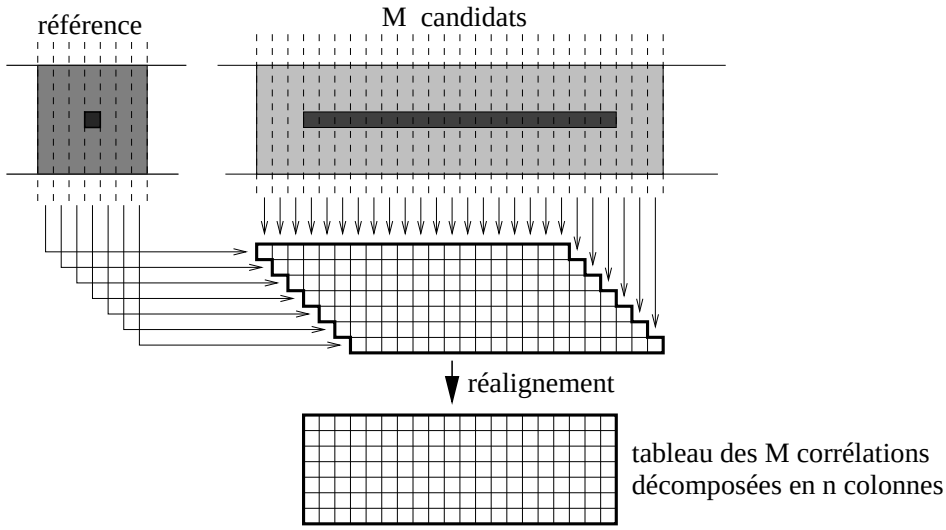

FIG. 3.7 - *Stockage des corrélations décomposées en colonnes entre la référence et les candidats*

similarité :

Pour chaque ligne épipolaire :

- Calcul complet des S_g et S_{g^2} avec leurs sommes partielles pour la fenêtre $n \times n$ du premier point à apparier dans l'image de référence,
- Calcul complet des S_d et S_{d^2} avec leurs sommes partielles pour la fenêtre du premier candidat,
- Calcul incrémental des S_d et S_{d^2} pour les autres candidats en utilisant la relation (3.8),
- Calcul complet des S_{gd} avec leurs sommes partielles pour tous les candidats,
- Calcul des mesures de similarité entre la référence courante et ses P candidats en utilisant (3.7),
- Pour chaque pixel de référence suivant :
  - Calcul des sommes partielles de S_g et S_{g^2} pour la dernière colonne de la nouvelle fenêtre de référence,
  - Calcul incrémental de S_g et S_{g^2} pour la nouvelle fenêtre de référence en utilisant (3.8),
  - Calcul des sommes partielles de S_d et S_{d^2} pour la dernière colonne de la fenêtre du dernier candidat de la nouvelle référence,
  - Calcul incrémental de S_d et S_{d^2} pour le dernier candidat en utilisant (3.8),
  - Calcul des sommes partielles des S_{gd} entre la dernière colonne de la fenêtre de référence et celle des fenêtres des P candidats,
  - Calcul incrémental des S_{gd} en utilisant (3.9),
  - Calcul des P mesures de similarité entre la nouvelle référence et ses candidats en utilisant (3.7).

Par souci de clarté, nous n'avons pas fait apparaître les permutations circulaires des sommes partielles, leur placement étant seulement relatif à l'ajout de la nouvelle colonne, il reste assez libre.

Ainsi s'achève la description de notre optimisation calculatoire. On peut noter qu'une technique similaire a été utilisée et détaillée dans [FHM+93] mais en utilisant une mesure de similarité un peu simplifiée. De même, nous aurions pu appliquer ce principe de dépendance des calculs pour déduire les informations des fenêtres d'une ligne épipolaire à partir de celles de la ligne précédente. Néanmoins, cela impliquerait le stockage en mémoire de toutes les sommes et sommes partielles pour une ligne épipolaire, ce qui peut représenter une quantité de mémoire très importante pour des images assez larges. Pour cette même raison, cela limiterait le domaine des hauteurs possibles pour les colonnes (donc les fenêtres de corrélation). De plus, cette amélioration représenterait un gain bien moins important que la précédente, car elle n'intervient, de manière vraiment sensible, qu'au niveau du premier point de référence de chaque ligne. Nous avons donc décidé de faire un compromis entre l'occupation de l'espace du temps et celui de la mémoire, en optimisant les calculs seulement au sein d'une ligne épipolaire mais pas entre les lignes.

Nous allons maintenant quantifier le gain apporté par l'utilisation de ces calculs incrémentaux.

Tout d'abord, la complexité de la formule (3.1) est en $O(n^2)$ puisque les calculs des σ_k^2 et de σ_{GD}^2 se font chacun en $O(n^2)$. Puisque Memier calcule toutes les similarités indépendamment les unes des autres, son algorithme est donc en $O(P.n^2)$ pour un pixel de référence. Ce qui donne sur une ligne épipolaire de largeur L, une complexité totale en $O(L.P.n^2)$. Maintenant, considérons le nouvel algorithme. On voit qu'il est divisé en deux parties qui sont la phase d'initialisation de la ligne en calculant les similarités du premier point de référence puis la seconde phase où l'on traite les autres points de référence de manière optimisée. Concernant la phase d'initialisation, on doit calculer complètement les σ_k^2 de la fenêtre de référence et de son premier candidat, ce qui se fait en $O(n^2)$. Les variances des autres candidats étant calculées incrémentalement, leur coût total est en $O(P.n)$. Enfin, il nous faut aussi calculer complètement tous les σ_{GD}^2. On retombe donc sur une complexité en $O(P.n^2)$ pour cette première phase. Maintenant, lors la deuxième phase, les calculs des σ_k^2 et des σ_{GD}^2 se font tous en $O(n)$. On a donc une complexité en $O(P.n)$ pour le traitement de chaque point de référence et donc une complexité sur le restant de la ligne épipolaire en $O(L.P.n)$. Nous arrivons donc finalement à un algorithme en $O(P.n^2 + L.P.n)$. On constate donc que lorsque $L \gg n$, ce qui est le cas le plus général, nous gagnons quasiment un facteur n sur l'algorithme initial. Les valeurs courantes de n pour les images SPOT étant de l'ordre de la dizaine, on se rend bien compte du gain de temps réalisé. Pour donné une idée des temps d'exécution, le traitement d'images 1951×1951 avec $N_1 = N_2 = 17, R = 4, P_1 = 15$ et $P_2 = 3$, a demandé au programme de Memier environ 5 heures de calculs sur un SUN SPARC serveur 1000, alors qu'il n'a fallu que 45 minutes environ à notre nouveau programme. Des mesures plus précises sont données dans le paragraphe 3.4.1. Enfin, cette analyse montre aussi que l'utilisation des calculs incrémentaux entre les lignes épipolaires nous ferait seulement gagner $P.n^2$ opérations par ligne, ce qui est relativement négligeable par rapport au reste des calculs.

Maintenant que nous avons augmenté la vitesse de l'algorithme de corrélation, nous pouvons passer au problème de la qualité des résultats obtenus.

3.2.2 Qualité des résultats

De la même manière que pour l'aspect calculatoire, nous tentons ici d'apporter quelques modifications à l'algorithme initial pour améliorer la qualité des résultats. Il est important de noter que la qualité et la quantité des résultats sont deux notions étroitement liée dans notre cas. Par exemple, pour une précision donnée des calculs, un MNT plus dense, c.-à-d. contenant plus de points valides sera considéré de meilleure qualité. De ce point de vue, on peut dire que la qualité englobe ou implique la quantité.

Le premier point sur lequel nous intervenons concerne la phase d'interpolation des disparités entre les deux corrélations. Ensuite, nous abordons le problème du recalage des pixels dans l'image droite.

Interpolation des disparités

Comme nous l'avons montré dans le chapitre 2, la nature globale des splines utilisées n'est pas très bien adaptée à l'interpolation de données d'origine naturelle telles que des disparités (ou des altitudes). Dans le cas particulier des disparités, les splines bi-cubiques tendent à lisser les données alors que nous savons que ce genre de données peut contenir des discontinuités importantes dues soit à un relief accidenté, tel que des hauts sommets ou au contraire des gorges, soit à des occlusions dans l'une des deux images. Par conséquent, le résultat de l'interpolation peut ne pas être aussi précis que nécessaire, impliquant des erreurs potentielles dans la seconde corrélation effectuée à la résolution maximale des images.

Pour remédier à ce problème, nous proposons d'utiliser une autre méthode d'interpolation permettant de conserver les discontinuités dans le flot de données reconstruit. Même si ces discontinuités peuvent plus facilement générer des trous dans le MNT final, cela est préférable car les géologues avec lesquels nous travaillons préfèrent des données manquantes à des données fausses. Pour obtenir une telle caractéristique, le recours à une fonction d'interpolation locale semble être le meilleur choix. C'est pourquoi nous utilisons une approche similaire à celle de Catmull-Rom [CR74]. Le principe de notre technique est basé sur l'idée que les valeurs des données à la résolution initiale sont considérées comme les moyennes des données à une résolution plus haute. Notre processus de base multiplie la résolution par deux mais pourrait facilement être étendu à des coefficients entiers quelconques. De plus, le coefficient de réduction R utilisé pour la première corrélation est, pour des raisons pratiques, toujours un multiple de deux. Ainsi, notre interpolation est effectuée en appliquant le processus de base itérativement autant de fois que nécessaires pour arriver à la résolution finale, soit en $Log_2(R)$ étapes. Si l'on considère le cas unidimensionnel, nous décomposons chaque donnée P_i^j à la résolution j, en deux nouvelles valeurs P_i^{j+1} et P_{i+1}^{j+1} à la résolution j+1, de part et d'autre de P_i^j, de telle sorte que :

$$\frac{P_i^{j+1} + P_{i+1}^{j+1}}{2} = P_i^j$$

et

$$Pente([P_i^{j+1} P_{i+1}^{j+1}]) = Pente([P_{i-1}^j P_{i+1}^j])$$

Cela revient à dire que la moyenne des deux nouvelles valeurs est égale à la valeur initiale dont elles sont issues, et que la droite formée par les deux nouveaux points est parallèle à celle formée par les deux voisins de l'ancien point. La figure 3.8 met en évidence ces deux contraintes.

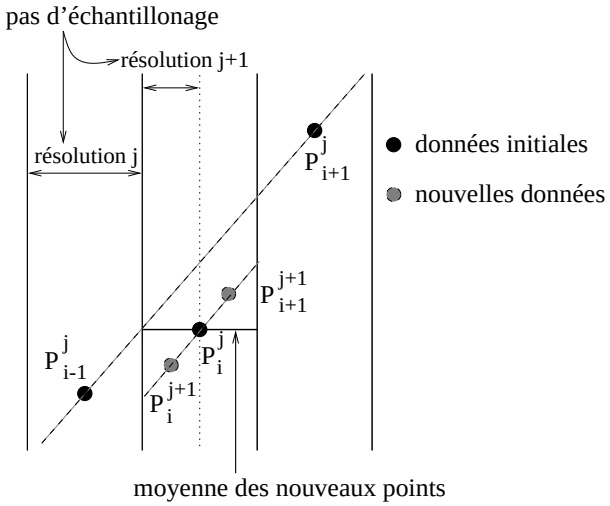

FIG. 3.8 - *Interpolation locale en divisant chaque donnée existante en deux nouvelle données*

Nous n'utilisons pas directement les splines de Catmull-Rom principalement pour des raisons pratiques quant à la gestion des discontinuités. De plus, notre méthode ne nécessite que trois points initiaux pour créer deux nouveaux points alors que la méthode de Catmull-Rom en requiert quatre pour ne créer qu'un nouveau point. Cela se traduit par un ensemble de points interpolés légèrement plus grand (sur les bords).

Pour prendre en compte les discontinuités, nous introduisons un ratio r défini par :

$$r = \frac{P_{i-1}^{j} - P_{i}^{j}}{P_{i+1}^{j} - P_{i}^{j}} \tag{3.10}$$

Ce qui n'est autre que le rapport des différences relatives (signées) de la valeur de la donnée en cours de traitement et des valeurs de ses deux voisins. Ensuite, nous utilisons un seuil t pour limiter ce ratio. Étant donné que ce seuil représente en quelque sorte l'extension maximale de la courbe au-delà de laquelle une discontinuité apparaît, on l'appellera aussi coefficient d'élasticité. Ainsi, notre algorithme doit gérer quatre cas :

- $r > 0$: les valeurs des voisins sont du même côté du point courant, c.-à-d. toutes les deux supérieures ou inférieures
- $r < 0$: les valeurs des voisins sont de part et d'autre du point courant

  et
- $|r| < t$: cas normal
- $|r| > t$: discontinuité

Plaçons nous dans le cas où $r > 0$. Lorsqu'une discontinuité est trouvée ($|r| > t$), nous gardons toujours la contrainte de la moyenne des deux nouvelles valeurs mais par contre nous modifions celle sur les pentes des droites. En effet, dans ce cas, nous calculons les nouveaux points en nous basant sur la pente entre le point courant et son voisin ayant la valeur la plus proche de la sienne. Cela permet de suivre le comportement de la courbe du côté opposé à la discontinuité. La figure 3.9 décrit les cas normaux et discontinus apparaissant avec un ratio positif.

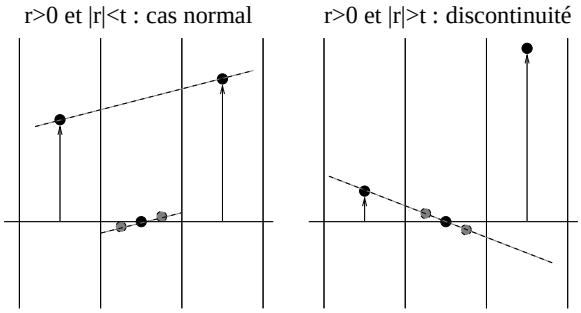

FIG. 3.9 - *Situations possibles avec un ratio positif*

Si nous considérons maintenant les cas où le ratio est négatif, nous n'avons pas besoin de distinguer le cas normal du cas discontinu puisque notre schéma d'interpolation agira implicitement de la manière voulue, sans effacer ou lisser les discontinuités. Néanmoins, dans le cas de discontinuités fortes, il peut apparaître des artefacts renforçant la cassure, comme on peut le voir dans la figure 3.10. En fait, tout comme pour les splines de Catmull-Rom, notre fonction d'interpolation ne possède pas la propriété de l'enveloppe convexe (voir [FvDFH90]). Cette propriété qui est vérifiée par d'autres familles de courbes (Bezier, Hermite, B-Splines, ...) assure que tous les points créés se trouvent obligatoirement dans l'enveloppe convexe des points initiaux. Enfin, notons que la configuration normale continue est exhibée dans la figure 3.8.

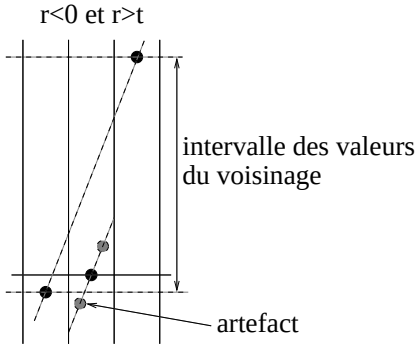

FIG. 3.10 - *Artefacts générés par les fortes discontinuités avec un ratio négatif*

Dans le contexte d'interpolation de disparités, les points voisins d'une donnée peuvent ne pas être valides à cause des occlusions entre les images et des faux appariements (pas de correspondance dans l'espace de recherche). Memier traite ce problème en calculant la spline bi-cubique centrée en un point donné seulement si au moins quatre des cinq points voisins sont valides. Dans le cas contraire il donne une valeur nulle à la spline. Ensuite, les nouvelles parallaxes ne sont calculées

que si au moins dix des seize coefficients des splines 2D nécessaires sont non nuls. Dans notre cas, pour chaque point valide nous parcourons l'ensemble des données à gauche et à droite du point et nous cherchons le premier point valide dans chaque direction. Par conséquent, la formule initiale du ratio r donnée dans (3.10) devient :

$$r = \frac{P^j_{i-k} - P^j_i}{P^j_{i+l} - P^j_i} \text{ où } k \geq 1 \text{ et } l \geq 1 \tag{3.11}$$

où P^j_{i-k} représente le premier point valide à gauche du point courant, et P^j_{i+l} celui à droite. Ensuite, le processus d'interpolation reste le même si ce n'est que l'on prend en compte les distances des deux voisins valides au point courant pour le calcul des pentes. Enfin, lorsque l'on rencontre un point non valide, nous n'interpolons pas et nous indiquons que les deux nouvelles valeurs à la résolution supérieure sont elles aussi invalides. On remarque que selon l'importance de la région non valide et le coefficient de réduction des images, notre méthode peut laisser aussi des zones de points (disparités) non valides à haute résolution pour lesquels aucun traitement ne pourra être réalisé. Par contre, notre méthode aura tendance à rétrécir la zone invalide avec de bonnes estimations sur les bords par effet de prolongement des zones valides. Enfin, si la zone invalide est petite, notre nouvelle interpolation bouchera la zone avec des valeurs raisonnablement justes par prolongement des zones valides avoisinantes.

Finalement, nous avons donc un algorithme de sur-échantillonnage mieux adapté à l'interpolation des données naturelles que sont les disparités. De plus, étant donné que ce processus unidimensionnel (facilement extensible au cas 2D) est considérablement plus rapide que le calcul des splines bi-cubiques, nous gagnons aussi sur la vitesse de l'algorithme général.

Recalage des pixels de l'image droite

Une autre modification apportée au programme de Memier porte sur le recalage des pixels de l'image droite en fonction des disparités approchées obtenues par la première corrélation. Dans la version initiale, les pixels sont simplement déplacés. Mais cela peut créer des trous entre des pixels voisins lorsque ceux-ci ont des disparités assez différentes, comme le montre la figure 3.11. Et ces trous, qui n'ont plus de lien direct avec la réalité de l'observation peuvent perturber la deuxième corrélation. C'est pourquoi nous avons ajouté une phase d'interpolation linéaire entre les pixels initialement voisins de manière à combler ces trous avec des informations plus réalistes directement en rapport avec les observations.

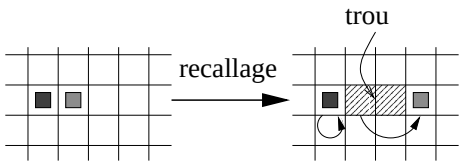

FIG. 3.11 - *Trou dans l'image droite recalée généré par des déplacements différents de pixels initialement voisins*

Cependant, on peut noter que cette modification a une portée relativement faible sur la qualité

globale de la corrélation. Cela vient du nombre peu élevé de trous (les disparités varient continuement la plupart du temps) et de leur taille relativement faible (bornée par la parallaxe maximale).

Ceci termine la description des modifications apportées en vue d'améliorer la qualité des résultats. Comme nous l'avons déjà mentionné à la fin du chapitre précédent, la qualité des résultats est difficilement vérifiable de manière précise. La comparaison avec d'autres modèles de terrain, calculés par d'autres méthodes n'est pas vraiment adaptée puisque ceux-ci ne sont pas forcément ni à la même résolution ni plus justes que nos résultats. De plus, nous ne faisons aucune modification sur la méthode de choix de l'appariement vue aux paragraphes 2.1.3 et 2.1.4 dont nous savons qu'elle est relativement robuste et fiable. En effet, la technique de pondération des maxima locaux lors des appariements nous assure de choisir, dans la majorité des cas, le meilleur appariement possible lorsque celui-ci est disponible et sinon de ne pas faire d'appariement du tout pour éviter des erreurs. D'un point de vue topologique, les terrains obtenus semblent tout à fait précis et réalistes aux spécialistes géologues avec lesquels nous avons travaillé. Enfin, nous obtenons un nombre d'appariements et donc de points 3D plus important que la version de Memier. Cela a un impact direct sur la précision finale du MNT puisque lorsqu'il y a peu de points 3D, un grand nombre de cellules du MNT peuvent être vides et l'on est donc obligé de recourir à des techniques de remplissage plus ou moins précises pour boucher ces trous. Bien sûr, les terrains que nous obtenons contiennent aussi des trous mais beaucoup moins nombreux et en général plus petits que dans la version initiale. Les images de la figure 3.12 montrent les MNT obtenus à partir d'images de taille 1951×1951, avec le programme de Memier puis avec notre programme. Bien qu'il y ait encore des trous et quelques erreurs de corrélation, on constate nettement un accroissement de la densité avec notre version de l'algorithme. Enfin, on peut vérifier que le relief de notre MNT est en accord avec celui du MNT généré par l'algorithme initial.

Le point suivant à aborder est la réduction de la mémoire utilisée. Celle-ci n'a pas le seul avantage d'économiser les ressources matérielles de mémoire vive mais peut aussi contribuer à augmenter la vitesse d'exécution en limitant les accès mémoire distants les uns des autres et en évitant le recours à la mémoire virtuelle sur disque dur (*swaping*) ralentissant considérablement les accès aux données et donc l'exécution générale du programme.

3.2.3 Gestion de la mémoire

Le programme de Memier n'est vraiment pas optimal en ce qui concerne la gestion de la mémoire. En effet, les différents tableaux utilisés sont alloués statiquement et à chaque étape de l'algorithme, l'ensemble des données à traiter est entièrement chargé en mémoire, ce qui implique des ressources énormes en mémoire. Bien sûr, il faut noter que la préoccupation de Memier n'était pas de produire un programme optimal en temps et en mémoire, mais surtout de confirmer le bien-fondé et la qualité de sa méthode de reconstruction tridimensionnelle.

Notre principal travail pour gérer efficacement la mémoire a consisté principalement à ne charger en mémoire, à un instant précis, que les données nécessaires au traitement en cours en utilisant un système de tampons pour chaque étape du schéma algorithmique quand cela est possible.

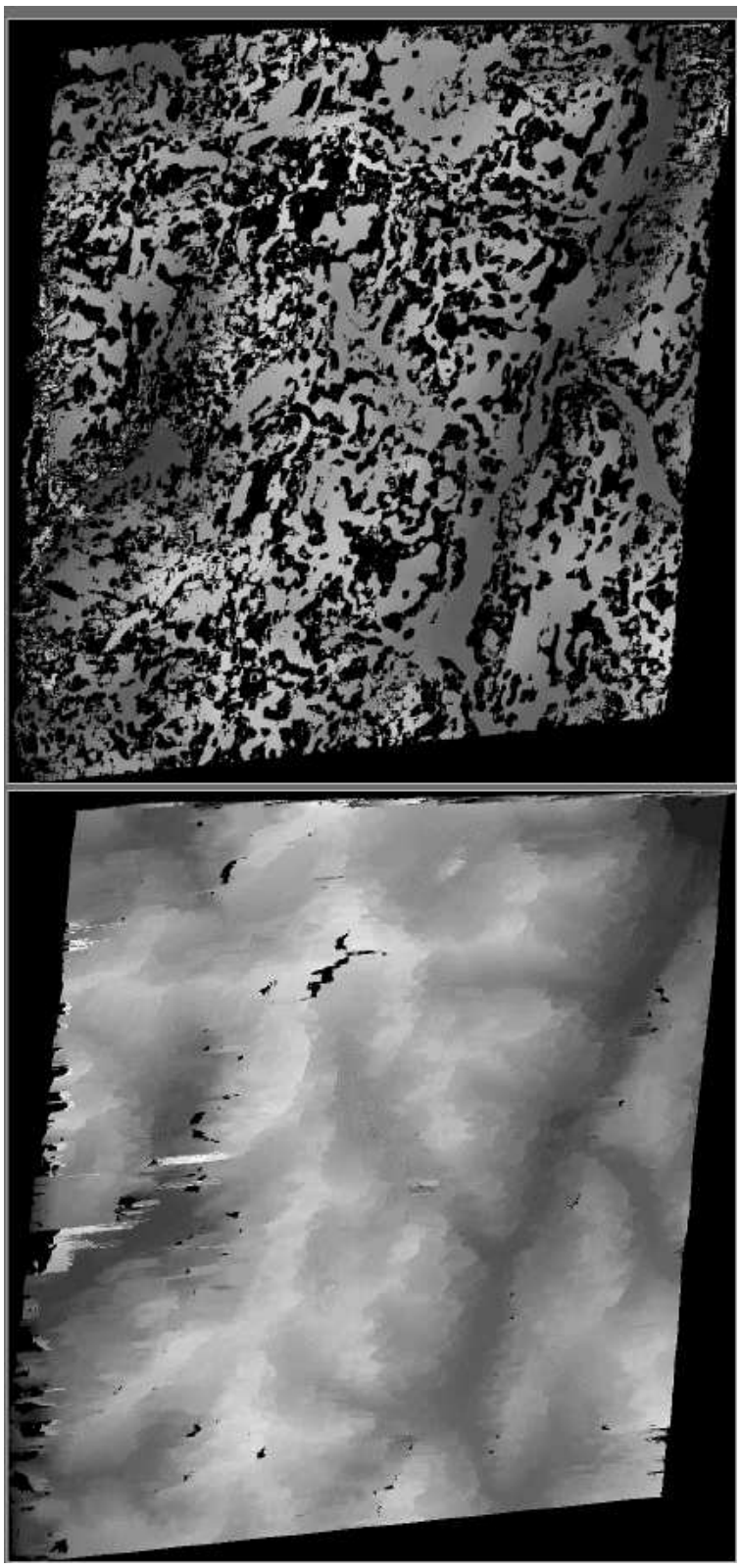

FIG. 3.12 - *Comparaison des MNT générés par l'algorithme de Memier (en haut) et notre nouvel algorithme (en bas)*

Ainsi, si l'on prend les deux étapes de corrélation de l'algorithme donné au paragraphe 2.4.1, une économie importante de mémoire a été réalisée sur le stockage des images. Cela a été possible grâce à l'indépendance des lignes épipolaires par rapport aux traitements de mise en correspondance, au caractère ordonné des lignes dans les images et à l'étroite relation entre les données nécessaires au traitement de deux lignes consécutives. En effet, on a vu dans le chapitre 2 que l'algorithme de mise en correspondance traite les lignes épipolaires indépendamment les unes des autres. De plus, lorsque l'on utilise des fenêtres de hauteur n pour mesurer les similarités entre les points des deux images sur une même ligne épipolaire, le traitement de cette ligne épipolaire ne met en jeu que les $\lfloor \frac{n}{2} \rfloor$ lignes (en plus d'elle même) de part et d'autre de celle-ci. Enfin, le sens de parcours du traitement des lignes épipolaires étant identique à l'ordre de ces lignes (du haut des images vers le bas), cela implique que les $n-1$ *dernières* lignes requises pour le traitement d'une épipolaire correspondent aux $n-1$ *premières* lignes pour le traitement de la suivante, comme le montre la figure 3.13.

FIG. 3.13 - *Lignes épipolaires intervenant dans le traitement de deux lignes consécutives*

On en déduit donc que pour les deux phases de corrélation de notre schéma algorithmique, on peut ne stocker en même temps en mémoire que les n lignes consécutives impliquées dans le traitement de l'épipolaire courante. Puis lorsque l'on passe au traitement de l'épipolaire suivante, on effectue un roulement des lignes en les décalant d'une case vers le haut puis on stocke dans la dernière ligne du tableau, les pixels de la ligne suivante de l'image. Cependant, lors de la deuxième corrélation, pour calculer les disparités finales sur une ligne épipolaire, nous avons besoin des disparités approchées sur cette ligne, obtenues lors de la quatrième étape de l'algorithme, ce qui nécessite simplement le stockage supplémentaire d'une ligne de disparités.

Considérons maintenant la deuxième étape du processus qui correspond à la mise en correspondance globale des lignes épipolaires par programmation dynamique. On a vu, dans les paragraphes 2.1.3 et 2.1.4, que l'on utilise un graphe 2D connectant entre eux tous les pixels appariables des deux lignes épipolaires conjuguées des deux images. Si l'on considère la largeur des images et que l'on tient compte des bords non traités des images, le parcours du graphe implique le recours à un tableau de $(L - 2(\lfloor \frac{P_1}{2} \rfloor + \lfloor \frac{n}{2} \rfloor))$ colonnes et $(L - 2(\lfloor \frac{n}{2} \rfloor))$ lignes. Et comme P_1 et n sont toujours choisis impairs pour avoir autant d'éléments à gauche qu'à droite, cela revient à un tableau de $(L - P_1 - n + 2)$ colonnes et $(L - n + 1)$ lignes. Les éléments du tableau étant codés sur plusieurs octets (au moins 4 pour des réels), on se rend bien compte de la quantité totale de mémoire nécessaire pour ce tableau.

Maintenant, si l'on reprend les contraintes relatives au chemin que l'on cherche dans ce graphe, on sait qu'il est confiné dans la bande diagonale de hauteur P_1 indiquée sur la figure 2.4. On a donc seulement besoin des valeurs dans cette bande pour calculer notre chemin. Ainsi, notre idée est de réduire le tableau du graphe à cette seule bande diagonale tout en réalignant verticalement

les colonnes de cette bande pour obtenir un tableau de P_1 lignes au lieu des $(L - n + 1)$ nécessaires initialement. La figure 3.14 indique cette transformation.

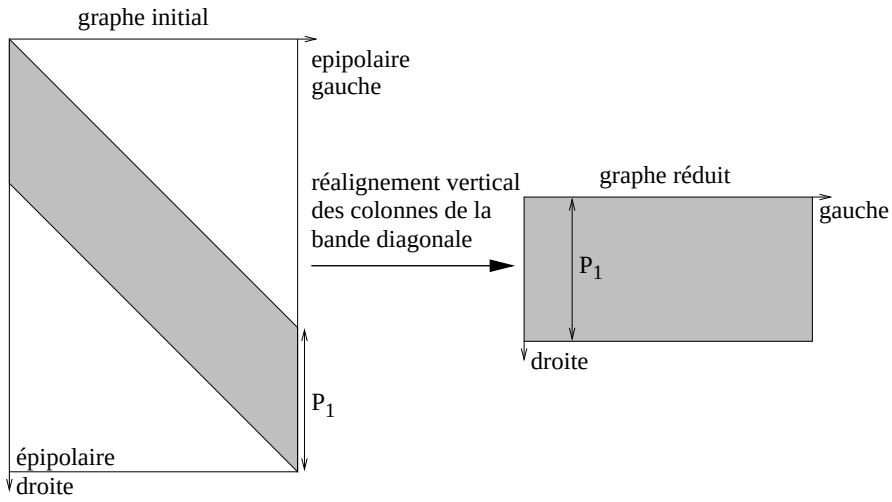

FIG. 3.14 - *Restriction du graphe utilisé pour la programmation dynamique à la bande diagonale et réalignement vertical des colonnes*

Cette modification de l'organisation du graphe implique aussi une légère modification pour le calcul du chemin de coût minimal. En effet, reprenons la contrainte d'ordre sur le chemin, illustrée dans la figure 2.5. Le réalignement vertical décale d'un pixel vers le haut l'ensemble de la zone d'évolution du chemin et par conséquent celui de ses points de prolongement. Ainsi, lorsque l'on cherche le point de prolongement du chemin courant dans la colonne suivante, la seule différence avec la méthode initiale est qu'il faut le chercher à partir de la ligne précédente dans le tableau. Pour bien faire ressortir cette modification du parcours du graphe dû au recalage vertical, la figure 3.15 indique les points de prolongement du même exemple de chemin que celui donné dans la figure 2.5.

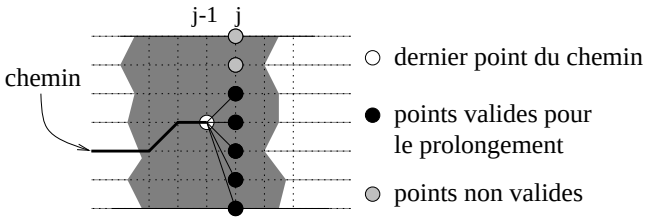

FIG. 3.15 - *Méthode de parcours du nouveau graphe pour le calcul des chemins*

L'étape suivante dans le processus de reconstruction est l'interpolation des disparités. On a vu dans le paragraphe précédent une nouvelle méthode d'interpolation unidimensionnelle plus adaptée à la nature des données traitées. L'interpolation du tableau bidimensionnel des disparités avec cette fonction peut se faire en deux passes, une première interpolation dans la direction horizontale puis une seconde dans la direction verticale. Comme les données sont générées ligne par ligne, l'interpolation dans la direction horizontale ne pose pas de problème puisqu'elle ne requiert qu'une ligne de disparités à la fois. Par contre, dans la direction verticale, on aura besoin d'au moins trois lignes consécutives. Cela vient du fait que notre fonction a besoin des deux voisins d'un point

pour le diviser en deux nouvelles valeurs et que ces voisins doivent être des points valides. Pour interpoler verticalement, il faudrait donc stocker le nombre de lignes minimum pour que chaque pixel à interpoler dans la ligne courante ait deux voisins valides dans sa colonne. On s'aperçoit donc que le plus simple serait encore de stocker toutes les lignes interpolées horizontalement et de réaliser l'interpolation verticale de chaque colonne seulement après. Cela n'est malheureusement pas très économique en terme de consommation mémoire. Une autre alternative est possible en faisant une nouvelle fois un compromis entre le temps et la mémoire. En regard à l'accélération apportée sur les temps des calculs, on peut envisager de réduire les images pour la première corrélation seulement dans le sens horizontal. Ainsi, nous n'aurions plus à faire l'interpolation verticale et donc plus à stocker toutes les lignes des disparités mais seulement une à la fois.

Comparons les temps de calcul ajoutés et supprimés par cette nouvelle modification. On reprend R comme facteur de réduction des images, et on pose L et H les largeur et hauteur initiales des images. La suppression de la réduction verticale implique que la première corrélation devient R fois plus longue puisque l'on a H lignes à traiter au lieu de $\lfloor \frac{H}{R} \rfloor$. Ainsi, le temps de calcul supplémentaire est donc en $O((H - \lfloor \frac{H}{R} \rfloor)(P_1.n^2 + \lfloor \frac{L}{R} \rfloor.P_1.n)$. D'un autre côté, la suppression de l'interpolation verticale nous fait gagner environ un facteur 2 sur la phase d'interpolation, et plus précisément un temps en $O(L.(2R-1).\lfloor \frac{H}{R} \rfloor)$. Enfin, si l'on compare les ordres de grandeur des éléments de ces complexités, on voit que l'on perd effectivement du temps mais pas autant que ce que l'on aurait pu imaginer a priori.

La phase de rééchelonnement des disparités et de recalage des pixels de l'image droite pouvant se faire indépendamment sur chaque ligne de données, il suffit de stocker une seule ligne à la fois. D'une manière similaire, la reconstruction tridimensionnelle des points étant indépendante d'un point à l'autre, on pourrait ne stocker qu'un seul point à la fois. Néanmoins, vue la faible quantité de mémoire demandée pour un couple de points appariés (3 ou 4 coordonnées entières), et sachant qu'il est préférable de faire peu d'accès au disque pour beaucoup de données plutôt que beaucoup d'accès pour peu de données, on chargera en mémoire plusieurs points en même temps. Enfin, l'étape d'échantillonnage du MNT peut se faire cellule par cellule et requiert la présence en mémoire de tous les points 3D se trouvant dans la cellule en court de traitement. Le cellules ayant généralement des tailles de l'ordre de quelques pixels, le nombre de points à stocker en même temps reste très faible et on peut éventuellement stocker les points de plusieurs cellules en même temps.

Ceci termine les principaux changements effectués dans chaque étape pour économiser la consommation de mémoire, la liste suivante récapitule ces modifications :

- la première corrélation est réalisée sur des images réduites en largeur, en stockant N_1 lignes consécutives des deux images pour traiter une ligne épipolaire. Les lignes de chaque tampon sont permutées cycliquement vers le haut et la dernière ligne est mise à jour avec la ligne suivante dans l'image,
- la recherche du chemin de coût minimal se fait dans le graphe restreint à la bande diagonale réalignée verticalement pour former un tableau $(L - P_1 - N_1 + 2) \times P$,
- l'interpolation des disparités ne se fait plus que dans la direction horizontale, en utilisant la nouvelle fonction unidimensionnelle décrite dans 3.2.2, et en stockant une ligne à la fois,

- le rééchelonnement des disparités approchées et le recalage de l'image droite ne requièrent qu'une seule ligne de chaque en même temps,
- la deuxième corrélation nécessite un tampon de N_2 lignes consécutives pour chaque image, et une ligne de disparités approchées,
- pour la reconstruction des points 3D, on choisit de stocker les couples de points appariés de toute une ligne épipolaire pour rester cohérent avec toutes les autres étapes qui génèrent leurs résultats ligne par ligne,
- le même choix a été fait pour les cellules d'échantillonnage de toute une ligne du MNT.

Toutes ces modifications ont considérablement réduit la consommation de mémoire vive de l'algorithme de Memier. Le code initial ne pouvait traiter que des sous-images d'environ 2000×2000 pixels au lieu des images 6000×6000 délivrées par SPOT, et ce, sur une machine avec plus de 128 Mo de mémoire vive. Alors que notre nouvelle version nécessite moins de 10 Mo de mémoire pour ce même traitement, ce qui se trouve couramment sur la plupart des machines à l'heure actuelle.

Nous venons de voir comment on a réduit la consommation de mémoire de notre algorithme. La dernière caractéristique nous restant à traiter concerne les entrées/sorties des données entre chaque étape du processus.

3.2.4 Entrées/sorties

Les différents programmes correspondants aux étapes de l'algorithme communiquent leurs données en utilisant des fichiers temporaires. Ainsi, à chaque étape, les données en entrées sont lues dans le ou les fichiers généré(s) à l'étape précédente et les données produites sont écrites dans un ou plusieurs fichiers pour l'étape suivante. Cela pose deux problème qui sont d'une part la taille très importante des fichiers temporaires et d'autre part les temps d'accès en lecture et en écriture qui sont relativement longs surtout si le nombre de données à lire ou à écrire est important, ce qui est le cas. De plus, dans un contexte de parallélisation d'algorithme, il est préférable d'avoir un seul programme important plutôt que plusieurs petits programmes. Par conséquence, notre dernière étape d'amélioration va consister à regrouper en un seul programme toutes les étapes de l'algorithme pour former une chaîne complète de traitement (*pipeline*). Cette chaîne est réalisée en reliant directement les sorties (résultats) d'une étape aux entrées (données) de l'étape suivante. La contrainte principale est de conserver au maximum le système des tampons vu précédemment pour économiser la mémoire. La figure 3.16 donne la structure de la chaîne ainsi que les données nécessaires à l'entrée de chaque étape.

On se rend compte que seules les étapes de corrélation nécessitent plusieurs lignes pour leur fonctionnement, les autres étapes ne demandant qu'une ligne à la fois. De plus, concernant la deuxième corrélation, on sait que l'on a besoin de N_2 lignes de l'image gauche et de l'image droite recalée. Il va donc falloir remplir complètement les tampons des images avant de pouvoir traiter la première ligne épipolaire. Or, pour obtenir les lignes de l'image droite recalée, on a besoin des disparités approchées. Il faudrait donc stocker aussi dans un autre tampon, les N_2 lignes de

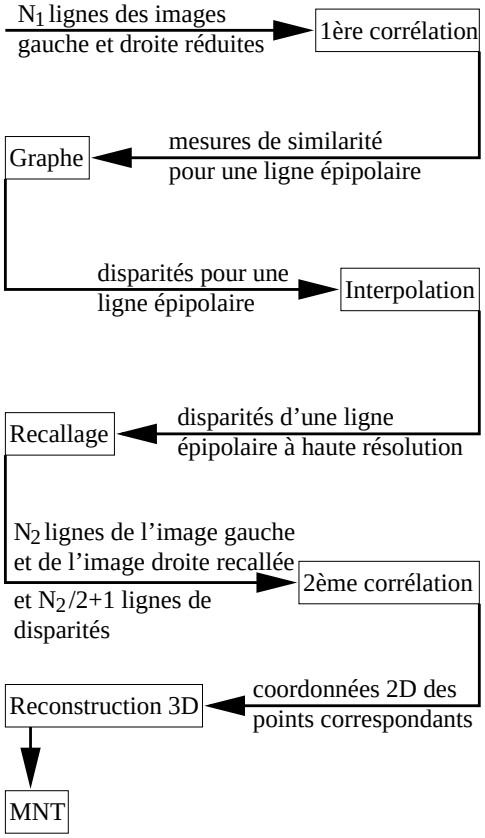

FIG. 3.16 - *Chaîne de traitement avec les données requises à chaque étape*

disparités approchées. Néanmoins, comme il est indiqué sur la figure 3.16, on peut se contenter de n'en garder en mémoire que $\lfloor \frac{N_2}{2} \rfloor + 1$ à la fois. En effet, on sait que les $\lfloor \frac{N_2}{2} \rfloor$ premières lignes recalées ne pourront être appariées puisque les fenêtres de corrélation ne seraient pas complètes. Elles sont justes utilisées comme données pour apparier les $\lfloor \frac{N_2}{2} \rfloor$ lignes suivantes. Ainsi, il est inutile de stocker les disparités approchées pour ces lignes. Par contre, la ligne courante est appariable dans la deuxième corrélation puisque l'on a les N_2 lignes nécessaires. De même, les $\lfloor \frac{N_2}{2} \rfloor$ lignes suivantes le seront aussi lorsque l'on changera de ligne et que le tampon sera remis à jour par décalage des lignes vers le haut. Et comme on a été obligé de calculer à l'avance les disparités sur ces lignes pour pouvoir les recaler, on les garde en mémoire en attendant leur traitement pour ne pas avoir à refaire le calcul. Ainsi, on a finalement besoin d'un tampon de $\lfloor \frac{N_2}{2} \rfloor + 1$ lignes pour stocker les disparités approchées de la ligne en cours de traitement et des ses $\lfloor \frac{N_2}{2} \rfloor$ suivantes. Les autres étapes ne posent pas de problème particulier puisqu'elles prennent une ligne en entrée et produisent une ligne en sortie. Par contre, la dernière étape d'échantillonnage du MNT n'a, en fait, pas été incorporée dans cette version séquentielle de la chaîne de traitement (bien que cela eut été trivial). La raison vient du fait que pour un même ensemble de points 3D obtenus après reconstruction, on peut produire des MNT à des résolutions différentes en prenant des tailles de cellule différentes. De même, les seuils de divergence au sein des cellules peuvent varier eux aussi d'un MNT à l'autre. On comprend donc bien qu'il ne serait pas intéressant de refaire tous les calculs de reconstruction 3D dans le seul but d'obtenir plusieurs MNT d'une même région échantillonnés différemments.

3.2.5 Conclusion

Finalement, nous disposons maintenant d'un programme rapide, peu gourmand en mémoire et minimisant les entrées/sorties, qui intègre toutes les étapes du processus de reconstruction. De plus, cette nouvelle version séquentielle produit des résultats de meilleure qualité car plus denses qu'auparavant. Elle constitue donc une base solide et va nous servir de modèle pour la conception de l'algorithme parallèle. Cette phase de parallélisation est décrite en détails dans la partie suivante.

3.3 Algorithme parallèle

Comme nous l'avons précisé dans l'introduction, nous utilisons le modèle de programmation parallèle MIMD avec mémoire distribuée. La machine parallèle est donc constituée d'une collection de NbP processeurs numérotés de 0 à $NbP-1$, chacun d'eux ayant son propre espace de mémoire. Le communications de données entre les processeurs se font par échange de messages explicites. De plus, l'indexage des lignes des images commence à 0.

La parallélisation de notre algorithme séquentiel compact est relativement directe grâce à l'indépendance verticale du traitement des lignes. Notre problème principal réside dans le choix de la stratégie de répartition du travail et donc des données. Une première idée serait d'utiliser une stratégie de force brute où l'ensemble des données serait *dupliqué* sur chaque processeur, et où chaque processeur traiterait une sous-partie différente des données. Mais cela nous ferait perdre tout le bénéfice de notre travail sur les économies de mémoire. Ainsi, nous considérons dans la suite les stratégies où les données sont *réparties* sur les processeurs de manière à n'avoir sur chacun d'eux que les données qui lui sont nécessaires pour ses calculs.

Nous proposons tout d'abord une méthode de répartition naturelle des données sur les processeurs. Puis, dans un deuxième temps, nous abordons une stratégie permettant d'obtenir une meilleure répartition des charges de travail sur l'ensemble des processeurs. Enfin, nous interprétons les résultats expérimentaux obtenus sur une machine parallèle Cray T3E et nous donnons un modèle partiel d'exécution de l'algorithme qui nous permettra d'évaluer approximativement le nombre optimal de processeurs à utiliser en fonction des données initiales. Mais avant tout cela, la toute première étape de l'algorithme qu'il nous faut étudier est la méthode de chargement des données et de leur distribution initiale sur les processeurs.

3.3.1 Chargement des données

On suppose que l'on connaît pour chaque processeur, la partie des données qui lui sont assignées au départ. Il existe alors deux méthodes pour charger et distribuer les données en respectant cette répartition initiale. La première consiste à charger toute les données sur un processeur puis celui-ci redistribue une partie des données sur les autres processeurs. S'il n'y a pas assez de mémoire disponible sur le processeur qui lit, on peut faire une lecture partie par partie avec à chaque fois une redistribution à la volée. Cette méthode est généralement utilisée sur les machines parallèles

où seulement un processeur spécifique peut accéder aux fichiers. La seconde méthode consiste à ce que chaque processeur lise directement sa partie de données dans les fichiers. Néanmoins, si le fichier ne peut être lu par plusieurs processeurs à la fois, il faudra utiliser des synchronisations pour séquencer les accès de chacun d'eux. Dans notre cas, nous avons opté pour la seconde méthode car les machines que nous utilisons permettent l'accès aux fichiers par tous les processeurs.

Dans les paragraphes suivants, on considère que les données sont déjà chargées en mémoire en accord avec la répartition initiale propre à chaque stratégie.

3.3.2 Répartition naturelle des données

Puisque les traitements sont indépendants d'une ligne à l'autre, comme on l'a vu précédemment, la première idée qui vient à l'esprit est de diviser les images en bandes de même hauteur (à une ligne près si le nombre de bandes n'est pas un diviseur du nombre de lignes) et d'assigner le traitement de chaque bande à un processeur.

On a vu dans la partie 3.2 que l'appariement de deux lignes épipolaires conjuguées nécessite $\lfloor \frac{n}{2} \rfloor$ lignes d'image de part et d'autre de la ligne courante. Cela implique deux choses : d'une part que les $\lfloor \frac{N_1}{2} \rfloor$ premières et dernières lignes des images ne pourront être traitées dans la première corrélation, et d'autre part, que les $\lfloor \frac{N_1}{2} \rfloor$ lignes avant et après chaque bande d'image associée à un processeur devront être rajoutées (quand cela est pertinent) si l'on veut pouvoir traiter toutes les lignes de la bande initiale. Le schéma de partitionnement qui en découle est donné dans la figure 3.17, où H est la hauteur des images, "/" représente la division entière et "%" est l'opérateur modulo. Pour des raisons de clarté, nous alternons les niveaux de gris entre les bandes consécutives, et les flèches indiquent les ensembles de données correspondants aux bandes gris foncé. Cette figure met aussi en évidence les duplications de données entre les processeurs dues aux chevauchements des bandes après l'ajout des lignes supplémentaires. Cependant, ces duplications ont généralement des proportions bien plus petites que sur la figure 3.17, où elles ont été sciemment exagérées pour être visibles. Enfin, on note que la première et la dernière bande des images ne peuvent être complétées que d'un côté et contiennent les premières et dernières lignes qui ne pourront être appariées.

Une fois la redistribution des données effectuées, chaque processeur peut traiter sa bande d'image localement et générer les résultats correspondants jusqu'à l'étape du recalage de l'image droite. Après cette étape, on dispose donc pour chacune des lignes traitées dans une bande, des disparités approchées et des pixels recalés de l'image droite. On peut noter au passage que l'on doit stocker les disparités approchées de toutes les lignes traitées par un processeur, puisque celles-ci sont nécessaires au calcul des disparités finales après la deuxième corrélation. Ensuite, pour cette seconde corrélation, on va se trouver confronté au même problème que précédemment concernant les lignes supplémentaires. Cette fois, il va falloir compléter les bandes en ajoutant $\lfloor \frac{N_2}{2} \rfloor$ lignes de part et d'autre, en sachant que pour l'image droite, se sont les lignes recalées qui nous intéressent maintenant. La figure 3.18 montre la répartition des données qui sont disponibles avant la deuxième corrélation (à gauche) et les bandes avec les lignes supplémentaires (à droite). Une fois cette nouvelle répartition réalisée, chaque processeur va appliquer la seconde corrélation sur sa bande locale et générer les points 3D correspondants.

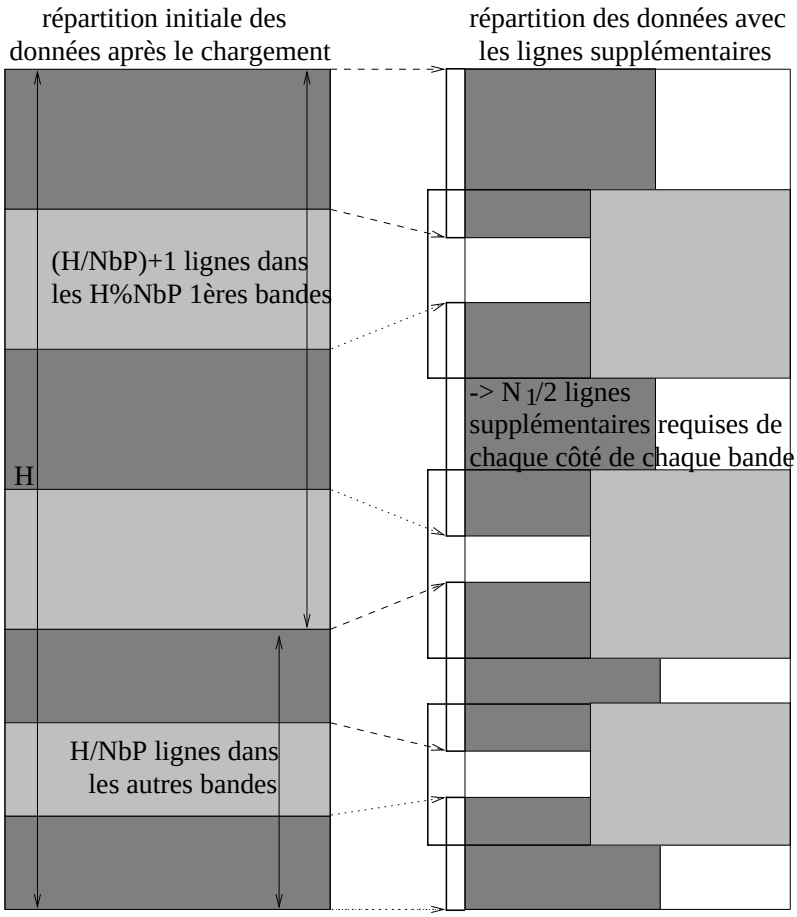

FIG. 3.17 - *Répartition naturelle des données selon la direction verticale pour la première corrélation dans le cas de 7 processeurs*

On peut noter que l'on ne rajoute pas de lignes au début de la première bande et à la fin de la dernière bande. Cela vient du fait que les lignes de l'image droite recalée que l'on est sensé ajouter ne sont simplement pas disponibles dans les zones blanches (en haut et en bas) puisque ces lignes n'ont pu être traitées lors de la première corrélation. De plus, toujours à cause de la hauteur des fenêtres de corrélation, les $\lfloor \frac{N_2}{2} \rfloor$ premières lignes de la première bande et dernières lignes de la dernière bande ne pourront être traitées lors de la seconde corrélation. Au total, on en conclue que les $\lfloor \frac{N_1}{2} \rfloor + \lfloor \frac{N_2}{2} \rfloor$ premières et dernières lignes des images ne produiront pas de points tridimensionnels. D'ailleurs, en faisant la même analyse sur les colonnes des images en prenant en plus de la largeur des fenêtres, la plage de recherche des candidats et le coefficient de réduction horizontale des images pour la première corrélation, on arrive à une perte de $\lfloor \frac{N_2}{2} \rfloor + R*(\lfloor \frac{N_1}{2} \rfloor + \lfloor \frac{P_1}{2} \rfloor)$ colonnes au début et à la fin de chaque ligne. Le MNT final sera donc toujours un peu moins haut et un peu moins large que les images initiales.

Finalement, cette stratégie fonctionne relativement bien mais son principal inconvénient est la possibilité d'avoir des déséquilibres importants entre les quantités de données réellement traitées sur chaque processeur, et donc entre les charges de travail de ceux-ci. Ces différences sont visibles

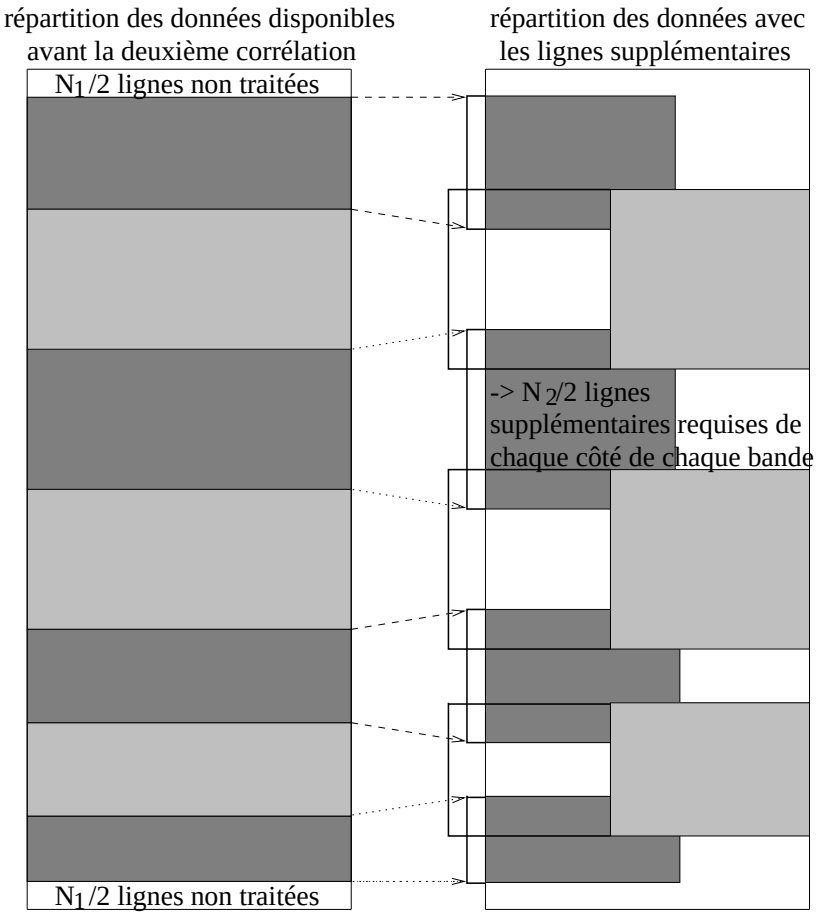

FIG. 3.18 - *Répartition des données pour la seconde corrélation dans le cas de 7 processeurs*

dans les figures 3.17 et 3.18 si l'on regarde la hauteur des bandes. Cela vient du fait que cette répartition des données ne prend pas en compte la répartition du travail sur les processeurs. C'est pourquoi nous avons décidé de modifier notre stratégie de répartition des données en nous basant sur un équilibrage des charges de travail.

3.3.3 Répartition équitable du travail

Le but de cette méthode est de distribuer les données de façon à répartir équitablement sur les processeurs, les charges de travail relatives à ces données. Pour cela, nous devons d'abord définir la charge de travail élémentaire qui correspondra à la plus petite charge de travail assignable à un processeur. Ensuite, nous aurons à considérer la répartition spatiale des données correspondant à ces unités de travail pour déduire une partition des images qui réalise l'équilibrage.

Charge élémentaire de travail

Si l'on regarde le processus d'appariement jusqu'à la deuxième corrélation, on constate que la quantité de travail à effectuer pour une ligne épipolaire est indépendante du contenu de celle-ci (valeurs des pixels). Les lignes épipolaires sont donc toutes équivalentes en terme de calculs. De plus, l'indépendance relative du traitement des lignes épipolaires autorise une division verticale des images. Enfin, la nature globale de l'appariement à l'intérieur d'une ligne empêche toute division horizontale des images. On arrive donc logiquement à la conclusion que l'unité de travail la plus adaptée correspond au traitement d'une ligne épipolaire.

Dans les paragraphes suivants, nous décrivons plusieurs redistributions de données agissant à différents niveaux de la chaîne de traitement vue au paragraphe 3.2.4. Par conséquent, celles-ci sont indépendantes et peuvent être utilisées conjointement ou séparément pour donner un algorithme complètement ou partiellement équilibré.

Première corrélation

Si l'on reprend la répartition donnée dans la figure 3.17 pour la première corrélation, on a vu que les $\lfloor \frac{N_1}{2} \rfloor$ premières et dernières lignes des images ne peuvent être traitées, réduisant ainsi la zone de calculs effectifs aux lignes dans l'intervalle $[\lfloor \frac{N_1}{2} \rfloor \ldots H - \lfloor \frac{N_1}{2} \rfloor]$. Par conséquent, la meilleure manière d'équilibrer le travail de cette étape de l'algorithme est de diviser *cet* intervalle en NbP bandes de même taille plutôt que la hauteur totale des images, puisqu'il représente la quantité totale de travail. Une fois que chaque bande est associée à un processeur, celui-ci doit traiter *toutes* les lignes de cette bande d'image. Il faut donc, de la même manière que pour la méthode simpliste de répartition des données, rajouter $\lfloor \frac{N_1}{2} \rfloor$ lignes de part et d'autre de chaque bande. Il est important de bien faire la différence entre les bandes d'image relatives à la répartition du travail (lignes effectivement traitées) et les bandes correspondantes (avec les lignes supplémentaires) associées aux données nécessaires pour réaliser ces calculs. Dans le cas de notre première approche de répartition, cela n'était pas vraiment le cas puisque les deux bandes extrêmes, par exemple, contenaient initialement aussi bien des lignes appariables que non appariables. Finalement, le schéma de partitionnement résultant est donné dans la figure 3.19, où H' est le nombre de lignes effectivement traitées, et les symboles / et % ont les mêmes significations que dans la figure 3.17.

Étant donné que toutes les tâches de la chaîne de traitement entre la première et la seconde corrélation ne requièrent qu'une ligne de données à la fois, et demandent le même temps de calcul pour traiter chaque ligne (calculs indépendants des valeurs effectives des données), on s'aperçoit que la répartition réalisée ci-dessus est valable (équilibre le travail) pour toutes les étapes avant la seconde corrélation.

Nous montrons maintenant que ce n'est plus le cas lorsque l'on arrive à la seconde corrélation et qu'il faut donc revoir la répartition du travail et des données associées.

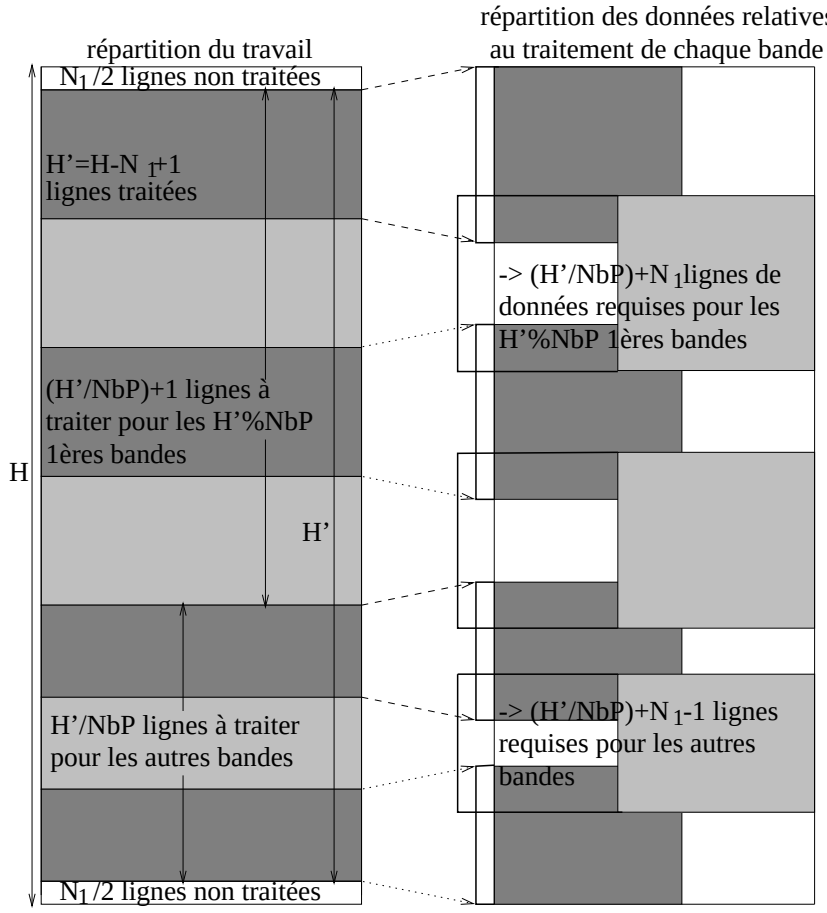

FIG. 3.19 - *Répartition équitable du travail et des données correspondantes pour la première corrélation dans le cas de 7 processeurs*

Seconde corrélation et calcul des disparités finales

On a déjà vu que lors de la seconde corrélation les $\lfloor \frac{N_2}{2} \rfloor$ premières et dernières lignes de l'image gauche et de l'image droite recalée ne pouvaient être traitées à cause du problème des fenêtre de corrélation. Et comme l'image droite recalée n'est définie que sur l'ensemble des lignes appariées lors de la première corrélation, cela implique qu'il faut ôter les $\lfloor \frac{N_2}{2} \rfloor$ premières et dernières lignes de cet ensemble pour trouver les lignes à traiter dans la deuxième corrélation. Finalement, cela revient à effectuer le même type d'équilibrage que précédemment mais cette fois sur l'ensemble des données requises pour cette étape d'appariement et de calcul des disparités finales. Ces données sont les disparités approchées et les lignes de l'image gauche et de l'image droite recalée, et elles sont organisées sur les processeurs exactement comme l'indiquent les bandes grises de la colonne de gauche de la figure 3.19. On obtient donc une seconde redistribution du travail et des données, confrontée dans la figure 3.20 à celle de la première corrélation pour mettre en évidence les déplacements des bandes. Dans cette figure, H'' dénote le nombre de lignes effectivement appariées et les mouvements de données à faire sont visibles entre les deux colonnes de droite.

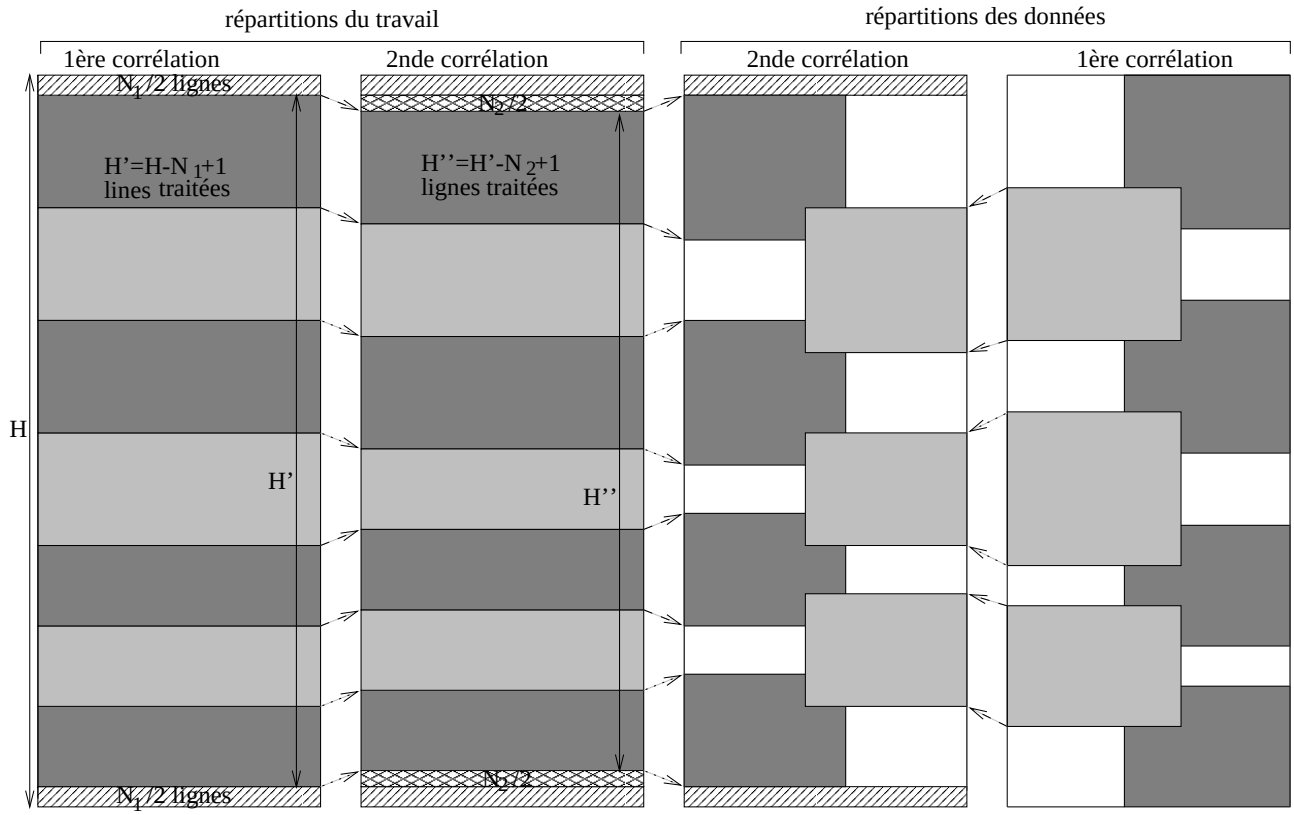

FIG. 3.20 - *Répartition équitable du travail et des données correspondante pour la seconde corrélation dans le cas de 7 processeurs, et comparaison avec la première répartition*

Une fois les données redistribuées convenablement, la corrélation et le calcul des disparités finales peuvent être effectués avec approximativement la même quantité de travail sur les processeurs. Ensuite, il nous reste à reconstruire dans un repère tridimensionnel les couples de points correspondants. Voyons maintenant comment on peut équilibrer ce travail sur les processeurs.

Reconstruction 3D

Dans cette étape, l'équivalence des lignes en quantité de travail demandé n'est plus vrai à cause du nombre variable d'appariements valides réalisés d'une ligne épipolaire à l'autre. La répartition réalisée à l'étape précédente n'est donc plus adaptée. Néanmoins, on peut redéfinir l'unité de travail par le traitement élémentaire effectué dans cette étape pour en faciliter l'équilibrage. Ce traitement correspond au calcul des coordonnées 3D pour un point stéréoscopique. L'équilibrage des charges se fera donc cette fois ci en redistribuant les couples de points appariés sur les processeurs. Une façon simple de faire revient à construire NbP *paniers* sur chaque processeur, qui représentent chacun un processeur destination, puis de remplir équitablement ces paniers en leur ajoutant chacun leur tour un point stéréoscopique à la fois. Enfin, une multi-distribution permet de réarranger les points en fonction de leur destination et d'obtenir une répartition homogène sur les processeurs.

Comme expliqué dans le paragraphe 3.2.4, l'échantillonnage du MNT n'a pas été intégrée dans la version séquentielle de la chaîne de traitement, pour éviter de refaire les calculs de reconstruction 3D à chaque fois. Cependant, lorsque nous avons fait les premiers tests de notre algorithme parallèle de reconstruction, nous avons constaté qu'il était relativement rapide. De plus, on sait que le temps d'échantillonnage du MNT, qui est fonction du nombre de points 3D calculés, devient relativement long pour de grandes images. Ainsi, nous avons pensé qu'il serait finalement intéressant d'intégrer cette étape de manière optionnelle à notre algorithme parallèle.

Échantillonnage du MNT

Si l'on prend une grille régulière donnée, le calcul d'un cellule est indépendant des autre cellules de la grille. De plus, le temps de calcul d'une cellule dépend du nombre de points 3D présents dans celle-ci, ce qui est variable d'une cellule à l'autre. La solution idéale pour équilibrer cette étape serait de distribuer les cellules de manière à avoir approximativement le même nombre de points sur chaque processeur. Or, cela exigerait une connaissance globale de tous les points et l'utilisation d'une heuristique de répartition impliquant des communications et des calculs supplémentaires. Finalement, nous utilisons la même méthode que celle utilisée pour équilibrer les corrélations : nous divisons la grille du MNT en bandes horizontales que nous redistribuons sur les processeurs. Cela demande juste un tri local des points en fonction de l'axe des lignes (axe Oy), puis une multi-distribution. Bien sur, ce schéma n'est pas parfait mais il a l'avantage d'être rapide et relativement bon dans la plupart des cas. Nous pouvons raisonnablement espérer une répartition homogène des points sur la grille en général si la taille des cellules est relativement petite.

Ainsi s'achève la description de l'algorithme final de reconstruction tridimensionnelle de terrain en parallèle. Dans cette description, nous avons eu recours à plusieurs reprise à des communications entre les processeurs pour échanger des données. Nous donnons maintenant une description de l'environnement de programmation parallèle que nous avons utilisé pour implanter l'algorithme sur machine.

PPCM

Nous ne tentons pas ici de donner une description détaillée de l'environnement PPCM mais nous donnons ses principales caractéristiques. Nous recommandons la lecture de [CBF+92], [CLM92] ou encore [FMP95] pour des informations plus complètes. PPCM (*Portable Parallel Communication Module*) est une bibliothèque C de communication entre processeurs basée sur le modèle MIMD, qui a été développée depuis 1991 par l'équipe d'imagerie du LIP dans le cadre d'un projet d'imagerie médicale. Le but était de fournir un environnement de programmation parallèle indépendant de la machine cible, exactement dans la même optique que les standards actuels que sont PVM et MPI, mais orienté vers les applications d'imagerie numérique. Au cours des années, PPCM a subi de nombreuses modifications et ajouts qui en font actuellement un environnement fiable et efficace mais surtout intégrant des fonctionnalités puissantes pour faciliter l'équilibrage des charges. Les

trois caractéristiques principales de PPCM sont :

- la portabilité sur plusieurs machines parallèles, PVM et MPI,
- l'intégration de fonctions de calcul et de redistribution des données spécifiques à l'équilibrage des charges,
- la disponibilité de plusieurs topologies virtuelles pour décrire le réseau de communication entre les processeurs.

Concernant la portabilité, PPCM est utilisable sur les machines suivantes : Volvox (Archipel), Paragon (Intel), Capitan (Matra), T3D et T3E (Cray), ainsi que PVM et MPI. Pour ce qui est des communications, la bibliothèque offre en plus des fonctions classiques (*point à point, distributions, multi-distributions, réductions, ...*) d'autres primitives de communication adaptée à la modification d'un partitionnement d'un ensemble de données. Par exemple, il existe un module nommé *ParList* qui permet de modifier la répartition de données rectilinéaires en admettant des chevauchements entre les sous-ensembles de données (décrits par des intervalles) tout en minimisant le volume de données déplacées. C'est d'ailleurs ce module qui est utilisé pour les redistributions avant les deux corrélations. En effet, on dispose à ces moments là d'un ensemble de lignes ordonnées linéairement que l'on veut redistribuer avec certaines duplications au niveau des frontières entre processeurs, ce qui correspond exactement à ce que réalise ParList. Ce type de fonction facilite énormément le travail du programmeur car dans ce cas précis, on voit que celui-ci a juste à calculer les indices de début et de fin de chacun des nouveaux intervalles sur l'ensemble puis la fonction s'occupe de faire les communications. Pour que ces communications puissent être optimisées, il faut que les processeurs soient organisés selon une topologie linéaire. Et comme cela n'est pas forcément le cas sur les machines parallèles que l'on est susceptible d'utiliser, PPCM propose tout en ensemble de topologies virtuelles utilisables sur n'importe quelle topologie réelle. Citons parmi celles-ci : l'anneau, la grille, l'hypercube et dernièrement le graphe complet. En ce qui concerne les autres redistributions de données dans notre algorithme, nous utilisons une procédure qui pour chaque processeur diffuse les données locales vers leurs destinations respectives en utilisant une structure de messages multiples puis qui rassemble les données provenant des autres processeurs.

Voici donc terminée la description rapide de notre environnement de programmation parallèle. Nous reviendrons sur certains aspects de celui-ci dans la suite et notamment sur les fonctions d'équilibrage au chapitre 4 puis sur les topologies virtuelles dans le chapitre 5. Nous présentons dans ce qui suit, les résultats expérimentaux obtenus avec nos nouveaux algorithmes. Nous en dégageons une comparaison entre la version séquentielle de Memier et notre version optimisée, ainsi qu'une interprétation du comportement de notre algorithme parallèle dont nous construisons un modèle partiel d'exécution.

3.4 Résultats

Lorsque nous avons abordé la qualité des résultats au paragraphe 3.2.2, nous avons montré que notre algorithme augmentait la densité des terrains et nous avons dit qu'il conservait la précision

des calculs. Pour bien se rendre compte de la véracité de ce deuxième point, nous montrons un exemple de reconstruction à partir d'images épipolaires complètes, de taille 5650×6000, que l'on peut voir en page 60. Le terrain reconstruit est donné en page 61, où l'image du haut représente une vue de dessus avec illumination, et l'image du bas est une visualisation 3D du terrain avec sa texture réelle. Cet exemple confirme bien la qualité générale de notre reconstruction (densité et précision). Notre algorithme permet en effet d'obtenir des terrains en haute résolution tout en conservant une densité et une précision élevées.

L'ordre de présentation des performances reprend l'ordre de conception des algorithmes, c.-à-d. la version séquentielle optimisée puis sa version parallèle.

3.4.1 Algorithme séquentiel

La comparaison des temps d'exécution du programme de Memier et de notre nouvel algorithme permet d'avoir une idée précise des améliorations apportées décrites au paragraphe 3.2.1. Les données utilisées sont deux images épipolaires de 1951×1951 pixels. La taille des fenêtres de corrélation est de 17×17 pixels pour les deux corrélations. Le nombre de candidats possibles est fixé à 15 pour la première corrélation et à 3 pour la seconde. Enfin, le facteur de réduction des images lors de la première corrélation est 4. Le tableau 3.1 donne les temps des deux versions exprimés en secondes étape par étape. Ces temps ont été obtenus sur une station SUN SPARC serveur 1000.

Étapes	Version de Memier	Version optimisée
initialisation	/	5
$1^{\text{ère}}$ corrélation	3160	860
parcours du graphe	151	35
interpolation	514	7
rééchelonnement et recalage	38	7
2nde corrélation	14231	969
reconstruction 3D	195	992
TOTAL	**18289**	**2879**

TAB. 3.1 - *Temps (en secondes) des différentes étapes des algorithmes séquentiels*

Dans ce tableau, on peut voir que notre version optimisée est bien plus rapide que la version initiale. Bien que presque toutes les étapes soient accélérées, les améliorations les plus importantes se trouvent au niveau des deux corrélations et de l'interpolation. De plus, l'étape supplémentaire d'initialisation de notre chaîne de traitement pour remplir les tampons de lignes n'est pas très coûteuse et ne détériore en rien les performances de notre algorithme. Enfin, si l'on regarde les rapports des temps entre les deux versions pour les corrélations, on constate qu'il y a bien un rapport proche $\frac{N_1}{R} = \frac{17}{4} = 4.25$ pour la première corrélation et un rapport proche de $N_2 = 17$ pour la deuxième corrélation, ce qui confirme bien les résultats théoriques de notre étude des complexités du paragraphe 3.2.1. Enfin, on remarque que le temps de la dernière étape est plus important dans notre version que dans celle de Memier. Cela vient simplement du fait que notre algorithme de

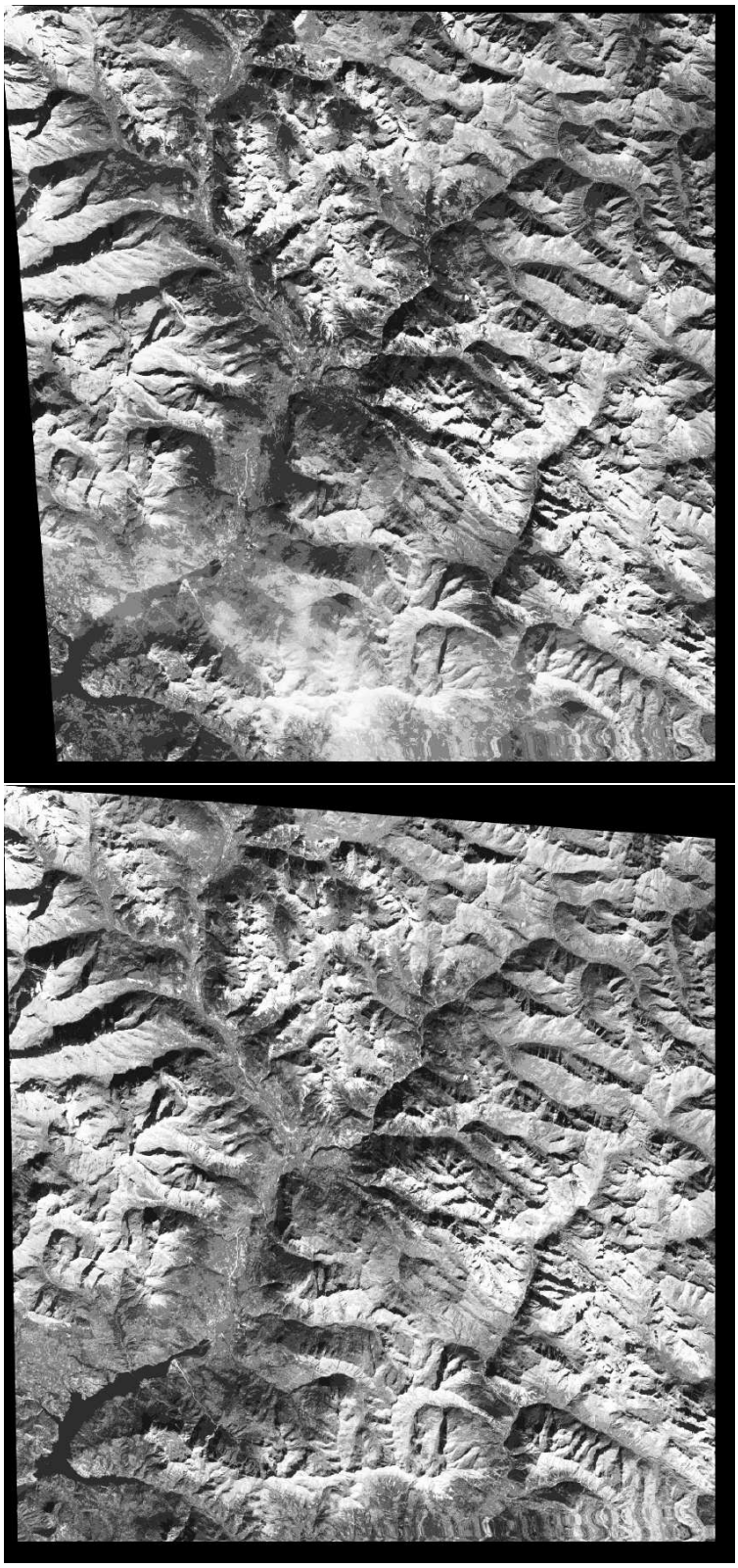

FIG. 3.21 - *Images épipolaires de la région de Briançon. Image gauche en haut, image droite en bas*

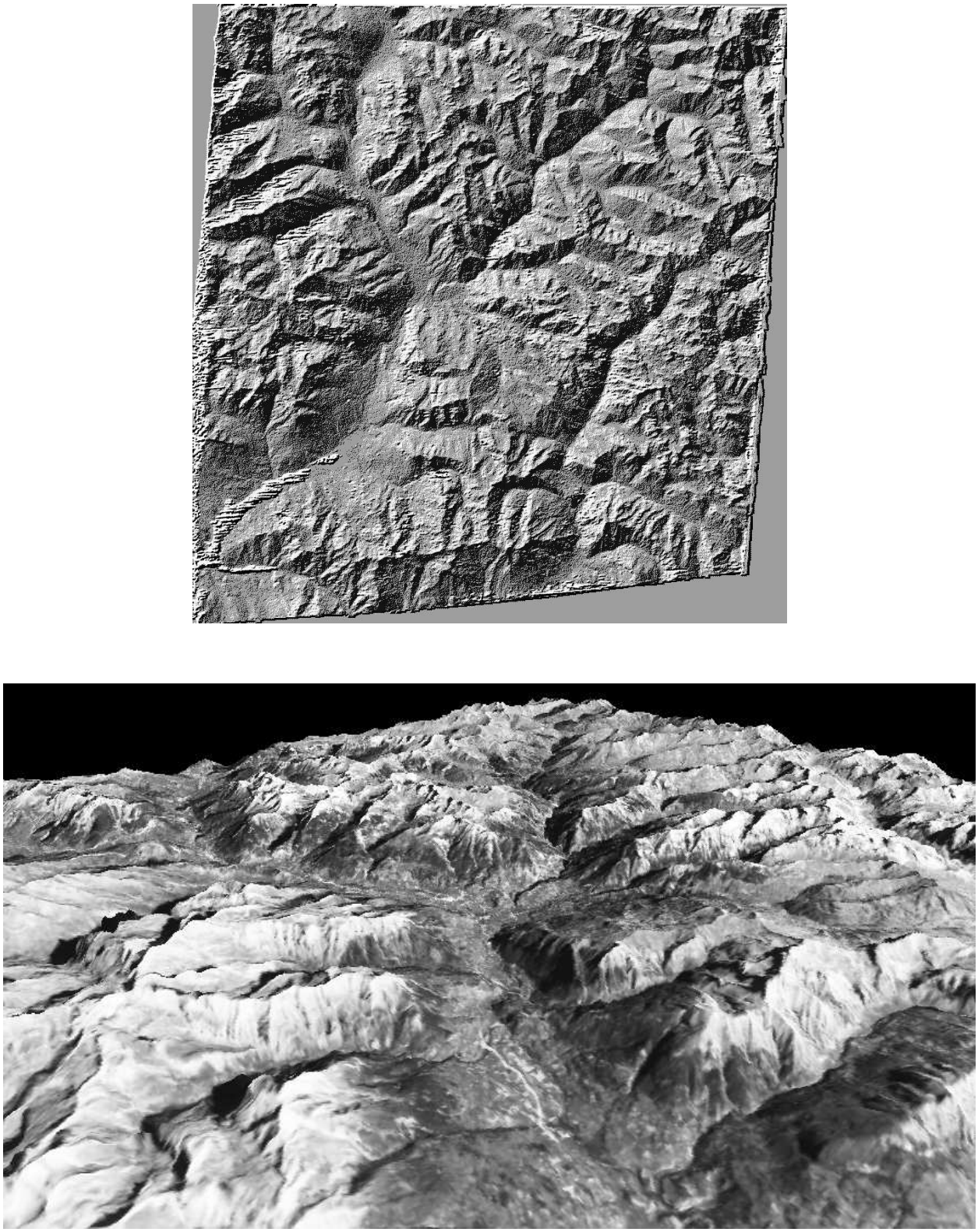

FIG. 3.22 - *Terrain reconstruit. Vue ombrée en haut, visualisation 3D avec texture en bas*

mise en correspondance génère beaucoup plus d'appariements et donc que le nombre de points à reconstruire en 3D est plus important.

Ces résultats confirment que les optimisations d'algorithmes séquentiels et le chaînage direct des différentes étapes d'un processus sont vraiment efficaces. Voyons maintenant si l'utilisation du parallélisme permet de diminuer d'avantage ces temps d'exécution.

3.4.2 Algorithme parallèle

Pour pouvoir comparer la version séquentielle et la version parallèle, nous utilisons des données de même taille que précédemment avec les mêmes valeurs des paramètres à part la plage de recherche de la première corrélation qui est cette fois de 14 pixels au lieu de 15 (différence négligeable en fait). Les expérimentations ont été réalisées sur une machine Cray T3E. Elle est composée de 256 processeurs DEC-α EV5.6 à 375 Mhz possédant chacun 128 Mo de mémoire vive, et reliés entre eux par un réseau à 100 Mbits/s avec une topologie de tors 3D.

Étant donné que notre version parallèle intègre l'étape d'échantillonnage du MNT, notre étude des performances est essentiellement basée sur le calcul complet du MNT à partir des deux images épipolaires.

Une des caractéristiques les plus importantes pour un programme parallèle est la bonne répartition de la charge de travail sur les processeurs. De cet équilibrage dépend directement les performances du programme et son bon comportement en fonction du nombre de processeurs. Nous avons vu lors de la description de notre algorithme toutes les opérations de redistribution des données effectuées dans ce but. Il faut maintenant vérifier que celles-ci soient réellement efficaces. L'histogramme présenté dans la figure 3.23 donne les temps d'exécution du programme sur chaque processeur pour une configuration de 16 éléments. On peut voir que ces temps sont réellement très proches les uns des autres ce qui confirme l'efficacité de note équilibrage et nous promet une bonne extensibilité du programme avec le nombre de processeurs.

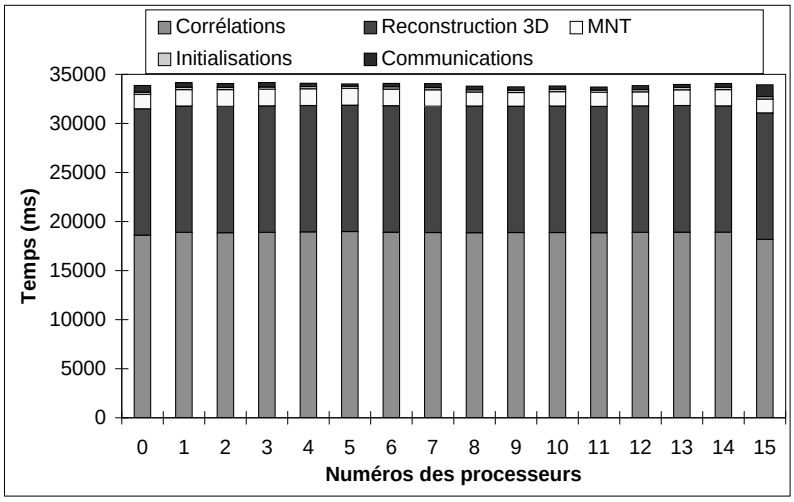

FIG. 3.23 - *Temps d'exécution sur chaque processeur pour une configuration à 16 éléments*

Une autre caractéristique importante d'un programme parallèle est son extensibilité en fonction du nombre de processeurs, aussi appelée accélération (*speed up*). Cela correspond à la capacité du programme à s'adapter efficacement à différentes configurations de processeurs pour toujours donner des performances maximales. Nous rappelons que l'accélération est définie par :

$$A = \frac{Temps\ sur\ un\ processeur}{Temps\ sur\ NbP\ processeurs}$$

De plus, on peut remarquer que si l'on veut que chaque processeur ait au moins une ligne de données à traiter, il faut que l'on utilise au plus un nombre de processeurs égale au nombre de lignes appariables dans la seconde corrélation, donc $H - N_1 - N_2 + 2$ processeurs (N_1 et N_2 sont toujours impairs). Néanmoins, pour faire ce calcul d'accélération, on utilise le temps sur un processeur comme temps de référence. Or, dans notre programme parallèle, l'étape d'échantillonnage du MNT requiert d'avoir en mémoire locale tous les points 3D se trouvant dans la bande du MNT assignée au processeur. La quantité de mémoire mise en jeu reste tout à fait raisonnable lorsque l'on utilise plusieurs processeurs mais devient trop importante si l'on en utilise qu'un seul avec de grandes images. Nous avons donc été obligé de modifier notre programme parallèle pour qu'il fonctionne différemment sur un processeur en stockant seulement un sous-ensemble des points à la fois. Mais cela donne une exécution plus lente que la normale et nous empêche donc d'utiliser les temps sur un processeur comme référence. C'est pourquoi, dans les figures suivantes, le temps sur un processeur n'est pas montré et les accélérations sont calculées relativement au temps sur deux processeurs. Néanmoins, pour nous assurer de la cohérence de cette méthode de calcul, nous avons comparé les accélérations absolues et relatives du programme sans échantillonnage du MNT (où le temps sur un processeur est valide), et nous obtenons des valeurs quasi identiques. En fait, ce calcul relatif est correct car l'accélération réelle est linéaire avec deux processeurs, nous pouvons donc raisonnablement penser que c'est aussi le cas lorsque l'on intègre l'échantillonnage du MNT. En effet, lorsque l'on utilise peu de processeurs, cet échantillonnage ajoute plus de calculs purs au programme que de surcoûts de parallélisme. Par conséquent, la figure 3.24 montre les accélérations pour les deux exécutions du programme avec ou sans échantillonnage du MNT, en fonction du nombre de processeurs.

On peut faire deux remarques importantes sur ce graphe : d'une part, les deux courbes sont très proches donc la dernière étape n'influe quasiment pas sur l'extensibilité du programme, et d'autre part, ces courbes sont bonnes dans la première moitié du graphe mais s'infléchissent considérablement lorsque l'on atteint un très grand nombre de processeurs (au-delà de 128 processeurs). Cela vient du fait que nous utilisons un parallélisme à gros grain dans notre approche (SPMD) or, lorsque l'on augmente le nombre de processeurs de manière très importante on se retrouve dans un cas de parallélisme à grain fin (SIMD). Nous obtenons donc sur chaque processeur une charge de travail effective très réduite (quelques lignes d'image) alors que les opérations d'équilibrage représentent toujours la même quantité de travail et des communications de plus en plus coûteuses. En effet, puisque les bandes allouées à chaque processeur sont très petites à chaque étape, lors de chaque redistribution, les nouvelles données assignées à un processeur ont peu de chances d'être déjà dans la mémoire locale de celui-ci, ce qui entraîne une augmentation du volume de données déplacées. Ainsi, les surcoûts dus au parallélisme deviennent plus importants que les calculs de reconstruction et les performances sont alors dégradées. Cela apparaît très nettement sur la figure 3.25 où la ré-

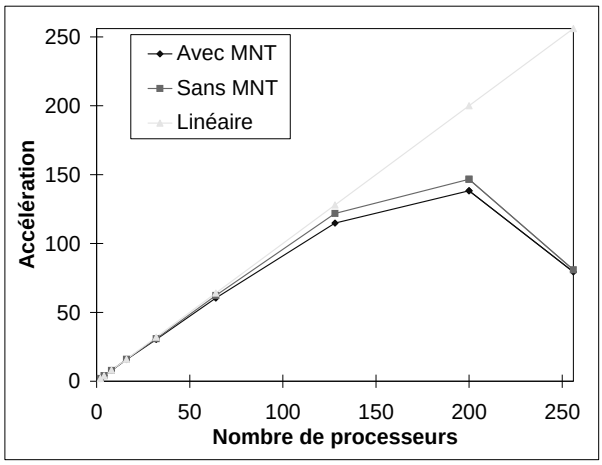

FIG. 3.24 - *Accélérations en fonction du nombre de processeurs pour le calcul avec ou sans échantillonnage du MNT sur des images* 1951×1951

partition des temps de calculs sur chaque processeur est donnée pour différentes configurations de processeurs. Dans cette figure, les *communications* incluent le calcul des partitions et la construction des messages.

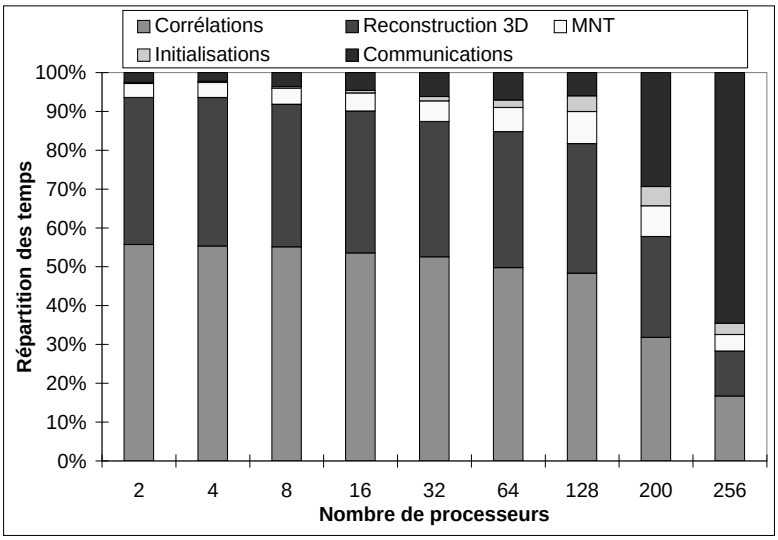

FIG. 3.25 - *Répartition des temps de calculs sur chaque processeur pour différentes configurations*

On constate bien que les surcoûts du parallélisme, principalement la phase d'initialisation et les communications augmentent considérablement avec le nombre de processeurs utilisés. Concernant les communications, en plus du volume de données déplacées plus important, l'organisation interne des processeurs intervient aussi. En effet, certaines des fonctions de redistribution que nous utilisons (par exemple ParList) sont prévues pour fonctionner sur un réseau linéaire de processeur (ligne ou anneau) alors que le réseau réel de la Cray T3E est un tors 3D. La topologie virtuelle que nous utilisons (l'anneau) ne correspond donc pas à la topologie réelle et deux processeurs voisins logiquement ne le seront peut être pas physiquement. Nous contraignons donc les données à suivre

des chemins qui ne sont pas forcément optimaux dans le réseau physique. Et comme la quantité de messages augmente, tout cela contribue à généré des embouteillages de données (*bottlenecks*) qui ralentissent de manière non négligeable les communications. La solution idéale à ce problème serait de réassigner les numéros logiques des processeurs de manière à faire coïncider l'anneau sur le tors 3D pour conserver une cohérence topologique entre les deux réseaux. Malheureusement, cela n'est pas possible en pratique car sur la machine Cray nous n'avons pas de contrôle ni d'accès direct aux numéros physiques des processeurs et donc à leur emplacement dans le tors. Bien sûr, on peut alors envisager l'utilisation d'une autre machine parallèle. Néanmoins, comme on peut le voir sur les figures 3.24 et 3.25, nos problèmes n'apparaissent qu'avec plus de 128 processeurs, et à l'heure actuelle, il n'existe encore pas beaucoup de machines parallèles avec autant de processeurs et qui de plus soit disponible. Enfin, nous pouvons espérer obtenir de meilleurs résultats avec des images plus grandes (6000×6000) car cela impliquerait plus de calculs effectifs sur chaque processeur et réduirait l'influence des surcoûts repoussant ainsi l'inflexion des courbes. Pour vérifier cela, nous avons fait une expérimentation sur des images 5650×6000, avec $N_1 = 9$, $N_2 = 7$, $R = 4$, $P_1 = 20$, $P_2 = 4$. La courbe d'accélération que nous obtenons (figure 3.26) est nettement meilleure que la précédente.

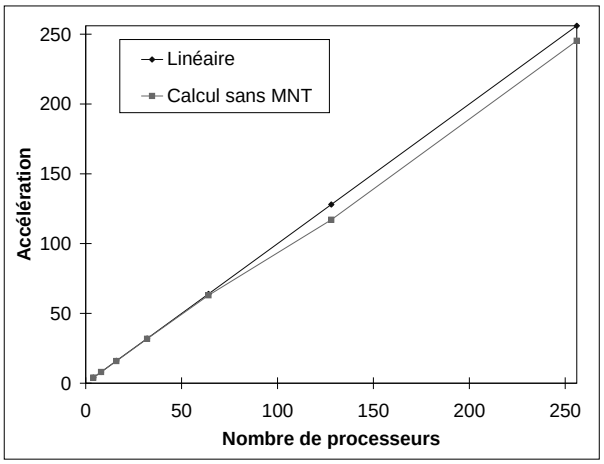

FIG. 3.26 - *Accélérations en fonction du nombre de processeurs pour le calcul sans échantillonnage du MNT sur des images* 5650×6000

Finalement, pour donner une idée plus précise des performances absolues de notre programme parallèle, nous donnons dans la figure 3.27 les temps séparés et le temps total en fonction du nombre de processeurs sur une échelle logarithmique, pour l'exemple des images 1951×1951.

Cette figure a l'avantage de mettre en évidence les parties non extensibles de l'algorithme. On remarque que les courbes des corrélations, de la reconstruction 3D et de l'échantillonnage du MNT sont quasiment des droites décroissantes avec une pente proche de -2, ce qui sur une échelle logarithmique correspond bien à des accélérations quasi linéaires. Par contre, les courbes des communications et des initialisations sont bien plus chaotiques et remontent même au-delà de 128 processeurs. On constate d'ailleurs l'influence directe des ces courbes sur le temps total. Néanmoins, cette figure indique aussi que nous obtenons des performances absolues plutôt bonnes puisque le temps minimal est au dessous des 7 secondes pour le traitement complet des images

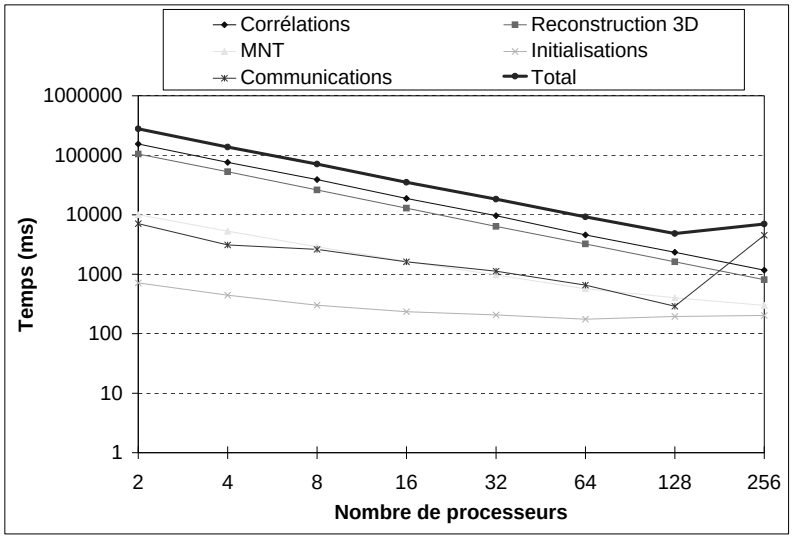

FIG. 3.27 - *Temps d'exécution séparés et total en fonction du nombre de processeurs, pour les images* 1951×1951

1951×1951. Cela est seulement un peu plus de 3000 fois plus rapide que la version initiale de Memier qui n'intègre pas l'échantillonnage du MNT.

3.4.3 Modèle d'exécution

D'après les résultats précédents, on s'aperçoit que le meilleur temps obtenu pour un ensemble d'images et de paramètres particuliers ne l'est pas forcément avec le plus grand nombre de processeurs. On aimerait donc pouvoir évaluer au mieux les temps d'exécution du programme et en déduire le nombre optimal de processeurs à utiliser avec des données précises pour obtenir les meilleures performances. Pour cela, on se propose de concevoir un modèle théorique d'exécution de notre algorithme parallèle. Nous procédons en évaluant la complexité de chaque étape de la chaîne de traitement. Ensuite, nous pouvons obtenir le modèle correspondant à une machine donnée en étalonnant ces complexités avec les temps obtenus sur des exemples tests. Dans notre modèle, le temps effectif et la complexité sont liés par un facteur réel (α_k), que la phase d'étalonnage permet d'évaluer. Ainsi, la liste ci-dessous donne les modèles d'exécution pour chaque étape :

- première corrélation, on reprend la complexité donnée dans 3.2.1 pour une ligne en multipliant par le nombre total de lignes traitées :

$$\begin{aligned} C_1 &= \alpha_{C_1} \left(\overbrace{\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor}^{lignes\ traitées} \right) \left[\overbrace{P_1.N_1^2}^{premier\ point} + \overbrace{P_1.N_1}^{autres\ points} . \left(\overbrace{\left\lfloor \frac{L}{R} \right\rfloor - N_1 - P_1 + 1}^{faible\ résolution} \right) \right] \\ &= \alpha_{C_1} \left(\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor \right) .P_1.N_1. \left(\left\lfloor \frac{L}{R} \right\rfloor - P_1 + 1 \right) \end{aligned} \quad (3.12)$$

- parcours du graphe, on doit calculer toutes les cases du tableau 2D pour chaque pixel de référence et pour toutes les lignes :

$$G = \alpha_G \left(\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor \right) . \overbrace{P_1}^{hauteur\ du\ graphe} . \left(\overbrace{\left\lfloor \frac{L}{R} \right\rfloor - N_1 - P_1 + 1}^{largeur\ du\ graphe} \right) \tag{3.13}$$

- interpolation des disparités et recalage de l'image droite :

$$\begin{aligned} IR &= \alpha_{IR} \left(\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor \right) (\overbrace{2R - 2}^{interpolation} + \overbrace{R}^{recallage}) \left(\left\lfloor \frac{L}{R} \right\rfloor - N_1 - P_1 + 1 \right) \\ &= \alpha_{IR} \left(\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor \right) (3R - 2) \left(\left\lfloor \frac{L}{R} \right\rfloor - N_1 - P_1 + 1 \right) \end{aligned} \tag{3.14}$$

- deuxième corrélation, même complexité que la première mais avec ses propres paramètres et les lignes en haute résolution :

$$\begin{aligned} C_2 &= \alpha_{C_2} \left(\overbrace{\left\lfloor \frac{H - N_1 - N_2 + 2}{NbP} \right\rfloor}^{lignes\ traitées} \right) \left[\overbrace{P_2.N_2^2}^{premier\ point} + \overbrace{P_2.N_2}^{autres\ points} .\overbrace{(L - (N_1 + P_1 - 1).R - N_2 - P_2 + 1)}^{haute\ résolution} \right] \\ &= \alpha_{C_2} \left(\left\lfloor \frac{H - N_1 - N_2 + 2}{NbP} \right\rfloor \right) .P_2.N_2.(L - (N_1 + P_1 - 1).R - P_2 + 1) \end{aligned} \tag{3.15}$$

- reconstruction 3D, la complexité est bornée par le nombre de pixels par ligne :

$$R3D = \alpha_{R3D} \left(\left\lfloor \frac{H - N_1 - N_2 + 2}{NbP} \right\rfloor \right) (L - (N_1 + P_1 - 1).R - N_2 - P_2 + 1) \tag{3.16}$$

- échantillonnage du MNT :

$$MNT = \alpha_{MNT} \left(\left(2.L. \left\lfloor \frac{H}{NbP} \right\rfloor .\beta_{MNT} \right) + (\left\lceil \frac{L}{ratio} \right\rceil \left\lceil \frac{\frac{H}{ratio}}{NbP} \right\rceil) (ratio^4) \right) \tag{3.17}$$

où β_{MNT} représente le pourcentage de pixels corrélés
et *ratio*, le rapport entre le pas d'échantillonnage du MNT (en mètres) et la résolution de l'image (en mètres/pixel)

Comme nous l'avons indiqué précédemment, les α_k sont évalués en utilisant les résultats d'un exemple test. Les évaluations que nous obtenons sont plus ou moins bonnes selon l'étape de calcul et les paramètres initiaux. Dans le tableau 3.2, nous indiquons les pourcentages d'erreur de notre estimation du temps de calcul en fonction du nombre de processeurs, pour l'exemple des images 5650×6000 en échantillonnant le MNT avec un ratio de 2 (MNT à 20 mètres). Il est à noter

Nombre de processeurs	Évaluation (ms)	Réel (ms)	Erreur (%)
4	1194950	1299790	-8,06
8	597475	623568	-4,03
16	298538	307206	-2,82
32	149220	151487	-1,5
64	74191	75519	-1,75
128	36716	31266	17,43

TAB. 3.2 - *Pourcentages d'erreur du modèle par rapport à la réalité en fonction du nombre de processeurs*

que nous ne prenons pas en compte les communications, par conséquent, les temps donnés ici représentent la somme des temps des différentes étapes de calcul décrites ci-dessus.

On peut voir que les erreurs commises sont plus ou moins importantes et le temps total des calculs purs est estimé avec une précision relativement correcte. Néanmoins, ce modèle est incomplet car il ne prend pas en compte la phase d'initialisation (qui reste négligeable) et surtout les communications. Cela vient essentiellement du fait que les temps des communications ne peuvent être modélisés efficacement car ils ne varient pas de manière complètement déterministe alors qu'ils jouent un rôle important dans le temps total, surtout quand le nombre de processeurs est élevé. Néanmoins, notre modèle va quand même nous permettre d'évaluer approximativement les temps totaux et donc de déduire quel est le nombre de processeurs optimal à utiliser.

Avec les résultats que nous avons obtenus, nous avons constaté que les meilleurs temps sont généralement obtenus lorsque les échanges de lignes de données se font uniquement entre processeurs logiquement voisins. C'est à dire, quand le nombre de lignes de données par processeur est légèrement supérieur ou égal à la hauteur de la fenêtre de corrélation. Comme on effectue deux corrélations on a donc les deux contraintes suivantes :

$$\left\lfloor \frac{H - N_1 + 1}{NbP} \right\rfloor \geq N_1 \tag{3.18}$$

$$\left\lfloor \frac{H - N_1 - N_2 + 2}{NbP} \right\rfloor \geq N_2 \tag{3.19}$$

Ainsi, pour estimer le nombre optimal de processeurs à utiliser, il suffit de trouver la valeur de NbP qui minimise les expressions à gauche dans les équations (3.18,3.19) tout en respectant ces deux contraintes.

Dans l'exemple des images 1951×1951, utilisé pour les expériences précédentes, on trouve une valeur de 112 processeurs, ce qui est effectivement proche de la meilleure performance absolue que nous avons observée.

3.4.4 Conclusion

Finalement, nous obtenons un algorithme parallèle plutôt efficace. On a pu constater la bonne répartition des charges de travail ainsi qu'une extensibilité relativement correcte. La principale faiblesse de notre algorithme se situe dans les communications de données qui augmentent avec le nombre de processeurs. Malgré tout, notre modèle théorique d'exécution nous permet d'approcher le nombre optimal de processeurs à utiliser pour un ensemble de données et de paramètres connus.

3.5 Conclusion

Deux algorithmes de reconstruction de terrain utilisant des images stéréoscopiques du satellite SPOT ont été décrits. Le premier est une version améliorée d'un algorithme séquentiel déjà existant et mis au point par Memier. Le second est une version parallèle de cet algorithme optimisé. La comparaison du code initial avec les approches existantes a montré que celui-ci produisait des résultats précis mais souffrait de temps de calcul élevés et d'une grosse consommation de mémoire.

Différentes voies pour améliorer la qualité des résultats, notamment la densité de la carte, ont été étudiés. En particulier, nous avons modifié la fonction d'interpolation pour prendre en compte la possibilité des discontinuités dans le flot de données reconstruites. Néanmoins, d'autres améliorations sont possibles. La plus intéressante serait de prendre en compte la cohérence verticale entre deux lignes épipolaires consécutives dans les images. Cependant, cela impliquerait une réorganisation importante de notre chaîne de traitement et surtout la modification de la gestion mémoire.

Nous avons pu constater dans ce chapitre que les optimisations calculatoires et l'enchaînement des étapes sont réellement efficaces et recommandés pour obtenir des temps d'exécution réduits. Concernant l'algorithme parallèle, les différents surcoûts dus au parallélisme, principalement les communications, ont été exhibés et analysés. Nous avons montré que l'importance de ces surcoûts sur les performances est directement liée à la taille initiale des données (donc des images) ainsi qu'au réseau d'interconnexion des processeurs.

La mise au point d'un modèle théorique de l'exécution du programme parallèle nous permet d'estimer de manière relativement précise le nombre optimal de processeurs à utiliser, en fonction des données initiales, pour obtenir les meilleures performances.

Finalement, nous pouvons conclure que le parallélisme est vraiment bien adapté à ce type de calculs et permet de considérer des traitements rapides sur des images entières produites par SPOT (6000×6000 pixels).

CHAPITRE 4

Algorithme parallèle de visualisation de terrains texturés

Une fois que le relief d'un terrain a été reconstruit pour donner un MNT, celui-ci peut être exploité par les géologues à l'aide de divers traitements tels que les calculs de pente ou de courbure, l'extraction de courbes de niveaux, ou encore la visualisation réaliste du terrain sous différentes incidences. Ces traitements permettent non seulement d'appréhender une région de manière globale en visualisant le terrain complet, mais aussi d'étudier localement certaines caractéristiques géologiques ou topologiques. Néanmoins, vue la taille relativement importante des données à traiter, le recours au parallélisme parait encore une fois très souhaitable. Ainsi, les travaux décrits dans ce chapitre portent sur un algorithme parallèle de visualisation en trois dimensions des terrains reconstruits. L'aspect réaliste est obtenu en utilisant la technique de plaquage de texture (*texture mapping*) avec une image déduite des deux image satellites initiales.

La première partie de ce chapitre présente un rappel de l'algorithme séquentiel de synthèse d'image utilisant la technique bien connue du tampon en profondeur (*Z-buffer*). Ensuite, nous donnons une description rapide d'un algorithme parallèle de Z-buffer, qui a été développé au sein de l'équipe d'imagerie du LIP, dans le cadre d'un projet d'imagerie médicale pour visualiser des surfaces triangulées. Les structures de données et les types de calculs utilisés par cet algorithme étant relativement différents de ceux nécessaires pour les terrains que nous voulons visualiser, nous décrivons dans la seconde partie, notre extension de cet algorithme pour le traitement efficace des MNT. Dans notre étude, nous nous intéressons plus particulièrement à la distribution des données et aux structures des messages utilisées pour minimiser les communications.

4.1 Algorithme séquentiel du Z-buffer

L'intérêt du tampon en profondeur est d'éliminer les faces cachées d'un scène vue d'un point précis de l'espace. L'idée est de garder en mémoire pour chaque pixel de l'image, la distance entre l'observateur et le point 3D le plus proche qui se projette sur ce pixel. Ainsi, lorsque l'on calcule une image I de taille $L \times H$, on initialise un tampon en profondeur P de même taille, avec une valeur infinie en chaque point. Ensuite, on projette dans le référentiel de l'image tous les éléments de base

composants la scène (des triangles en règle générale). Enfin, pour chaque pixel (i, j) de chaque triangle, on compare sa distance à l'observateur $Z(i, j)$ avec la distance stockée à la même position dans le Z-buffer $P(i, j)$. Si $P(i, j) > Z(i, j)$, cela signifie que le nouveau point est plus proche de l'observateur que celui qui était à cet endroit avant, c'est donc le nouveau pixel qui sera vu par l'observateur. Dans ce cas là, on affecte la couleur du pixel (i,j) avec la couleur du nouveau point (celle-ci dépend du type de visualisation donné, en général, on utilise un modèle d'illumination), et on met à jour $P(i, j)$ avec la distance du nouveau point $Z(i, j)$. Dans le cas contraire, c'est le nouveau point qui se trouve derrière l'ancien, il est donc caché et il n'y a alors aucun changement à faire. De cette manière, il est aisé de voir que l'on obtient à la fin du traitement une image de la scène où seuls les points qui ne sont pas cachés par d'autres apparaissent.

Si cette méthode est efficace du point de vue des résultats fournis, elle est cependant relativement coûteuse en temps de calculs puisqu'il faut, pour chaque pixel de chaque triangle, calculer la distance à l'observateur et faire le test de comparaison avec le Z-buffer. L'utilisation du parallélisme paraît donc une bonne solution pour obtenir de meilleures performances.

4.2 Z-buffer parallèle développé au LIP

L'algorithme parallèle développé au sein du LIP est le fruit de plusieurs travaux successifs réalisés par différentes personnes telles que Lefèvre, Charles, Miguet mais aussi Pierson, Silber et Feschet. Nous présentons ici les caractéristiques les plus importantes de l'algorithme. Il faut noter que cette présentation est en partie inspirée de celle réalisée par Jean-Marc Pierson dans sa thèse [Pie96]. Pour des informations plus détaillées, le lecteur pourra aussi se reporter à [CLM95, Lef93] et [Sil94].

4.2.1 Structures de données

Les objets de la scène à visualiser sont décrits par des surfaces triangulées. C'est à dire que l'on dispose d'une liste de triangles définis chacun par trois sommets. étant donné qu'un sommet peut appartenir à plusieurs facettes, on évite un stockage redondant des informations en mémorisant les sommets dans un tableau (3 valeurs par sommet X,Y,Z) et en décrivant les triangles par trois indices correspondants aux positions de ses sommets dans le tableau de sommets. La figure 4.1 reprend cette structure.

Pour optimiser le processus parallèle, on utilise là aussi un équilibrage des charges. Le paragraphe suivant présente une méthode de partitionnement de données rectilinéaires en fonction d'une heuristique d'équilibrage des charges introduite en 1991 par Robert et Miguet [MR91].

4.2.2 Algorithme élastique

Cette méthode d'équilibrage est similaire à celle que l'on a vue au chapitre précédent si ce n'est que l'on divise l'image en bandes horizontales de tailles variables de manière à ce qu'elles

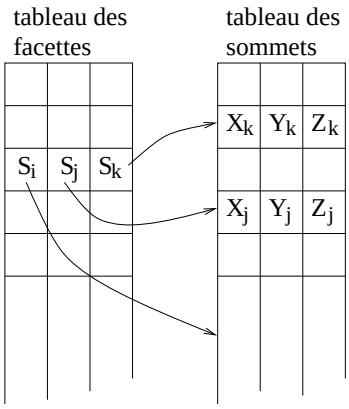

FIG. 4.1 - *Structure de données utilisée pour décrire la scène*

contiennent toutes approximativement la même quantité de travail. Cela peut être le cas lorsque le travail n'est pas réparti de manière homogène sur l'image.

L'algorithme élastique se place dans le cas du traitement de données rectilinéaires, c.-à-d. un ensemble de données ordonné linéairement. Le principe de cette heuristique s'appuie sur un module élémentaire représentant un ensemble minimal de données à traiter et une fonction d'évaluation de la charge relative à ces données. Les données sont donc initialement distribuées équitablement sur les processeurs et partitionnées en bandes de tailles égales. Chaque processeur calcule alors la charge des modules qu'il possède. Ensuite, ces charges sont diffusées sur tous les processeurs de manière à ce que chacun possède les charges de tous les modules de l'ensemble. Enfin, l'algorithme élastique consiste à calculer l'histogramme cumulé des charges du premier module vers le dernier, à diviser la charge totale par le nombre de processeurs puis à retrouver les indices des modules aux frontières de chaque division pour obtenir un partitionnement des données équilibré en terme de charge de travail. La figure 4.2 illustre ce processus sur l'histogramme cumulé.

Dans notre cas, le module est une ligne de l'image et la charge de travail associée est estimée par le nombre de triangles passant sur cette ligne. Ainsi, on se rend bien compte que les tailles des bandes finales d'image réalisant l'équilibrage des charges dépendent de la répartition des triangles dans l'image. Les paragraphes suivants indiquent les différentes stratégies de distribution des données pour réaliser cet équilibrage.

4.2.3 Première approche

Dans la première approche du Z-buffer parallèle, les facettes et les sommets sont distribués de manière équitable sur les processeurs, sans tenir compte de leurs relations dans la scène. Ainsi, les sommets composant une facette peuvent être sur un autre processeur que celle-ci. Ensuite, on calcule la projection de tous les sommets du référentiel de la scène vers le référentiel de l'image. Étant donnée la distribution équitable des points, on constate que cette opération de projection est complètement équilibrée au niveau des charges de travail. Enfin, tous les sommets projetés sont communiqués à tous les processeurs. En utilisant l'algorithme élastique, on calcule la partition

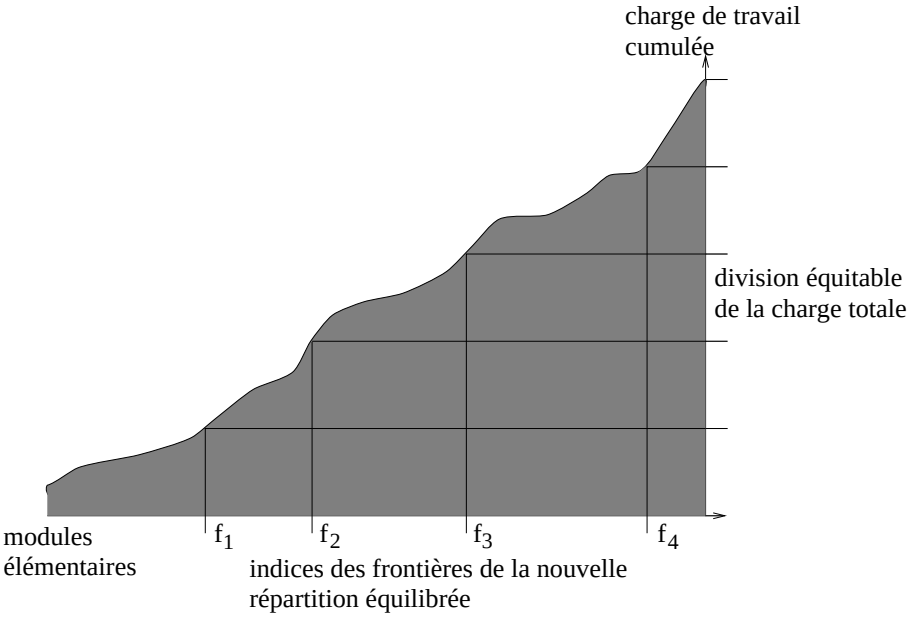

FIG. 4.2 - *Histogramme cumulé des charges de travail et calcul des frontières de la nouvelle partition équilibrée, dans le cas de 5 processeurs*

optimale de l'image en bandes horizontales en utilisant la charge de travail sur chaque ligne. Ensuite, les facettes sont redistribuées de manière à ce qu'un processeur possède seulement celles dont il a besoin pour traiter sa bande d'image. Les facettes qui sont à cheval sur plusieurs bandes sont dupliquées sur les processeurs correspondants.

Cette approche fonctionne relativement bien en terme de temps de calcul. Son principal inconvénient réside dans la duplication de tous les sommets sur les processeurs qui limite son extensibilité mémoire. C'est pourquoi un deuxième algorithme a été mis au point dans le but d'éviter ces duplications de données.

4.2.4 Seconde approche complètement distribuée

Le principe de cette nouvelle approche est de distribuer les points projetés sur les processeurs de manière à ce que chacun d'eux possède uniquement les points relatifs aux triangles se projetant dans sa bande d'image. Cette distribution des points ne pose pas de problème particulier puisqu'il est facile de savoir dans quelle bande d'image se projette chaque point. Par contre, l'accès à ces sommets devient plus complexe. En effet, dans la première approche, l'accès à un sommet est direct en fonction de son indice dans le tableau global, alors que dans cette nouvelle approche, il faut parcourir la liste des sommets locaux pour trouver le point que l'on recherche. Comme cette solution n'est pas satisfaisante du point de vue des performances, une table de hachage a été implantée pour permettre un temps d'accès moyen en temps constant ($\theta(1)$). Cette table a une taille de $4S/NbP$ où S est le nombre total de sommets et NbP le nombre de processeurs.

Cette nouvelle version permet donc d'obtenir des performances aussi bonnes que la précédente,

tout en économisant au maximum la mémoire utilisée. Néanmoins, une modification supplémentaire a du être apportée dans le cadre de la génération d'animations.

4.2.5 L'animation

Lorsque l'on calcule une animation où le mouvement de l'observateur et/ou des objets de la scène sont faibles, les changements de position des objets dans l'image de visualisation sont eux aussi relativement faibles. Ainsi, deux images consécutives de l'animation vont être très similaires et pour chaque bande associée à un processeur, c'est quasiment les mêmes ensembles de polygones et de sommets qui vont intervenir. On peut donc utiliser cette cohérence spatiale pour ne pas faire les calculs d'équilibrage des charges sur toutes les images mais seulement sur certaines images convenablement choisies.

Néanmoins, un autre problème apparaît dans ce nouveau contexte: dans les approches précédentes, lorsqu'une facette n'a aucun de ses points dans l'image, elle est effacée de la mémoire. Or, dans une animation, il est possible que des parties de la scène qui sont hors de l'image à un instant donné, entrent dans le champ de vision et deviennent donc visibles. Cela implique qu'il faut conserver une trace de tous les éléments de la scène et suivre leurs déplacements pour pouvoir gérer ces entrées et sorties du cadre de l'image. On obtient donc deux types de facettes: les *actives* qui sont dans l'image et les *passives* qui sont en dehors. Les facettes passives peuvent être totalement distribuées sur les processeurs puisqu'elles n'interviennent pas directement dans le calcul de l'image. Concernant les facettes actives, elles sont distribuées comme dans les approches précédentes avec d'éventuelles duplications pour les facettes à cheval sur plusieurs bandes d'image.

Le fond du problème vient du fait que les facettes peuvent changer d'état et donc changer d'ensemble. Or, les deux ensembles de facettes ne sont pas répartis de la même manière sur les processeurs et requièrent deux structures de données distinctes. De plus, il faut que ces structures permettent des insertions et suppressions rapides. Finalement, ce sont des listes stockées dans des tableaux qui ont paru être la structure la plus adaptée. Une facette est dorénavant composée de la liste des indices de ses sommets et de l'intervalle des processeurs sur lesquels elle se projette. Cela permet de savoir à quel(s) processeur(s) l'envoyer quand elle devient active. Enfin, lors de l'estimation de la charge, la contribution de chaque facette active frontière doit être prise en compte par un seul processeur, pour ne pas compter sa charge de travail plusieurs fois.

4.2.6 Résultats et conclusion

Le développement des algorithmes précédents a été réalisé à l'aide de PPCM. Les diverses fonctions d'équilibrage telles que l'algorithme élastique sont implantées dans PPCM et directement utilisables pour n'importe quelle structure de données rectilinéaire. Les résultats confirment la pertinence de la prise en compte de la cohérence entre images. En effet, pour une animation où le point de vue tourne de quelque degrés à chaque image, les temps de redistribution des données entre la première image et les suivantes sont réduits d'un facteur 2,5 environ.

En conclusion, nous avons décrit un algorithme parallèle de tampon en profondeur (Z-buffer). Cet algorithme, parti d'une première ébauche relativement simple, est finalement très complet et efficace. Nous avons vu qu'il est extensible aussi bien en temps de calcul (en distribuant les calculs de l'image sur les processeurs) qu'en mémoire (en distribuant les facettes et sommets de la scène).

Néanmoins, nous allons voir dans la partie suivante que cet algorithme n'est pas tout à fait adapté à la visualisation des terrains texturés, aussi bien par les structures de données que par les types de calculs utilisés. De plus, lorsque j'ai commencé ce travail sur la visualisation des terrains, le Z-buffer n'en était qu'à la première approche. Mon travail a donc consisté à adapter le Z-buffer existant aux données de type MNT tout en portant une attention particulière à la gestion de la mémoire et au volume des communications de données entre les processeurs.

4.3 Algorithme parallèle de visualisation de terrains texturés

Pour concevoir notre programme parallèle de visualisation, nous sommes donc parti du Z-buffer parallèle disponible mais aussi d'un programme séquentiel de visualisation de terrains que j'avais développé lors de mon stage de première année à l'ENS de Lyon. Ce dernier a été appelé *Volter* pour *surVOL de TERrain* et a fait l'objet d'un dépôt de logiciel [CVS96a]. Ce logiciel comprend une interface interactive et un module d'animation. Dans les deux cas, la taille importante des données à traiter et la quantité de calculs demandée ne sont pas compatibles avec les contraintes du temps réel. Une exploitation efficace de ce logiciel nécessite donc le développement d'une version parallèle.

Avant de présenter le schéma de notre algorithme parallèle, nous décrivons rapidement les structures de données spécifiques aux MNT ainsi que l'algorithme séquentiel de plaquage de texture. Une étude plus détaillée est également disponible dans [CV93].

4.3.1 Visualisation séquentielle de MNT

Comme nous l'avons évoqué précédemment, les terrains texturés sont composés de deux types de données. Le premier est le MNT dont on rappelle qu'il est constitué d'une grille orthogonale régulière de points dont on connaît les altitudes (voir la figure en page 26). Ainsi, la structure est très simple et compacte puisqu'il suffit de donner uniquement la suite des altitudes des points ligne par ligne. La largeur du MNT (largeur d'une ligne) nous permet de retrouver facilement les coordonnées X et Y des points dans la grille.

Le second est une image numérique, représentant la texture du terrain. Elle est habituellement calculée à partir des deux images satellites de manière à obtenir une image parfaitement perpendiculaire au terrain (ortho-image) que l'on peut donc superposer facilement sur celui-ci. La résolution de cette image est généralement bien plus élevée que celle de la grille du MNT, il n'est pas rare d'avoir un facteur 10 entre les deux.

Une méthode classique utilisée en visualisation consiste à décomposer le terrain en triangles, puis à les dessiner un par un. Dans notre cas, la triangulation du terrain est triviale grâce à sa structure

de grille régulière. Mais plutôt que d'utiliser la structure utilisée dans l'algorithme parallèle du Z-buffer (voir figure 4.1), il est préférable de garder la compacité de la description implicite du MNT. Ainsi, on va simplement stocker les altitudes des points dans un tableau à deux dimensions correspondant à la taille du MNT. Ensuite, on parcourt la grille du terrain ligne par ligne, en prenant deux lignes consécutives à la fois et on dessine les deux triangles contenus dans chaque case de la grille (4 points voisins).

Enfin, le calcul des couleurs de chaque pixel dans les triangles requiert la prise en compte de la texture. Lorsque l'on superpose l'image sur le MNT, on constate qu'à chaque point de la grille correspond une position particulière dans l'image de texture. Ceci signifie que pour chaque sommet de la scène, on doit stocker non seulement ses coordonnées dans l'espace tridimensionnel mais aussi ses coordonnées dans l'image de la texture (que l'on appellera *coordonnées textures*). Malgré tout, comme la texture est superposée au terrain, les coordonnées de texture d'un point sont eux aussi implicites puisque directement liés aux coordonnées réelles. Ainsi, lorsque l'on dessine un triangle dans l'image en traçant la suite des segments horizontaux qu'il contient (technique classique appelée *rasterization*), il faut interpoler les coordonnées textures des sommets d'un segment à l'autre et aussi le long de chaque segment. Cela permet de déduire la position dans la texture de chaque pixel du triangle et donc sa couleur. La figure 4.3 montre un exemple synthétique de visualisation où la texture au milieu est plaquée sur un terrain sinusoïdal à gauche pour donner l'image de droite.

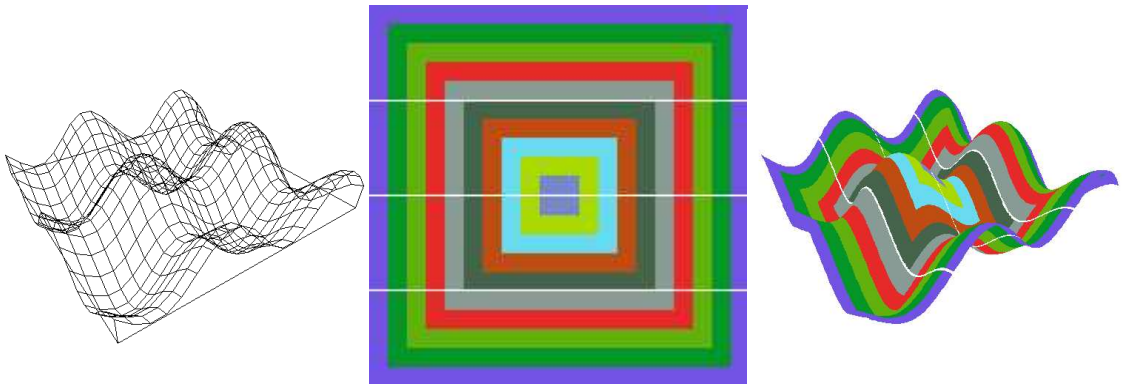

FIG. 4.3 - *Exemple de visualisation (droite) à partir d'un MNT (gauche) et d'une image de texture (centre)*

Si l'on compare cet algorithme avec celui décrit dans la section 4.2, on remarque que la scène est aussi décrite par décomposition en triangles mais d'une manière bien plus compacte. De plus, dans le cas présent, nous avons un type de données supplémentaire qui intervient, l'image de la texture. Aussi, les calculs eux-mêmes sont intrinsèquement différents puisque nous ne faisons plus appel à un modèle d'illumination pour calculer la couleur des pixels mais à l'image de la texture. Il est donc clair que l'algorithme parallèle préexistant ne peut être utilisé directement pour visualiser les terrains. Puisque nous devons concevoir un nouvel algorithme parallèle, il est important d'aborder dans un premier temps les différentes stratégies utilisées dans la littérature pour paralléliser les algorithmes classiques de visualisation. Cela va nous permettre de tirer le meilleur parti de l'expérience communautaire acquise jusqu'à maintenant dans ce domaine.

4.3.2 Les travaux sur la visualisation en parallèle

Lorsque l'on travaille dans le domaine de la synthèse d'image, deux espaces doivent généralement être traités. Le premier est l'espace objet, c.-à-d. la scène à visualiser. Le second est l'espace image qui correspond à une région définie du plan sur lequel on va projeter la scène. Ces deux espaces mènent à trois principaux schémas de parallélisation, selon l'espace que l'on décompose pour réaliser les calculs (objet, image ou les deux).

Division de l'espace objet

Il y a plusieurs manières de diviser l'espace objet. L'idée principale consiste à diviser la scène en autant de sous-ensembles disjoints qu'il y a de processeurs. Ensuite, chaque processeur calcule la contribution de son sous-ensemble local sur l'image finale. De même, le fait que le traitement des polygones soit décomposable en étapes indépendantes permet d'utiliser un chaînage (pipeline) logiciel ou matériel. Un second niveau de parallélisme peut être obtenu en utilisant plusieurs chaînes indépendantes comme dans [CDH+88, Ell94].

Néanmoins, deux problèmes apparaissent avec ces techniques. Le premier est que la bande passante de la chaîne de traitement est limitée par l'étape la plus lente du processus, ce qui peut représenter une perte significative d'efficacité. Le second est encore plus important car il concerne les goulots d'étranglement dus aux accès concurrents aux pixels de l'image. En effet, les objets de la scène peuvent se projeter sur les mêmes régions de l'image. Il faut donc une étape supplémentaire de composition des images partielles obtenues sur chaque processeur pour récupérer l'image finale. Des techniques basées sur des arbres de décomposition d'image ont été proposées pour résoudre ce problème, mais cette approche reste limitée à un petit nombre de processeurs.

Division de l'espace image

Comme pour l'espace objet, il existe plusieurs manière de diviser l'espace image. On peut procéder en travaillant directement au niveau des pixels ou des groupes de pixels. Ces méthodes sont les plus courantes et ont fait l'objet de nombreuses études parmi lesquelles on peut citer notamment [KG79, Whi94, MCEF94]. Cependant, les techniques les plus appropriées à un développement logiciel (en opposition au développement matériel) sont celles qui portent sur des groupes de pixels voisins. Les décompositions les plus répandues sont celles en bandes horizontales ou verticales de l'image ou encore en rectangles.

Les problèmes généralement rencontrés avec ces approches portent sur la distribution de la scène. Si celle-ci est dupliquée sur tous les processeurs, il apparaît alors des calculs redondants menant à un algorithme non extensible en temps. Une duplication partielle de la scène est aussi possible mais implique des surcoûts en communication relativement importants. Certains algorithmes ont été conçus de façon à minimiser les communications en distribuant statiquement les données [CJ81]. Mais ceux-ci se trouvent confrontés au problème de déséquilibre des charges de travail sur les processeurs. Whelan [Whe85] et Roble [Rob88] ont résolu ce problème en utilisant

une décomposition statique prenant en compte l'équilibrage des charges.

Le fait d'avoir à traiter un ensemble de pixels voisins sur un même processeur permet de tirer parti de la cohérence spatiale au moment du rendu des objets. D'un autre côté, cela peut entraîner des problèmes d'équilibrage puisque la répartition des objets sur l'image n'est pas forcément homogène. Par conséquent, certaines régions peuvent être beaucoup plus chargées que d'autres. Une allocation par blocs cycliques des lignes de l'image peut alors être utilisée comme compromis entre l'équilibrage et l'exploitation de la cohérence spatiale. Néanmoins, on retombe alors sur des problèmes de consommation mémoire.

Enfin, il existe une approche basée sur le dessin en parallèle de chaque polygone [FFR83, All91], mais il y a encore plus de contraintes et le parallélisme est relativement limité. Une fois encore, les performances sont restreintes par le recours obligatoire à des synchronisations ou par les conflits d'accès aux données.

Division hybride

Une troisième approche consiste à diviser à la fois l'espace de l'image et l'espace de la scène. Cette approche, qui semble très prometteuse, n'a pourtant pas suscité un engouement important de la part des chercheurs. Mais cela est sans doute dû au fait qu'elle est relativement récente. C'est cette méthode qui est utilisée dans la seconde version de l'algorithme du Z-buffer parallèle décrit précédemment. On a pu constater qu'elle a l'avantage de rendre l'algorithme extensible aussi bien en temps de calcul qu'en mémoire.

Conclusion

Cette étude bibliographique à permis de mettre en évidence les critères importants pour obtenir un algorithme parallèle de visualisation qui soit efficace. Ces critères sont :

- La granularité du parallélisme (taille des tâches),
- l'équilibrage des charges de travail,
- la distribution des données et leur accès (gestion mémoire, ...),
- l'utilisation de la cohérence spatiale dans l'image,
- l'extensibilité.

Nous pouvons observer qu'à l'heure actuelle, aucun algorithme présenté dans la littérature ne satisfait à tous ces éléments à la fois. Sans doute parce que cela n'est pas réellement possible. En effet, la granularité et l'extensibilité d'un tel algorithme sont deux notions relativement opposées. La granularité peut être vue comme l'intervalle des tailles des sous-ensembles de données/travail par processeur, sur lequel l'algorithme reste efficace. L'extensibilité est relative au bon comportement du programme lorsque le nombre de processeurs varie. Or, cette variation modifie continûment

la granularité du problème alors qu'en règle générale, les algorithmes ne sont écrits que pour un type de granularité prédéfini et fixe. On pourra donc toujours trouver un nombre de processeurs pour lequel l'algorithme perd son efficacité. Néanmoins, les méthodes hybrides paraissent être les plus prometteuses car elles ont l'avantage d'agir aussi bien sur les temps de calcul que sur la consommation mémoire.

Le but de notre étude est donc de concilier le maximum de caractéristiques énoncées ci-dessus dans le cas particulier des terrains texturés.

4.3.3 Algorithme parallèle

Dans la suite, nous reprenons le même modèle de machine parallèle que celui décrit dans la section 3.3. Tenant compte des conclusions de notre étude bibliographique, nous avons donc choisi d'utiliser la méthode hybride de parallélisation. La principale originalité de notre algorithme réside dans les structures de données utilisées pour les messages lors des communications entre processeurs. Mais voyons tout d'abord comment les données sont réparties initialement sur les processeurs et comment on réalise l'équilibrage des charges.

Équilibrage des charges

Le MNT et l'image de la texture sont initialement distribués équitablement sur les processeurs en utilisant une décomposition en bandes horizontales. Comme dans le paragraphe 4.2, nous estimons la charge de travail associée à chaque ligne de l'image en comptant le nombre de polygones qui la traversent. Cette étape comprend donc la phase de projection des points du terrain sur l'image. Comme chaque processeur ne fait ces calculs que pour sa bande locale de terrain, on constate que cette opération est complètement équilibrée. Toutes les estimations locales de charge sont ensuite globalisées en utilisant une opération de réduction. Enfin, on utilise l'algorithme élastique pour calculer une nouvelle partition de l'image en bandes horizontales qui équilibre les charges de travail. La figure 4.4 montre une telle partition de l'image dans le cas de 8 processeurs pour un terrain des Alpes[1]. Sur la droite, on peut voir les charges de travail sur chaque ligne et l'on constate que les bandes étroites ont des lignes plus chargées que les autres bandes.

Dans le paragraphe suivant, nous décrivons la structure des messages utilisée pour les échanges de données entre processeurs. Nous supposons dans un premier temps que la texture est dupliquée sur tous les processeurs. Les informations à redistribuer ne concernent donc que le terrain.

Redistribution du MNT

Une fois que l'image est partitionnée, le MNT doit être redistribué sur les processeurs relativement à celle-ci. De la même manière que pour l'algorithme parallèle de reconstruction de terrain

1. Aimablement fourni par le Département des Sciences de la Terre et de la Vie de l'ENS Lyon

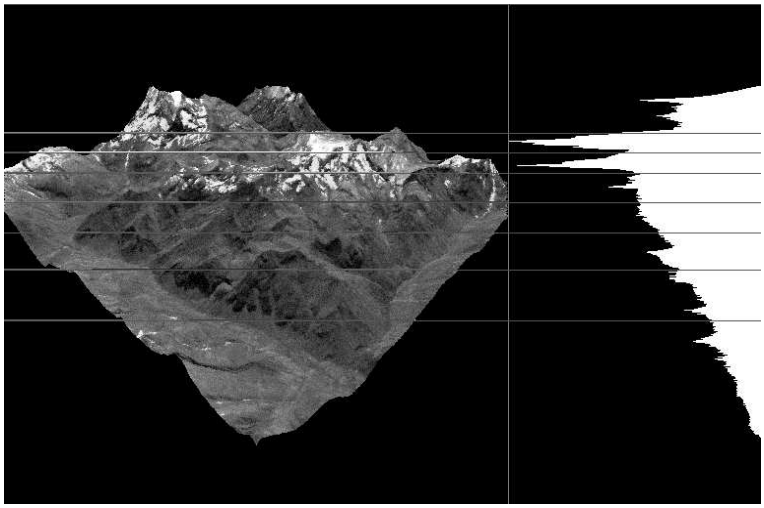

FIG. 4.4 - *Partition de l'image d'un terrain des Alpes dans les cas de 8 processeurs*

vu au chapitre précédent, nous voulons que chaque processeur ne reçoive que les données dont il a besoin pour dessiner sa bande d'image.

Comme dans la version séquentielle et pour les mêmes raisons, nous avons choisi de ne pas adopter la représentation explicite des polygones mais de garder la structure compacte et implicite de la grille du terrain. Ainsi, nous n'allons travailler que sur des suites d'altitudes. Cela va nous permettre de gagner une place mémoire considérable mais va nous obliger à conserver l'ordre précis des points pour que les processeurs puissent reconstituer correctement les parties de terrain qui leurs seront envoyées.

Ainsi, pour redistribuer les points du terrain, chaque processeur construit NbP listes correspondantes aux NbP destinations possibles. Un point du MNT va dans la liste l si sa projection se trouve dans la bande l de l'image. Une fois les listes constituées, on obtient une décomposition de la grille du MNT par des ensembles disjoints de points. Cela implique que les triangles à cheval sur plusieurs bandes d'images vont avoir leurs sommets dans différentes listes empêchant ainsi leur traitement. Il faut donc rajouter à chaque liste, tous les points de la grille qui complètent un triangle dont un des sommets au moins appartient à cette liste. Ces points à ajouter, aussi appelés points complémentaires, dépendent donc directement de la triangulation choisie. Celle-ci est indiquée dans la partie gauche de la figure 4.5 ainsi que les points complémentaires pour un sommet du MNT. Dans la partie droite de cette même figure est représenté un exemple plus général.

Sur cette figure, on voit aussi que l'ensemble de départ (en noir) et non connexe dans la grille. Cela vient du fait que selon le relief du terrain et la position de l'observateur, des régions distantes dans le terrain peuvent se retrouver dans la même bande d'image et donc sur le même processeur. Ce phénomène peut d'ailleurs être observé sur la figure 4.4.

Malgré tout, deux points voisins dans le terrain ont une grande probabilité d'être projetés dans la même bande d'image. Cette remarque peut être utilisée pour coder efficacement les messages lors des redistributions des données. Une structure compacte doit être mise au point de manière

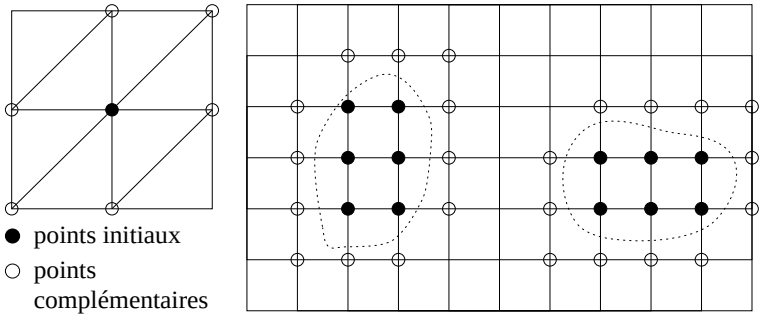

FIG. 4.5 - *Triangulation et points complémentaires à un sommet du MNT (à gauche), points complémentaires pour un ensemble quelconque de points (à droite)*

à envoyer toutes les données d'un processeur à l'autre en un seul message. Pour cela, on utilise la régularité de la grille. L'ensemble des points destinés à un processeur précis est décrit par la suite des segments horizontaux qui le composent. Par segment horizontal, nous entendons une suite de points connexes se trouvant sur une même ligne de la grille du MNT. Les segments sont donnés ligne par ligne et dans l'ordre croissant des abscisses. De cette manière, l'exemple général donné dans la figure 4.5 serait décrit par 9 segments, en commençant par le plus en haut à gauche et en finissant par le plus en bas à droite.

Pour transférer des morceaux de terrains sur leur processeur destination, le message doit donc contenir les informations suivantes :

- le nombre total de segments dans l'ensemble,
- et pour chaque segment :
  - le nombre de points dans le segment,
  - la position dans la grille (X, Y) du premier point du segment,
  - les coordonnées de projection (U, V et H) de chaque point du segment.

la structure des messages correspond donc à celle donnée dans la figure 4.6.

Nombre de segments	Nombre de points dans le segment 1	X^1	Y^1	U_1^1	V_1^1	H_1^1	U_2^1	V_2^1	.....	Nombre de points dans le segment i	X^i	Y^i	U_1^i	V_1^i	H_1^i	.....

FIG. 4.6 - *Composition des messages pour l'envoie des données du terrain*

Il est important de voir ici le compromis que nous avons fait entre la taille des messages et le temps de calcul. En effet, nous aurions pu ne pas envoyer les coordonnées de projection des points. Cela aurait donné des messages bien plus petits puisque nous aurions pu seulement donner l'altitude des points, donnant des messages presque 3 fois plus petits. D'un autre côté, le processeur recevant le message aurait du recalculer les projections de chaque point (25 opérations flottantes) pour pouvoir dessiner les triangles. Il serait donc dommage de ne pas profiter des calculs qui ont

déjà été effectués lors de l'équilibrage des charges. Il y a donc un compromis à faire entre temps de communication et temps de calcul. Après avoir testé les deux méthodes, il s'est avéré que le passage des coordonnées de projection était plus avantageux.

Finalement, une fois que les données du terrain sont redistribuées sur les processeurs en utilisant une multi-distribution, la visualisation peut être réalisée par chaque processeur sur sa bande d'image. Comme la texture est complètement disponible sur chaque processeur, le calcul des couleurs des pixels se fait exactement comme dans la version séquentielle. Par contre, pour dessiner les triangles reçus, il faut d'abord les reconstituer. Par cela, nous voulons dire qu'il faut retrouver dans le message, les segments qui forment des cases de la grille, c.-à-d. qui sont sur deux lignes consécutives et qui se chevauchent sur l'axe horizontal des abscisses, comme indiqué dans la figure 4.7. Les informations fournies dans le message et l'ordre des segments permettent une recherche rapide et efficace.

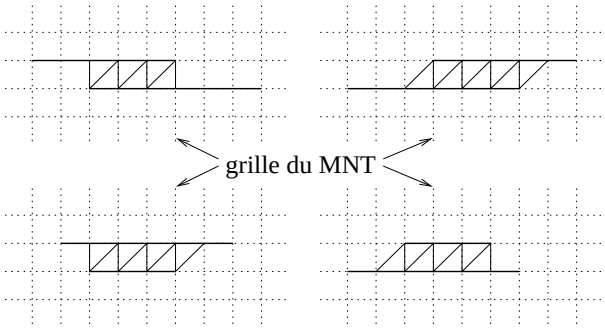

FIG. 4.7 - *Couples de segments générant des triangles à visualiser*

Néanmoins, il y a deux cas un peu particuliers. Lorsque deux segments sur deux lignes consécutives n'ont qu'un seul point en commun sur l'axe des abscisses, il se peut qu'aucun triangle ne soit généré. Dans le cas où le premier point du premier segment a la même abscisse que le dernier point du second segment (que l'on appellera chevauchement début-fin), alors deux triangles sont générés. Par contre, dans le cas inverse (chevauchement fin-début), il n'y a aucun triangle créé à cause de la triangulation choisie. Ces deux cas sont exhibés dans la figure 4.8 et l'on constate d'ailleurs que ce problème est symétrique si l'on change de triangulation.

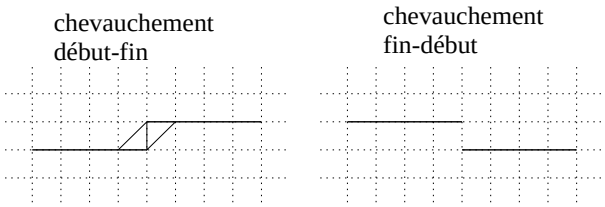

FIG. 4.8 - *Cas particuliers de chevauchements aux extrémités des segments, générant ou pas des triangles*

Jusque là, nous avions considéré que la texture était dupliquée. Mais il n'est pas difficile de s'apercevoir que cette stratégie n'est pas optimale en coût mémoire. Nous allons donc aussi la répartir sur les processeurs.

Distribution de la texture

Pour réaliser le plaquage de texture, il faut que chaque processeur possède au moins la partie de texture correspondante à la partie de terrain qu'il doit traiter. Étant donnée la grande similitude entre les structures d'image et de terrain (toutes les deux des grilles régulières 2D), les informations collectées lors de la construction des messages du terrain sont utilisées pour concevoir les messages relatifs à la texture. Ainsi, ces messages ont exactement la même structure que précédemment, mais à la place des coordonnées U, V et H des projections des points dans l'image, nous plaçons la couleur de chaque pixel. En voyant une telle ressemblance entre les messages de texture et de terrain, on peut se demander pourquoi on n'a pas fusionné les deux types de message. En fait, cela n'est pas possible dans le cas général puisque, comme évoqué précédemment, la texture peut avoir une résolution plus importante que le terrain. Il en découle donc un nombre de segments plus élevé dans le message de texture que dans celui du terrain.

Après la réception des pixels de la texture, ceux-ci sont stockés les uns à la suite des autres dans un tableau unidimensionnel. Or, ces pixels de texture sont référencés par leurs coordonnées absolues dans l'algorithme de plaquage. Il faut donc pouvoir retrouver un tel pixel dans le tableau local en fonction de ses coordonnées réelles dans l'image de texture. On utilise donc une table de référence qui contient les informations de chaque segment reçu : position du premier point dans la texture, nombre de points dans le segment, mais aussi la position du segment dans le tableau local. Ainsi, pour retrouver un pixel, il suffit de parcourir la table de référence des segments puis de comparer la position de chacun d'eux avec celle du pixel demandé pour trouver le segment auquel il appartient. On en déduit facilement la position de ce pixel dans le tableau local.

L'utilisation d'une telle table crée un léger surcoût dans le calcul de l'image finale. Néanmoins, le temps de recherche du segment peut être grandement accéléré. Il suffit de trier les segments dans la table de référence en fonction de leur position, et de commencer la recherche au dernier segment utilisé. En effet, les pixels de texture qui entrent en jeu lors du dessin d'un triangle sont très proches dans la texture. Il y a une grande probabilité que le prochain pixel demandé soit dans le même segment que le pixel précédent ou dans un segment proche. On utilise donc cette cohérence spatiale pour accélérer notre recherche.

4.3.4 Résultats

Notre algorithme parallèle de visualisation de terrain texturés a été porté sur machine en utilisant PPCM. Dans cette étude, nous avons utilisé la portabilité de cette bibliothèque pour tester notre algorithme sur deux machines parallèles. La première est une *Volvox* conçue par la société Archipel. Chaque noeud de la machine comprend deux éléments, un processeur i860 pour les calculs, et un *transputer* T800 pour les communications. La seconde machine est une Cray T3D[2]. Elle est similaire à la T3E décrite au paragraphe3.4.2 mais n'a que 128 processeurs et est plus ancienne et donc moins rapide. Par contre, elle présente des performances plus intéressantes que la Volvox. Enfin, les données utilisées pour nos tests sont un MNT de 700×680 points et une texture en

2. Appartenant au CEA de Grenoble

niveaux de gris (8 bits) de la même taille.

Résultats sur machine Volvox

Les graphiques de la figure 4.9 montrent les temps de calcul d'une image avec ou sans équilibrage en fonction du nombre de processeurs pour deux tailles d'image.

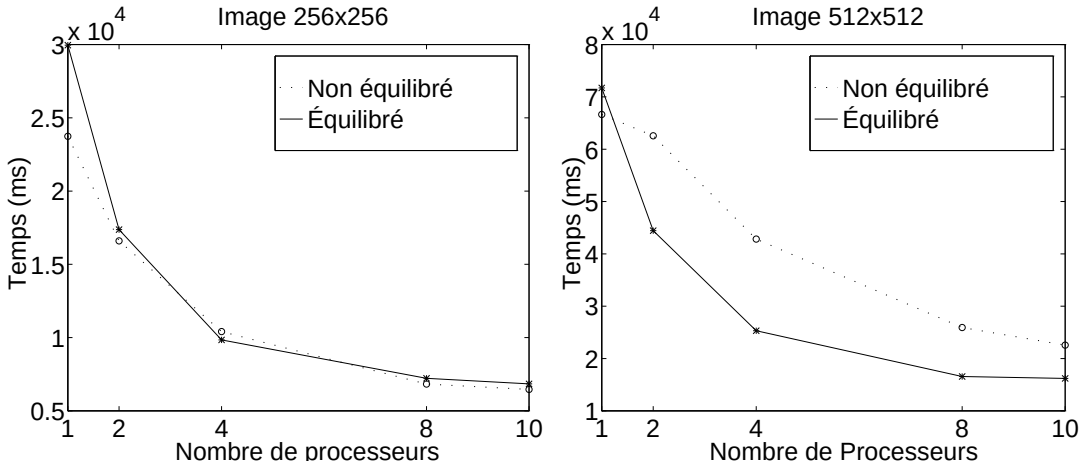

FIG. 4.9 - *Temps de calcul d'une image avec ou sans équilibrage en fonction du nombre de processeurs sur machine Volvox*

La version non équilibrée consiste tout simplement à diviser l'image en NbP bandes de mêmes tailles. Pour mettre en évidence le temps d'estimation de la charge dans la version équilibrée, nous avons inclus les temps des deux versions sur un seul processeur.

Pour la petite taille d'image, on peut observer que la version non équilibrée présente à peu près les mêmes performances que l'autre version et est même parfois meilleure. Néanmoins, pour l'image plus grande, la version équilibrée est toujours meilleure. Cette différence entre les deux graphes met bien en évidence l'influence importante de la taille de l'image sur les performances de l'algorithme. En fait, on est confronté exactement au même problème que pour l'algorithme de reconstruction des terrains. On passe d'un modèle parallèle à grain moyen ou gros à un modèle à grain fin. En effet, la quantité de calculs de visualisation dépend directement de la taille de l'image à calculer et lorsque celle-ci est petite, la quantité de travail effectif sur chaque processeur devient trop faible par rapport aux surcoûts du parallélisme. C'est d'ailleurs pour cela que la version non équilibrée présente des résultats au moins aussi bon que la version équilibrée puisqu'elle a moins de calculs *parasites* à faire. Donc, pour ces petites images, l'équilibrage n'est pas souhaitable. Par contre, dès que l'on a une taille d'image assez grande, l'équilibrage prend tout son sens et l'on peut vérifier son efficacité sur le graphe de l'image 512×512. Cette efficacité est encore plus visible sur les histogrammes de la figure 4.10. Chaque histogramme représente un calcul d'une image 512×512 sur 8 processeurs, et chaque barre indique le temps d'exécution total sur un processeur décomposé selon les temps des différentes parties de l'algorithme.

Les temps de visualisation sont nettement plus équilibrés dans l'histogramme de droite. Néan-

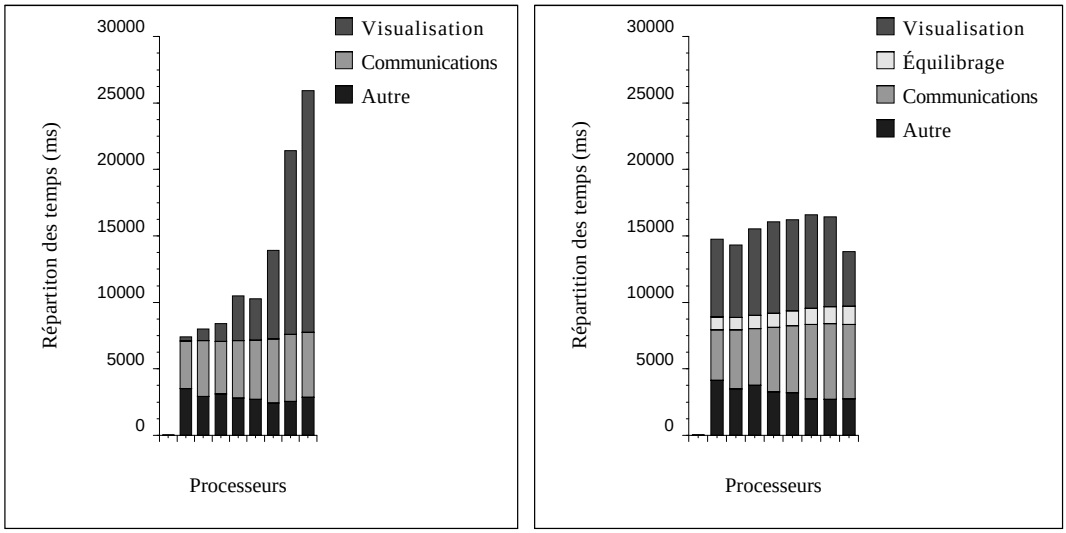

FIG. 4.10 - *Répartition des temps de calcul d'une image* 512×512 *sur 8 processeurs de la Volvox : version non équilibrée (à gauche), version équilibrée (à droite)*

moins, ils ne sont pas parfaitement égaux. Cela vient du fait que la précision de l'équilibrage repose sur la partition de l'image en bandes horizontales. Or, lorsque l'image est petite, les polygones sont plus condensés sur un nombre réduit de lignes, il est donc plus difficile de diviser l'image en obtenant la même quantité de travail dans chaque bande. L'équilibrage est donc moins précis. Quand l'image est plus grande, on a l'effet inverse, les polygones sont plus étalés dans l'image et la répartition du travail devient donc plus précise.

Une dernière illustration du lien entre la résolution de l'image calculée et les performances est donnée dans la figure 4.11. Dans ce graphique, l'efficacité de la parallélisation est exprimée en fonction de la taille de l'image de sortie. On rappelle que l'efficacité est définie par :

$$E_N = \frac{accélération\ obtenue\ avec\ N\ processeurs}{N} \tag{4.1}$$

L'évolution positive des courbes confirme l'influence de la taille de l'image de sortie.

Enfin, on constate aussi sur la figure 4.10 que les temps d'équilibrage sont identiques sur les processeurs et relativement faibles. Par contre, les temps de communication restent assez différents mais nous n'avons pas de moyen direct de les superviser puisqu'ils dépendent non seulement de la redistribution des données mais aussi de la configuration interne de la machine.

Maintenant que nous avons vérifié l'efficacité de l'équilibrage des charges, voyons ce qu'il en est de l'extensibilité. Dans la figure 4.9, les temps d'exécution décroissent quand le nombre de processeurs augmente, ce qui est plutôt encourageant vis à vis de l'extensibilité. Mais pour avoir une idée plus précise de celle-ci, nous avons fait des mesures pour obtenir une courbe d'accélération en fonction du nombre de processeurs. Le résultat pour les deux versions de l'algorithme est donné dans la figure 4.12.

La courbe de la version non équilibrée confirme que cette version n'est pas extensible. Par contre,

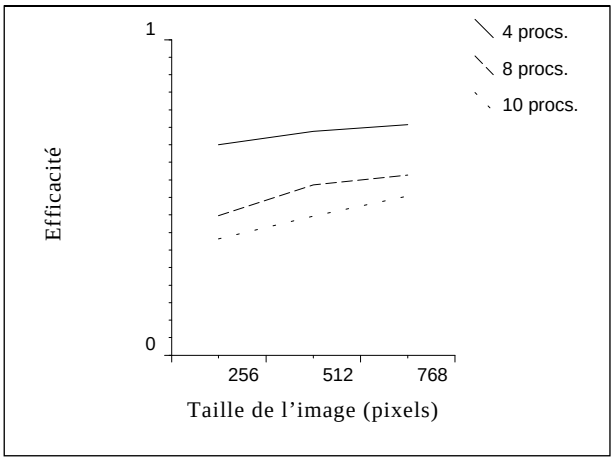

FIG. 4.11 - *Efficacité de l'algorithme parallèle en fonction de la taille de l'image de sortie sur machine Volvox*

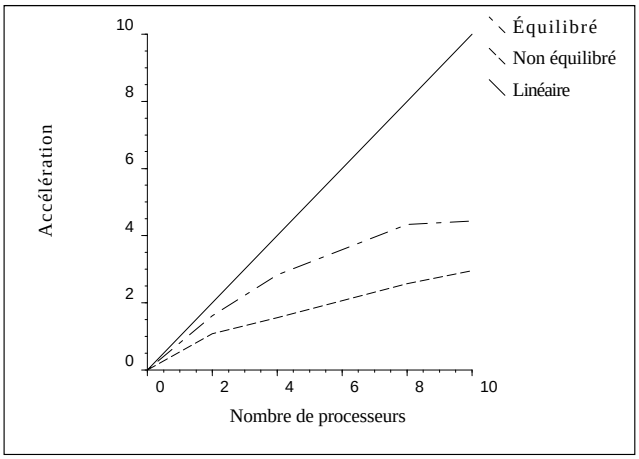

FIG. 4.12 - *Accélération en fonction du nombre de processeurs pour une image* 512×512 *sur machine Volvox*

la seconde courbe est meilleure mais tend aussi à s'infléchir assez rapidement. Comme on vient de le voir, cela est du à la taille de l'image calculée qui reste relativement petite, mais la taille des données à traiter intervient aussi. En effet, le terrain à traiter comporte un nombre moyennement important de triangles (un peu moins de 500000). Le nombre de triangle dans l'image est essentiel puisque de lui dépend directement la quantité de travail à effectuer.

Ce facteur est étroitement lié à la distance de l'observateur au terrain. Considérons des tailles de terrain et d'image de sortie fixées. Plus l'observateur est près du terrain et plus la partie de terrain qu'il voit est petite. Il y a donc moins de données à traiter. Par contre, le temps de visualisation reste important car triangles qui apparaissent dans l'écran couvrent des aires plus importantes puisqu'ils sont vus de plus près. Les calculs d'équilibrage et le volume des communications sont donc réduits. La figure 4.13 représente l'accélération en fonction de la distance entre l'observateur et le terrain pour plusieurs configurations de processeurs.

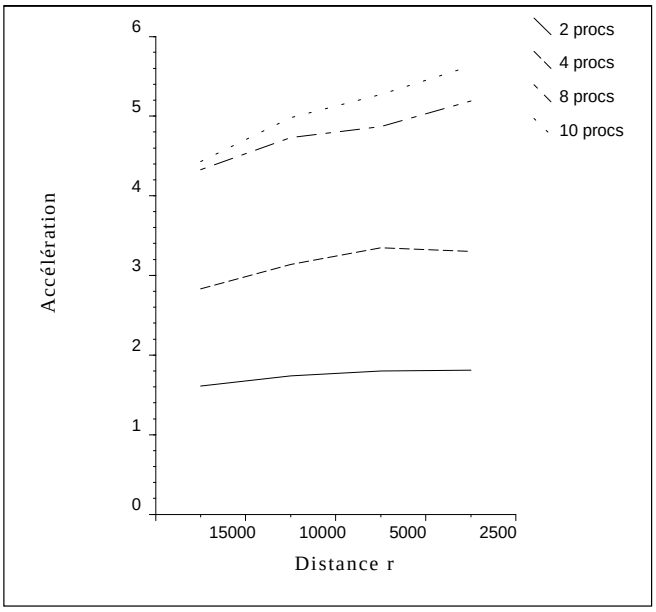

FIG. 4.13 - *Accélération en fonction de la distance entre l'observateur et le terrain sur machine Volvox*

Ces résultats sur machine Volvox nous ont permis de mettre en évidence certains paramètres qui jouent un rôle sur l'efficacité de l'algorithme parallèle. Cependant, d'autres expérimentations sur une machine plus performante avec plus de processeurs vont nous permettre d'obtenir des informations complémentaires sur le comportement de notre algorithme.

Résultats sur machine Cray T3D

La première motivation de ces tests est de vérifier le comportement observé sur la Volvox. Sur la figure 4.14, l'histogramme des temps de chaque processeur pour le calcul d'une image 512×512 est présenté pour une configuration à 16 éléments. L'efficacité de l'équilibrage des charges y est clairement confirmé. On peut remarquer que la répartition des charges sur processeurs dans la version non équilibrée n'est pas la même entre les deux figures 4.10 et 4.14. Cela vient d'un changement d'angle de vue entre les deux expériences. Bien sûr, ce changement n'a pas d'effet sur la répartition finale des charges dans la version équilibrée.

Une autre différence importante entre les deux algorithmes que l'on peut mieux appréhender ici, concerne les temps de communication. Ces temps sont moins bien répartis et plus importants dans la version non équilibrée. Dans cette version, les bandes d'image associées à certains processeurs peuvent contenir beaucoup plus de triangles que les autres. Ces processeurs vont donc recevoir beaucoup de données et cela va créer des goulots d'étranglement ralentissant les communications.

Enfin, un résultat supplémentaire que nous avons obtenu est le taux de calcul des polygones texturés. La figure 4.15 indique le nombre de polygones dessinés par notre algorithme parallèle selon le nombre de processeurs utilisés. Même si les performances ne sont pas linéairement propor-

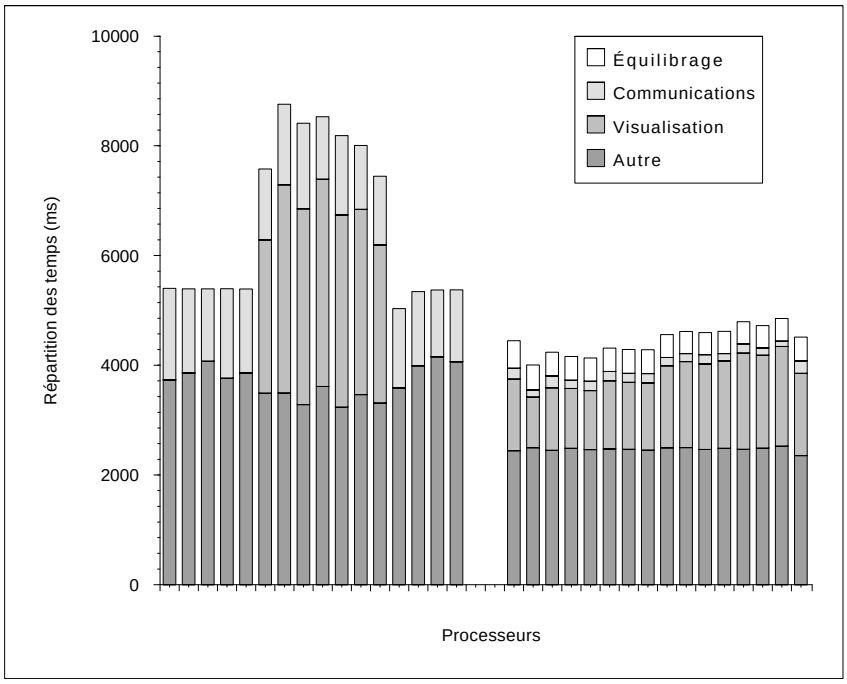

FIG. 4.14 - *Répartition des temps de calcul d'une image* 512×512 *sur 16 processeurs de la Cray T3D : version non équilibrée (à gauche), version équilibrée (à droite)*

tionnelles au nombre de processeurs, elles augmentent toujours avec le degrés de parallélisme. De plus, les performances absolues sont plutôt satisfaisantes pour des machines non dédiées puisque l'on atteint des taux d'environ 280000 polygones par seconde avec 32 processeurs.

Ceci termine l'analyse de nos résultats expérimentaux. Ceux-ci sont assez prometteurs et notre algorithme peut être utilisé comme une bonne base de départ pour d'éventuelles améliorations.

4.3.5 Améliorations possibles

Nous donnons ici quelques idées intéressantes qui pourraient compléter l'algorithme existant.

Multirésolution : le principe consiste à ne prendre en compte que les points du terrain qui sont significatifs dans l'image selon leur éloignement de l'observateur. Ainsi, tous les points du terrain près de l'observateur sont utilisés, mais pour les zones plus éloignées, seulement certains points entrent en jeu. La qualité de la visualisation n'est pas dégradée outre mesure par cette technique. Et la quantité de données à traiter étant réduite, les temps de calculs le sont aussi. Enfin, la même approche peut être utilisée pour la texture en la pré-filtrant à différentes résolutions.

Cohérence : cette idée a déjà été abordée lors de la description du Z-buffer développé au LIP donnée au paragraphe4.2. Elle consiste à utiliser la cohérence entre les images successives

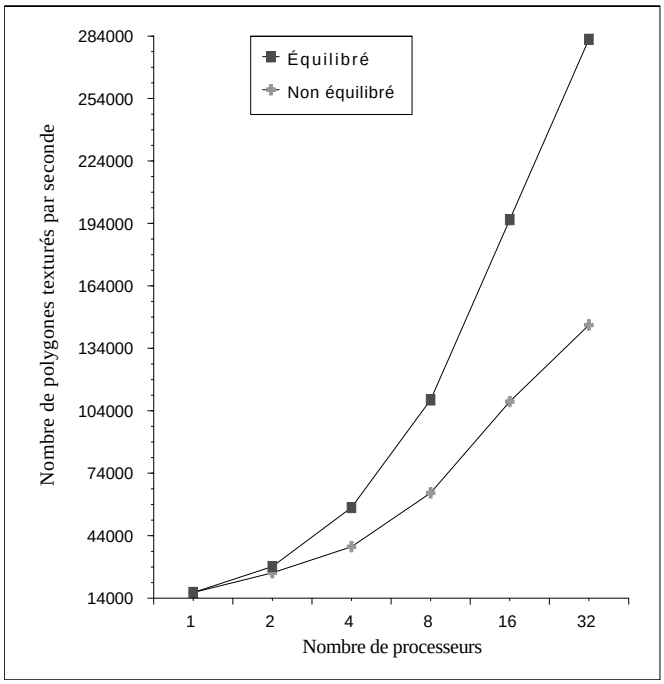

FIG. 4.15 - *Nombre de polygones texturés calculés par seconde en fonction du nombre de processeurs, sur Cray T3D*

d'une animation pour éviter de refaire certains calculs tels que l'équilibrage des charges et aussi réduire les communications.

Division rectangulaire : une des limitations principales de notre algorithme vient de la division de l'image en bandes horizontales. Elle contraint fortement le nombre de processeurs utilisables en même temps. Or, une division rectangulaire est plus flexible et repousse donc cette limite. De plus, la répartition du travail est plus précise avec ce type de partition. On pourrait adapter notre algorithme en estimant la charge non seulement sur les lignes mais aussi sur les colonnes de l'image.

4.4 Conclusion

Dans ce chapitre, nous avons étudié l'apport du parallélisme dans le domaine de la synthèse d'image et plus particulièrement sur la visualisation de terrains texturés. Pour concevoir notre algorithme, nous nous sommes inspiré d'un algorithme séquentiel de visualisation de terrains et d'un algorithme parallèle de Z-buffer qui a été développé par l'équipe d'imagerie du LIP. Nous obtenons un algorithme parallèle équilibré où les données de terrain et de texture sont distribuées sur les processeurs. Le développement réalisé avec la librairie PPCM nous a permis d'utiliser deux machines parallèles différentes pour faire nos expérimentations.

Cette étude nous a permis de mettre en évidence les principaux paramètres qui influent sur les

performances de ce type d'algorithmes parallèles. Les plus importants sont la taille de l'image de sortie et le nombre de polygones effectivement dans l'image qui dépend de la taille du terrain mais aussi de la position de l'observateur par rapport au terrain.

Le domaine de la visualisation est très gourmand en temps de calculs et en mémoire. Le temps d'exécution est d'autant plus essentiel lorsque l'on souhaite avoir une application interactive ou calculer des animations. Enfin, nous avons montré comment prendre en compte la structure spécifique du MNT et de la texture pour économiser la mémoire et réduire les communications. La méthode de parallélisation proposée ici permet non seulement des calculs plus rapides mais aussi le traitement d'ensembles de données de tailles plus importantes. Et cela est primordial dans le cas des MNT avec texture puisque l'on a vu dans le chapitre précédent que les terrains reconstruits peuvent atteindre une taille de 6000×6000 points et autant pour la texture.

Finalement, nous avons énoncé plusieurs voies d'amélioration de notre algorithme parallèle qui présente dors et déjà des performances absolues très intéressantes puisqu'il atteint presque 300000 polygones texturés par seconde.

CHAPITRE 5

Algorithme parallèle de transformations géométriques d'images

Les derniers travaux que nous présentons ici portent sur le problème des transformations géométriques d'image. Nous nous intéressons plus particulièrement aux transformations qui peuvent être décrites de manière paramétrée en fonction des coordonnées initiales de chaque pixel de l'image. Chaque pixel $P(x, y)$ de l'image initiale, se projette dans la nouvelle image aux coordonnées :

$$\begin{aligned} U &= \mathcal{F}(x, y) \\ V &= \mathcal{G}(x, y) \\ H &= \mathcal{H}(x, y) \end{aligned} \tag{5.1}$$

où $\mathcal{F}, \mathcal{G}$ et $\mathcal{H}$ sont des fonctions, tout point (x,y) ayant une et image unique. De plus, H est une coordonnée optionnelle utilisée seulement dans le cas des transformations tridimensionnelles.

Ce type d'algorithme s'insère à deux niveaux de notre chaîne de traitement des images satellites. Il peut être utilisé lors du redressement des images satellites brutes en géométrie épipolaire. Ce redressement n'est autre qu'une transformation des images initiales en fonction d'un modèle précis dépendant de la géométrie du système de vision. Il peut aussi être mis en œuvre pour la visualisation des terrains texturés abordée au chapitre précédent. En effet, cette visualisation peut aussi être décrite comme une transformation tridimensionnelle particulière de l'image de la texture. Il suffit d'associer à chaque pixel de texture ses nouvelles coordonnées dans l'image finale en fonction de la projection du triangle auquel il appartient dans le terrain.

Comme nous l'avons vu tout au long des chapitres précédents, les données que nous utilisons ont des tailles relativement importantes ce qui requiert une grande quantité de calculs et de mémoire. Les transformations décrites ci-dessus n'échappent pas à cette règle et nous proposons donc encore une fois le recours au parallélisme.

Par conséquent, cette étude décrit et compare trois algorithmes parallèles pour calculer n'importe quelle transformation d'image vérifiant les conditions données en (5.1). Nous gardons le modèle algorithmique MIMD ainsi que la bibliothèque PPCM pour la programmation parallèle. Nous proposons 3 variantes d'un schéma général en nous focalisant toujours sur l'équilibrage des charges et les structures de données pour les communications.

Comme nous l'avons déjà noté à plusieurs reprises, PPCM permet d'utiliser différentes topologies virtuelles pour les communications entre processeurs. Dans le cadre de notre étude, nous nous sommes aussi intéressé à l'adéquation d'une topologie virtuelle donnée par rapport à celle réellement employée dans la machine parallèle. Pour cela, nous comparons les performances de deux topologies, l'anneau et le graphe complet sur la machine Cray T3D disposant d'un tors 3D.

Notre but final est donc de pouvoir déduire, à partir de nos résultats expérimentaux, la meilleure combinaison entre algorithme parallèle et topologie virtuelle pour une machine parallèle donnée.

Dans ce chapitre, nous donnons tout d'abord un bref aperçu de la littérature concernant le domaine des transformations d'image. Nous en déduisons notre schéma général parallèle et décrivons ses 3 variantes avec leurs résultats expérimentaux. Enfin, nous abordons le problème des schémas de communication, aussi étayé par des tests sur machine.

5.1 Transformations géométriques d'image

Depuis le développement des ordinateurs graphiques, les transformations d'image ont continuellement suscité un grand intérêt de la part des chercheurs. Ainsi, un nombre important d'études ont été menées dans le domaine séquentiel (voir par exemple [CS80, TKKW86, Hec89, She92, BB93, Zha96, Li96]). Tous ces travaux ont menés à plusieurs approches plus ou moins distinctes. Parmi celles-ci, on peut noter les techniques discrètes, les méthodes floues ou encore les algorithmes basés sur le traitement ligne par ligne (*scanline*). Cependant, trois grandes stratégies se dégagent de cet ensemble ([Wol90, FvDFH90]) :

- calcul direct (*forward mapping*),
- calcul inverse (*inverse mapping*),
- calcul décomposé (*separable mapping*).

Dans la première approche, la destination de chaque pixel est calculée et sa couleur est affectée en utilisant un tableau d'accumulation. En effet, comme plusieurs pixels de départ peuvent avoir la même destination, il faut prendre en compte la contribution de chacun au fur et à mesure. Ainsi, on a l'assurance qu'un pixel de l'image de sortie est complètement calculé seulement lorsque tous les pixels de l'image de départ ont été traités. Cette méthode est très bien adaptée aux transformations de type réduction, mais peut produire des trous lorsque l'on effectue des agrandissements. Cela peut néanmoins être évité mais requiert un échantillonage supplémentaire. Une autre variante consiste à considérer les pixels de départ comme des éléments de surface carrés qui sont déformés en quadrilatères quelconques dans l'image de sortie (voir figure 5.1).

La méthode de calcul inverse est plus communément utilisée. C'est l'image finale qui est parcourue et pour chaque pixel, on calcule son antécédent dans l'image initiale en utilisant la transformée inverse. Là aussi, plusieurs variantes existent pour calculer sa couleur finale. La plus simple et rapide consiste à prendre la couleur du pixel le plus proche de l'antécédent dans l'image initiale. Il

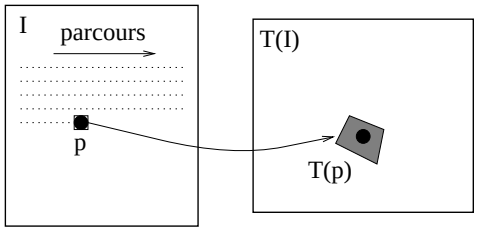

FIG. 5.1 - *Calcul direct en considérant chaque pixel initial comme un polygone*

est aussi possible de prendre en compte les couleurs de ses quatre plus proches voisins en effectuant une interpolation bilinéaire. Comme dans le calcul direct, ces approches présentent des problèmes d'échantillonage car certains pixels initiaux peuvent être *perdus* dans le cas des réductions d'image. La solution est encore de considérer la déformation (inverse) des pixels carrés de l'image finale et de calculer leur couleur en fonction de celles des pixels initiaux se trouvant dans le quadrilatère résultant (voir figure 5.2). Une simplification de cette technique (appelée *mip-mapping*) revient à précalculer l'image initiale à différentes résolutions puis à calculer la couleur du pixel de sortie avec la résolution appropriée selon la taille du quadrilatère obtenu. Un avantage de la technique de calcul inverse est que chaque pixel de l'image finale est entièrement calculé dés qu'il a été parcouru et traité par le processus de calcul. Par contre, le principal défaut est que cette méthode demande de connaître la transformée inverse, ce qui n'est pas toujours possible dans un cadre général.

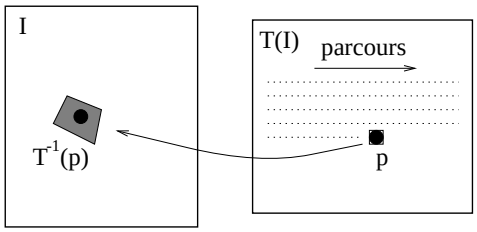

FIG. 5.2 - *Calcul inverse en considérant chaque pixel final comme un polygone*

La dernière méthode consiste à décomposer la transformation en une série de transformations unidimensionnelles (horizontales et verticales). L'intérêt est de rendre plus simple et rapide la phase d'échantillonage. Cette technique procède donc en calculant autant d'images intermédiaires que nécessaire, obtenues uniquement par transformation unidimensionnelle de l'image précédente. La figure 5.3 montre un exemple simple où il n'y a qu'une image intermédiaire. Cette approche peut néanmoins demander des décompositions complexes et pas toujours exprimables algébriquement, ce qui implique l'utilisation de tables pour stocker les coordonnées de destination des pixels (*spatial lookup tables*) et calculer de manière implicite les transformations intermédiaires. Pour cela, on applique chaque transformation non seulement à l'image mais aussi à la table des coordonnées, et l'on en déduit les destinations des pixels pour la prochaine transformation.

Si l'on s'intéresse maintenant aux approches parallèles, elles augmentent encore les possibilités d'obtenir des algorithmes efficaces. Cependant, moins d'études ont été menées dans cette voie, on peut citer par exemple [Yam85, Whi92, GW93]. La plupart des algorithmes parallèles sont basés sur l'un des trois modèles séquentiels présentés ci-dessus. Selon la stratégie utilisée, soit l'image

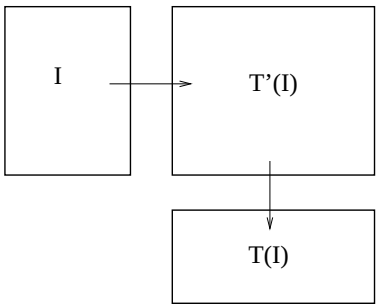

FIG. 5.3 - *Calcul par décomposition en transformations unidimensionnelles*

initiale, soit l'image finale est divisée et répartie sur les processeurs.

Pour le calcul direct, l'image initiale est divisée et chaque processeur calcul la transformation de la sous-image qui lui est associée. Ensuite, une étape de fusion des images partielles obtenues est nécessaire car celles-ci peuvent se chevaucher.

Concernant le calcul inverse, c'est l'image finale qui est divisée. Chaque processeur calcule sa sous-image associée mais la phase de fusion des images résultantes est implicite car, dans cette méthode, les images partielles résultantes sont distinctes. Le problème principal de cette technique vient du stockage de l'image initiale : soit elle est dupliquée sur tous les processeurs, ce qui rend les calculs plus faciles mais consomme beaucoup de mémoire, soit elle est distribuée sur les processeurs, ce qui implique des accès concurrents à ces données.

Enfin, dans le cas des décompositions, l'image initiale est divisée et distribuée. La distribution ne pose pas les problèmes de la méthode précédente. En effet, la nature unidimensionnelle des transformations implique que le calcul d'une ligne (colonne) de l'image finale requiert uniquement les données de cette ligne (colonne) de l'image initiale. Cependant, cette distribution doit être répétée pour chaque transformation intermédiaire. La fusion finale est là aussi implicite puisque la partition de l'image I_n à l'étape n est conservée dans l'image I_{n+1}.

Nous avons donc décrit les principaux schémas algorithmiques utilisés pour les déformations d'image. À partir de cela, nous construisons un schéma parallèle général et décrivons trois variantes possibles de ce schéma.

5.2 Transformation parallèle d'image

Dans la suite de ce chapitre, nous reprenons le même modèle de machine parallèle que celui donné au chapitre 3 et les mêmes notations.

Comme les algorithmes vus dans les chapitres précédents, celui-ci est basé sur un équilibrage des charges dirigé par les données. L'approche adoptée ici est de diviser l'image de sortie pour distribuer le travail et les données sur les processeurs.

5.2.1 Équilibrage des charges

Nous avons déjà vu à plusieurs reprises l'intérêt de l'équilibrage des charges. Dans le contexte des transformations d'image, on retrouve plusieurs stratégies plus ou moins efficaces pour réaliser un tel équilibrage.

Une première idée consiste à diviser l'image d'entrée équitablement. Néanmoins, en plus des problèmes de fusion mentionnés plus haut, cette technique n'assure pas un bon équilibre des charges. En effet, le travail à fournir n'est pas seulement fonction de la quantité de données que l'on traite mais aussi de leur agencement dans l'image finale. Supposons que nous divisions l'image en deux bandes horizontales et que nous ayons une transformation qui agrandit la partie haute de l'image et réduit l'autre partie. On constate que les deux processeurs auront des charges de travail bien différentes.

On peut alors essayer de diviser équitablement l'image de sortie. Cette approche est meilleure car elle ne contient pas d'étape de fusion. Malgré tout, elle n'assure pas non plus un bon équilibrage des charges car la répartition du travail dans l'image finale n'est pas forcément (et même rarement) homogène. On peut donc avoir une grande partie du travail concentré dans une zone de l'image et presque rien dans le reste de l'image.

Finalement, on arrive vite à la même conclusion que pour l'algorithme de visualisation des terrains. les techniques précédentes ne sont pas efficaces car elles ne tiennent pas compte de la répartition du travail. La méthode la plus appropriée est donc celle où l'image de sortie est divisée en parties ayant approximativement la même quantité de travail. Cela implique de connaître la répartition du travail dans l'image, mais il serait trop coûteux en temps de calculer précisément cette répartition. Nous utiliserons donc le même type d'heuristique que dans le chapitre précédent. Dans le cas présent, le module élémentaire pris comme unité de travail est une ligne de l'image de sortie, et l'estimation de la charge d'une ligne est effectuée en comptant le nombre de pixels de l'image initiale qui se projette sur cette ligne. Nous avons fait ce choix car le coût de calcul d'un pixel final est directement lié au nombre de pixels initiaux qu'il met en jeu. Ensuite, nous pouvons calculer la partition finale en utilisant l'algorithme élastique.

5.2.2 Schéma général de l'algorithme parallèle

Maintenant que nous avons fixé notre stratégie d'équilibrage, nous pouvons en déduire le schéma général de notre algorithme parallèle de transformation d'image. Celui-ci a donc la structure sui-

vante :

1. charger l'image sur les processeurs,
2. sur chaque processeur :
   (a) calculer les destinations des pixels contenus dans la mémoire locale du processeur (U,V et éventuellement H),
   (b) calculer la contribution de ses pixels sur la charge des lignes de l'image finale,
   (c) fusionner les charges partielles obtenues sur les processeurs et calculer la partition de l'image finale,
   (d) redistribuer les données en fonction de la partition,
   (e) calculer la bande d'image finale,
3. sauvegarder ou afficher l'image,

Les trois approches que nous allons étudier diffèrent aux étapes 1, 2d et 2e. Par soucis de clarté, nous allons nommer chacune de ces variantes et nous ferons référence à ces noms lorsque cela sera nécessaire :

1. DUP1P : les données sont dupliquées sur les processeurs aux étapes 1 et 2d. L'étape 2e est effectuée par un algorithme en une passe utilisant une variante du calcul direct avec interpolation bilinéaire. Dans cette variante, on utilise aussi des polygones plutôt que des pixels. Mais les polygones que l'on utilise sont formés par quatre pixels voisins dans l'image initiale plutôt que par une zone carrée autour de chaque pixel (voir figure 5.4). Chaque polygone est projeté dans l'image finale et dessiné d'une manière similaire à celle du plaquage de texture réalisé au chapitre précédent. Cela permet de transformer les polygones directement

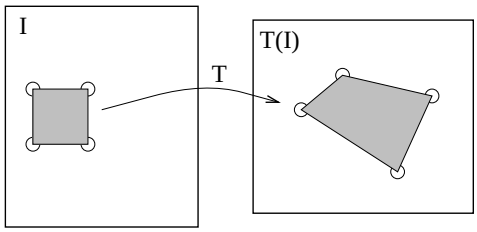

FIG. 5.4 - *Calcul direct utilisant des polygones formés par quatre pixels voisins*

en calculant les destinations des pixels, et évite ainsi le calcul des sommets des carrés qui ne sont pas directement disponibles.

2. DIST1P : les données sont complètement distribuées et l'étape 2e est réalisée avec le même procédé que DUP1P.

3. DIST2P : cette approche est similaire à la seconde mais l'étape 2e est effectuée en utilisant un algorithme en deux passes par décomposition en transformations unidimensionnelles. Ce type d'algorithme est relativement bien connu en traitement d'image ([TKKW86, FvDFH90, Wol90]). Si l'on considère l'image initiale I et une fonction de transformation 2D :

$$\mathcal{T}(x(i,j), y(i,j)) : \mathcal{Z}^2 \rightarrow \mathcal{R}^2 \tag{5.2}$$

On décompose alors T en deux fonctions $\mathcal{U}$ et $\mathcal{V}$ ne modifiant chacune la position des pixels que selon un des deux axes. Nous avons donc :

$$\begin{aligned} \mathcal{V}_{\circ}\mathcal{U} &= \mathcal{T} \\ \mathcal{U}(i,j) &= (x(i,j), j) \\ \mathcal{V}(i',j') &= (i', \phi(i',j')) \end{aligned} \tag{5.3}$$

Donc, la première passe calcule une image intermédiaire I', en changeant seulement les abscisses des points par application de $\mathcal{U}$. Une interpolation linéaire est effectuée entre les pixels d'une même ligne. Cette première passe peut être réalisée entre les étapes 2c et 2d. En effet, à ce moment là de l'algorithme, l'image est équitablement distribuée en bandes horizontales sur les processeurs. Or, l'application de $\mathcal{U}$ demande la même quantité de travail d'une ligne à l'autre. On constate donc que le processus est complètement équilibré si on le fait juste avant la redistribution des données.

La seconde passe génère l'image finale à partir de I' en ne changeant que les ordonnées des pixels par application de $\mathcal{V}$. Cependant, si la fonction $\mathcal{U}$ est directement déductible de $\mathcal{T}$, ce n'est pas le cas pour $\mathcal{V}$. En effet, sa définition donnée dans (5.3) implique que $\phi(i',j') = y(x^{-1}(i',j'))$. Il n'est donc pas toujours possible d'avoir une expression mathématique de ϕ. Malgré tout, comme expliqué au paragraphe 5.1, il est possible de la calculer pour chaque pixel (i',j') de l'image intermédiaire en appliquant $\mathcal{U}$ sur les ordonnées dans la table de destination lors de la première passe. Ces ordonnées transformées peuvent ensuite être utilisées pour faire la seconde passe. Cette seconde étape doit être réalisée après la redistribution des données puisque la répartition verticale du travail est affectée.

Toutes les étapes du schéma algorithmique sont décrites dans les paragraphes suivants en faisant ressortir les différences entre les variantes quand cela est pertinent.

5.2.3 Chargement de l'image initiale

Il y a deux manières d'effectuer cette tâche. Soit on duplique l'image sur tous les processeurs (variante DUP1P), soit on la distribue en la divisant en bandes horizontales de mêmes tailles (versions DIST1P et DIST2P).

Duplication

Là aussi, il existe deux grandes façons de procéder. La première consiste à charger l'image complètement sur un processeur puis à utiliser une diffusion pour communiquer l'image aux autres

processeurs. Une deuxième solution est que chaque processeur lise directement l'image depuis le disque, ce qui requiert une synchronisation lorsque les lectures concurrentes sont impossibles. Ces méthodes dépendent surtout de l'architecture de la machine. Dans le cadre de notre étude, elles sont équivalentes puisque nous nous concentrons plus particulièrement sur la partie 2 de l'algorithme.

Distribution

Ici aussi, deux méthodes similaires aux précédentes sont utilisables, la diffusion étant juste remplacée par une distribution. Cependant, si l'on considère que l'image entière ne peut tenir dans la mémoire d'un processeur, la méthode de lecture concurrente doit être utilisée. Et si cela n'est pas possible, il faut alors que le processeur qui peut faire les E/S lise l'image bande par bande et redistribue à la volée celles-ci.

Finalement, l'information importante dans cette étape est la taille de l'espace mémoire mis en jeu pour charger l'image.

Division de l'image initiale

Comme on peut le voir dans l'algorithme général, des précalculs sont nécessaires avant de calculer la partition de l'image finale (étapes 2a,2b). Pour effectuer ces calculs de manière équilibrée, et comme ceux-ci sont équivalent d'une ligne à l'autre, on réparti équitablement l'image initiale sur les processeurs. Dans la version DUP1P, cette subdivision est indépendante de la répartition physique des données. Chaque processeur possède toute l'image mais ne fera les précalculs que sur une partie de celle-ci. Par contre, pour les deux autres versions, cette division logique est équivalente à la répartition physique puisque cette dernière est justement effectuée par rapport à la division logique.

Étant donnée notre méthode de calcul des couleurs en une passe, basée sur une décomposition en polygones de l'image initiale, on se heurte au problème de la perte d'information au niveau des frontières entre les bandes. En effet, les polygones formés par les points de part et d'autre d'une frontière sont perdus si l'on conserve une partition stricte de l'image, comme le montre la figure 5.5. De plus, si la projection d'un tel polygone se retrouvent à cheval sur une frontière de l'image finale, il faut que celui-ci puisse-t-être dessiné convenablement des deux côtés de la frontière. C'est pourquoi les deux lignes de part et d'autre d'une frontière dans l'image initiale doivent être dupliquées sur les deux processeurs concernés. Bien sûr, ces duplications ne sont pas prises en compte lors des précalculs concernant l'estimation de la charge, sinon celle-ci serait faussée.

Enfin, les bandes d'image sont décrites par l'indice de leur première ligne et leur taille (nombre de lignes). Une fois ces informations disponibles, on peut commencer les précalculs.

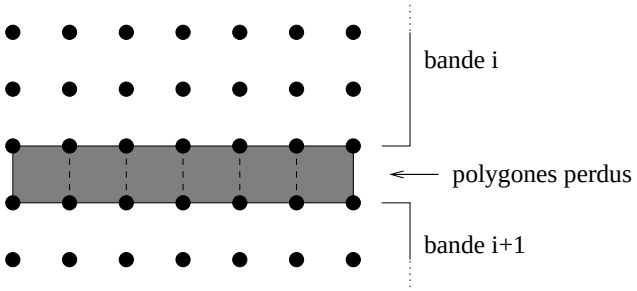

FIG. 5.5 - *Polygones perdus à la frontière entre deux bandes de l'image initiale*

5.2.4 Table de destination

Cette étape est la même pour les trois variantes. Comme explicité plus haut, pour calculer la partition de l'image finale, on doit estimer les charges de travail qui dépendent des destinations des pixels. Celles-ci sont aussi requises pour calculer les dimensions de l'image de sortie lorsque l'utilisateur ne la spécifie pas. Dans ce cas, on prend le rectangle englobant l'ensemble des images des pixels initiaux. La première étape consiste donc à calculer ces destinations en utilisant la transformation directe. Un tableau bidimensionnel ayant la même taille que la bande locale d'image sert à stocker les coordonnées U, V et éventuellement H.

5.2.5 Partition de l'image finale

Cette étape est elle aussi commune aux trois variantes. Sur chaque processeur, la contribution de charge des données locales est calculée sur la hauteur totale de l'image finale en utilisant l'heuristique donnée au paragraphe 5.2.1. Ensuite, toutes les contributions locales sont globalisées en utilisant une communication de type réduction (somme). Enfin, la partition verticale de l'image est déduite à l'aide de l'algorithme élastique décrit au chapitre précédent. Les bandes sont, cette fois, décrites par l'indice de leur première ligne et de leur dernière ligne.

La qualité de la partition en terme d'équilibrage dépend de l'heuristique choisie. Notre choix est relativement fiable compte tenu de la relation étroite entre le nombre de points dans une zone d'image et le nombre de polygones à traiter, donc la quantité de travail. L'image de la figure 5.6 montre un exemple de partition dans le cas de 8 processeurs avec l'histogramme des charges de travail sur la droite.

On constate que les bandes du centre, où la charge de travail est plus importante, sont moins larges que celles du haut et du bas de l'image.

5.2.6 Redistribution des données

Cette étape est effectuée différemment selon la variante de l'algorithme.

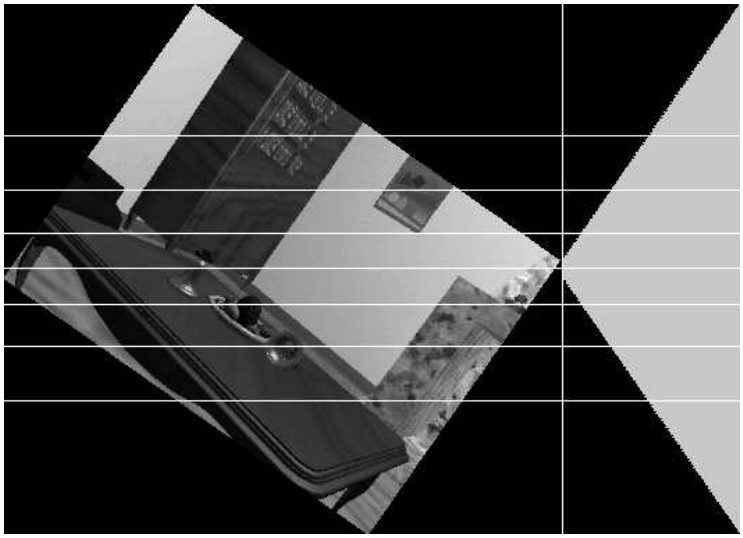

FIG. 5.6 - *Partition de l'image de sortie en fonction de l'histogramme des charges (à droite), dans le cas de 8 processeurs*

Version dupliquée

Comme les données de l'image initiale sont déjà dupliquées, les seules données qui doivent être échangées entre les processeurs sont celles de la table de destination. Comme ces données sont réparties sur les processeurs, il faut utiliser une multi-diffusion pour que chaque processeur envoie ses données à tous les autres et reçoive toutes celles des autres.

Approches distribuées

Ici, les données doivent être redistribuées pour n'avoir sur chaque processeur que celles qui lui sont nécessaires. Comme les deux approches DIST1P et DIST2P n'utilisent pas la même méthode de calcul des couleurs, les redistributions des données sont différentes. C'est pourquoi on les distingue ici.

DIST1P Une solution simpliste consisterait à placer dans le message pour chaque point initial, ses coordonnées dans l'image initiale, ses coordonnées de destination et sa couleur. On a besoin de la position initiale pour pouvoir reconstituer les polygones après l'envoi. Néanmoins, cela donnerait un volume de communication très important.

Il est donc utile de prendre en compte certaines caractéristiques des transformations d'image et des données pour nous permettre de concevoir des messages plus compactes :

Continuité : les transformations considérées ici ont une certaine continuité, c.-à-d. deux pixels voisins dans l'image initiale ont une grande probabilité d'avoir des projections voisines dans l'image finale.

Non injectivité : pour pouvoir traiter des transformations quelconques (telles que des pliages...), on doit prendre en compte le fait que des parties non connexes de l'image initiale peuvent se retrouver dans la même bande de l'image de sortie. Cela implique aussi que des ensembles non connexes de points d'une bande de l'image initiale peuvent se retrouver sur le même processeur destination (voir figure 5.7).

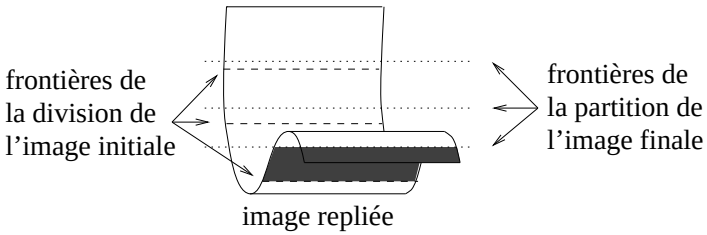

FIG. 5.7 - *Ensembles non connexes de pixels. Les deux zones grises appartiennent à la même bande initiale et arrivent sur la même bande finale*

On se retrouve donc exactement devant le problème déjà rencontré lors de la redistribution des données de l'algorithme de visualisation des terrains. Étant donnée la similitude entre les structures de données que l'on manipule dans ces deux algorithmes (images 2D), on peut essayer de réutiliser la même méthode. On va donc construire les messages en décrivant les données envoyées à un même processeur par l'ensemble des régions connexes qu'elles constituent. Chaque région étant elle même décrite par l'ensemble des segments horizontaux qui la composent. Un exemple de cette description est donné dans la figure 5.8.

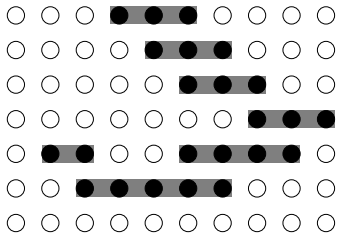

FIG. 5.8 - *Ensemble de pixels (noir) constituant une région connexe décrite par segments horizontaux (gris)*

Enfin, les informations nécessaires pour chaque segment sont la position de son premier point et le nombre de pixels qu'il contient. Comme on ne sait pas a priori combien de segments vont être construits, on utilise des listes chaînées pour les stocker. Une liste étant construite pour chaque processeur destination.

Cette technique de codage permet d'économiser une grande quantité de mémoire puisque les coordonnées des pixels sont implicites dans chaque segment. Par conséquent, La redistribution est réalisée en deux temps :

– la construction des segments,

– la construction des messages.

La première étape se fait en parcourant la table de destination. Si le pixel courant se projette dans la même bande d'image que son prédécesseur (appartenant au segment en cours de construction), alors il est ajouté à ce segment en incrémentant son nombre de points. Sinon, on crée un nouveau segment en partant de ce point. Ce procédé produit une partition de la sous-image locale à chaque processeur. Cependant, pour ne pas perdre les polygones qui se trouveront à cheval sur plusieurs bandes de l'image finale (comme dans l'algorithme de visualisation de terrains), il faut rajouter à ces ensembles, tous les points voisins dans la 8-connexité. Il faut donc rajouter les deux points qui se trouvent aux extrémités de chaque segment ainsi que les segments au-dessus et au-dessous de l'ensemble, comme le montre la figure 5.9.

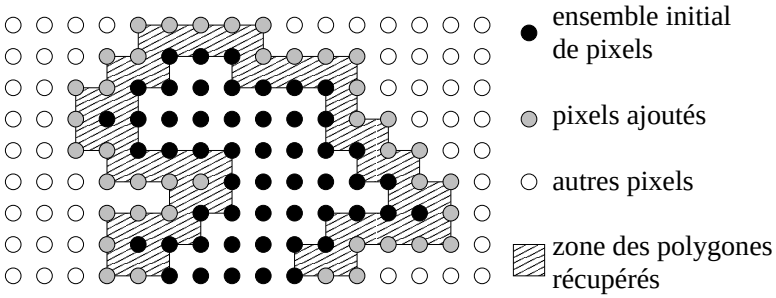

FIG. 5.9 - *Pixels à rajouter à un ensemble initial de points*

Cette première opération demande plusieurs étapes. Nous parcourons tout d'abord la liste des segments initiaux du haut vers le bas pour rajouter les pixels se trouvant au dessus de chaque segment et n'appartenant pas déjà à l'ensemble. Ensuite, un second parcours est effectué dans le sens inverse pour ajouter cette fois les pixels au-dessous de chaque segment. Enfin, un dernier parcours permet d'ajouter les pixels aux extrémités des segments et de fusionner les segments qui se chevauchent ou qui sont contigus. Tout ce processus est résumé dans la figure 5.10.

Comme on peut le voir dans la quatrième ligne de la dernière étape, deux segments peuvent n'être séparés que par quelques pixels. Dans ce cas, la fusion des deux segments peut quand même avoir lieu si leur éloignement est inférieur à un certain seuil (1 ou 2 pixels).

La seconde opération du processus consiste à construire les messages en fonction des listes des segments. On génère un message contenant les données pour chaque destination possible (NbP). Comme il est possible de n'avoir aucune donnée à transférer à certaines destinations, les messages correspondants seront donc vides. Concernant la structure des messages, on reprend la description des segments vue ci-dessus (position, nombre de points). Puis, pour chaque point du segment, il faut indiquer ses coordonnées de projection et sa couleur. On aurait pu, là aussi, ne pas envoyer les coordonnées de projection mais cela impliquerait de les recalculer après la réception du message, ce qui est plus coûteux encore que d'avoir un message plus important. La figure 5.11 indique la structure des messages qui en découle, dans le cas d'un transformation 2D. Si nous avions une fonction 3D, il faudrait rajouter la coordonnée H entre V et C.

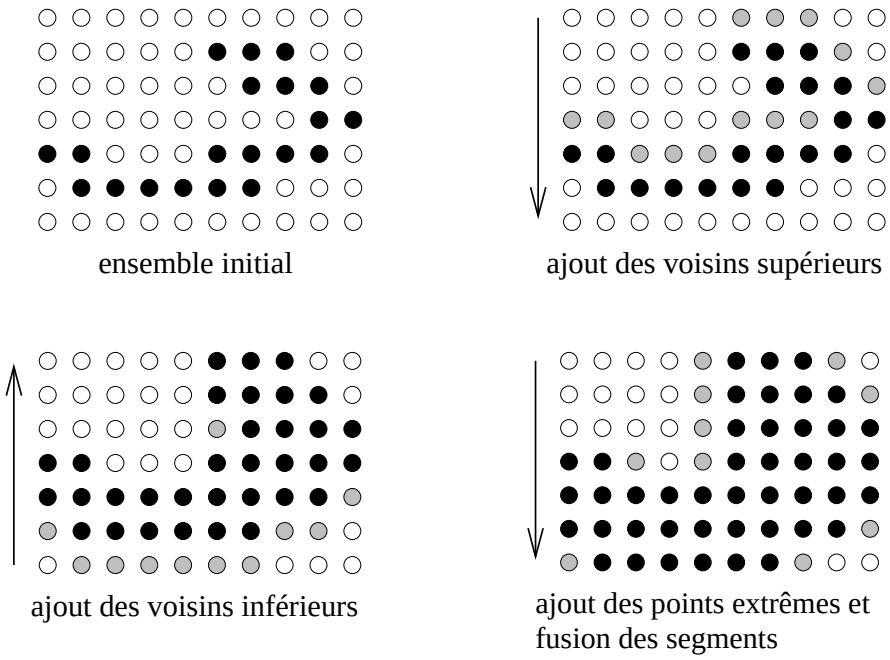

FIG. 5.10 - *Les 3 étapes nécessaires pour ajouter tous les points voisins d'un ensemble initial. Les flèches indiquent le sens de parcours*

Nombre de segments	X^1	Y^1	Nombre de points dans le segment 1	U_1^1	V_1^1	C_1^1	U_2^1	.....	X^i	Y^i	Nombre de points dans le segment i	U_1^i	V_1^i	C_1^i	.....

FIG. 5.11 - *Structure des messages pour la redistribution des données, dans le cas de l'algorithme en une passe et pour une transformation 2D*

DIST2P La redistribution est légèrement différente de la précédente. On conserve la description par segments des régions d'image à communiquer. Mais ici, on n'a plus besoin d'ajouter les pixels voisins puisque l'on n'utilise pas de polygones. De plus, la première passe étant réalisée avant la redistribution, et la seconde passe ne modifiant que les ordonnées des pixels, il est plus adéquat de décomposer les ensembles de points en segments verticaux plutôt que horizontaux. Ainsi, l'ordonnée du premier point d'un segment n'est plus utile. De même, les destinations en abscisse sont toutes identiques pour les points d'un segment, on peut donc donner uniquement celle du premier point. Par conséquent, les messages obtenus (figure 5.12) sont plus simples et plus courts que dans la version DIST1P.

Nombre de segments	X_1	Nombre de points dans le segment 1	Y_1^1	C_1^1	Y_2^1	C_2^1	.....	X_i	Nombre de points dans le segment i	Y_1^i	.....

FIG. 5.12 - *Structure des messages pour la redistribution des données, dans le cas de l'algorithme en deux passes et pour une transformation 2D*

Enfin, les messages ainsi obtenus sont échangés à l'aide d'une multi-distribution.

5.2.7 Calcul des couleurs

La phase de calcul des couleurs des pixels de l'image finale se fait en deux temps :

- décodage des messages,
- calcul des pixels.

Le décodage des messages est quasi inexistant dans la variante DUP1P. Les messages contiennent effectivement toute la table de destination. Il suffit donc de parcourir la table et de ne prendre en compte que les pixels se projetant dans la bande d'image associée au processeur.

Pour les deux autres versions, cette phase est un peu plus complexe. Les messages reçus sont composés de segments comme on peut le voir dans les figures 5.11 et 5.12. Par conséquent, le parcours est effectué segment par segment. Dans la version DIST2P, chaque segment décodé peut être directement traité pour dessiner sa transformation dans l'image finale. Pour la version DIST2P, il faut reconstituer les polygones initiaux. Pour chaque segment du message, il faut trouver tous les segments qui se trouvent sur la ligne suivante dans l'image initiale et qui ont une intersection non vide avec le segment courant sur l'axe des abscisses. Ensuite, les polygones reconstitués sont dessinés à la volée dans la bande d'image locale.

La deuxième étape qui concerne le calcul des pixels a déjà été détaillée au paragraphe 5.2.2.

5.2.8 Sauvegarde / Affichage

Une fois que tous les processeurs ont fini leur travail, l'image résultante peut être soit sauvegardée sur disque, soit affichée à l'écran. Pour la sauvegarde, un synchronisation est nécessaire entre les processeurs pour que les bandes d'image soient écrites dans le bon ordre. En ce qui concerne l'affichage à l'écran, chaque processeur a juste à envoyer sa bande d'image sur l'écran, à la bonne position dans l'image finale.

5.3 Résultats

Nos expériences ont été réalisées sur la machine Cray T3D. Comme nous l'avons indiqué précédemment, notre algorithme de transformation d'images peut gérer n'importe quelle déformation exprimable sous la forme donnée dans 5.1. Pour avoir des résultats comparables avec les autres études menées dans ce domaine, nous avons choisi la rotation qui est un traitement très courant.

Comme pour les autres algorithmes que nous avons étudiés dans les chapitres précédents, on évalue ici les caractéristiques les plus importantes pour un algorithme parallèle.

On commence par comparer les vitesses de nos trois variantes. La figure 5.13 montre les temps d'exécution pour tourner une image en fonction du nombre de processeurs.

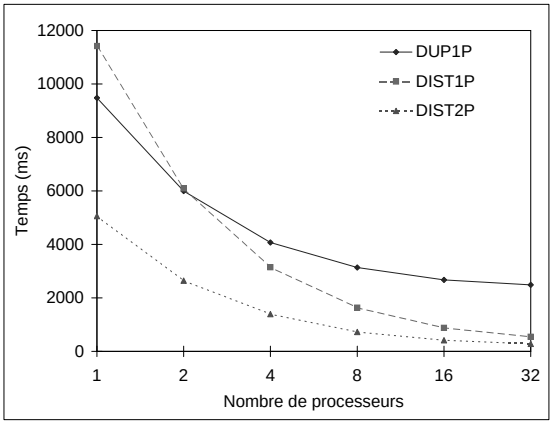

FIG. 5.13 - *Temps de calcul de rotation de 35 degrés d'une image 640×480 en fonction du nombre de processeurs*

Le point commun des trois courbes est qu'elles décroissent avec le nombre de processeurs. Cela dénote simplement que notre schéma général est relativement bon. Par contre, ces courbes sont assez différentes les unes des autres. On constate que la variante DUP1P n'est pas vraiment extensible. Cela est principalement du au volume beaucoup plus important de données communiquées lors de la redistribution, ainsi qu'au décodage des messages qui est intrinsèquement inefficace.

Par contre, cette version est légèrement meilleure que DIST1P sur peu de processeurs. Cela vient du codage et du décodage des messages. En effet, la variante DIST1P possède ces deux étapes de plus lors de la redistribution. Les deux variantes utilisant la même méthode de calcul et les quantités de données à traiter par processeur étant quasiment les mêmes entre les deux versions. Le traitement des messages crée un surcoût par rapport à la version DUP1P qui n'est pas compensé sur peu de processeurs. Ce problème n'apparaît pas avec la version DIST2P car le calcul des couleurs est bien plus rapide et compense largement ce surcoût de codage et décodage des messages.

Si l'on compare maintenant les deux versions distribuées, on peut voir que les temps d'exécution se rapprochent quand le nombre de processeurs augmente. On sait que dans ce type d'algorithme, la proportion du temps de calcul d'image devient de moins en moins importante quand le nombre de processeurs augmente. Or, étant donnée la rapidité séquentielle de l'algorithme en deux passes, sa version parallèle atteint plus vite ses limites imposées par les précalculs et les communications. Pour la version en une passe, le comportement est identique mais repoussé à un nombre de processeurs plus important à cause des calculs d'image plus lents. La figure 5.14 met en évidence les répartitions des temps de calculs pour les deux variantes distribuées en fonction du nombre de processeurs.

Comme dans l'algorithme de visualisation de terrains, il est clair que les tailles des images d'entrée et de sortie jouent un rôle majeur sur le comportement de l'algorithme.

Si l'image initiale était plus grande, les précalculs et les communications le seraient aussi alors que les calculs de l'image finale ne changeraient pas énormément. L'algorithme atteindrait alors plus rapidement ses limites et présenterait donc une extensibilité moins bonne. Maintenant, si c'est l'image de sortie qui est plus grande, les précalculs et les communications ne changeraient pas alors

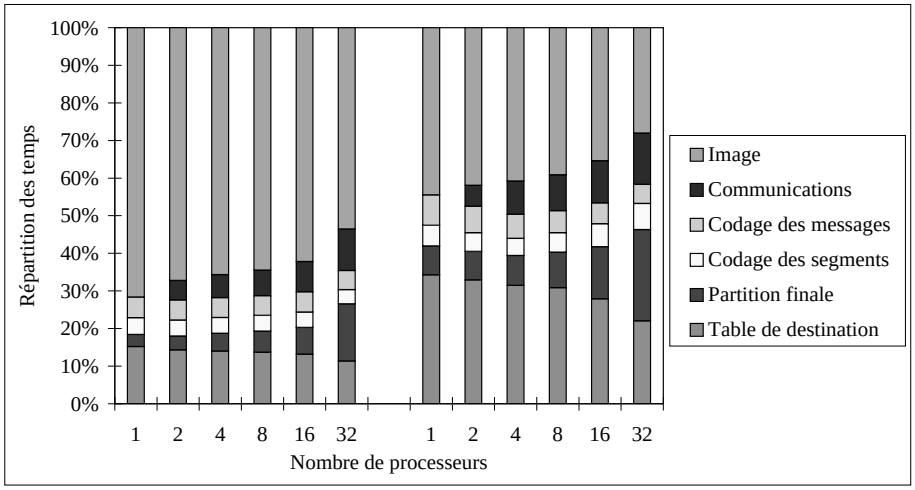

FIG. 5.14 - *Répartition des temps de calcul des deux versions distribuées en fonction du nombre de processeurs : DIST1P à gauche, et DIST2P à droite*

que le temps de calcul de l'image finale augmenterait, impliquant une extensibilité meilleure. On en déduit que le comportement de l'algorithme dépend directement du ratio entre les tailles des images. Le cas idéal étant d'avoir une image de sortie plus grande que celle d'entrée.

Finalement, la figure 5.15 donne les accélérations obtenues par les trois variantes. L'inflexion des courbes ainsi que leurs positions relatives (DIST1P au-dessus de DIST2P) confirme nos observations précédentes. On peut aussi constater que la version dupliquée n'est réellement pas extensible.

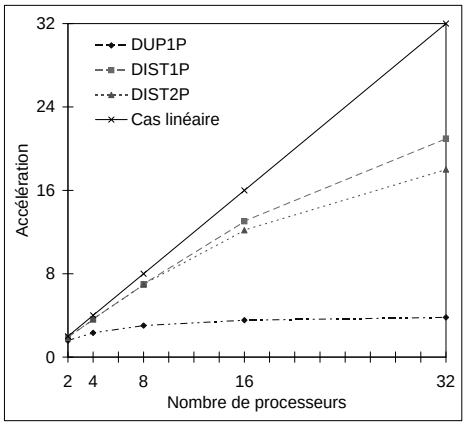

FIG. 5.15 - *Accélération des trois variantes en fonction du nombre de processeurs*

5.3.1 Conclusion

En conclusion, l'approche qui consiste à dupliquer toutes les données sur les processeurs n'est pas adaptée aux contraintes de mémoire et de temps nécessaires à l'obtention d'un algorithme extensible. Les versions distribuées sont, quant à elles, nettement plus efficaces. Nous obtenons deux

algorithmes extensibles en temps mais aussi en mémoire. Cela nous permet de pouvoir considérer le traitement d'images de grandes tailles, telles que celles provenant du satellite SPOT. Les meilleurs temps absolus sont obtenus avec l'algorithme en deux passes, mais celui-ci est limité par l'ensemble plus réduit des transformations qu'il peut traiter (à cause de la décomposition). On peut donc combiner les deux approches distribuées pour effectuer les calculs en deux passes quand cela est possible, sinon revenir à l'algorithme en une passe qui présente tout de même des performances similaires quand le degré de parallélisme augmente. Cette étude confirme aussi l'influence du rapport entre les tailles des données lues et générées par l'algorithme parallèle sur ses performances.

5.4 Comparaison des schémas de communication

Comme nous l'avons déjà vu auparavant, une des originalités de l'environnement PPCM est de permettre l'utilisation de topologies différentes. Clairement, on sait que la topologie virtuelle la mieux adaptée à une machine parallèle est celle effectivement utilisée dans cette machine. Cependant, même si PPCM propose plusieurs topologies, il ne possède pas toutes celles qui existent. Vu le rôle majeur des communications dans nos algorithmes parallèles, il est donc intéressant d'évaluer l'importance du choix de la topologie virtuelle (ou logique) en fonction de la topologie réelle de la machine utilisée. Ce travail va donc nous permettre d'estimer les contentions dans les communications et de déduire la meilleure topologie à utiliser, parmi celles disponibles, sur une machine donnée.

Étant données les fonctions de communication que nous utilisons dans nos algorithmes et la topologie de la machine Cray T3D (tors 3D) sur laquelle nous faisons cette étude, deux schémas semblent intéressants à comparer : l'anneau et le graphe complet. Sur l'anneau, chaque processeur ne peut envoyer et recevoir des données que de ses deux voisins (gauche et droite). Sur le graphe complet, tous les processeurs sont directement accessibles entre eux.

Nos expériences portent sur deux types de communication : un envoi point à point et une multi-distribution car notre algorithme parallèle l'utilise particulièrement. Enfin, nous nous sommes référés à [DD95] pour les caractéristiques des communications de la machine T3D (latence et bande passante).

Le graphe de la figure 5.16 indique les temps pour envoyer un très petit message (1 octet) à partir d'un processeur, en fonction du processeur destination. La petite taille du message va nous permettre d'estimer le nombre de nœuds (processeurs) traversés par le message, pendant la communication.

L'histogramme de gauche indique les temps obtenus avec le graphe complet. On peut voir que ceux-ci sont légèrement sensible à la distance entre les processeurs mais dans des proportions assez réduites. Par contre, les temps obtenus avec l'anneau, indiqués à droite sur la figure, révèlent une sensibilité bien plus importante en fonction de la distance de la destination. La forme générale de l'histogramme est conforme à l'anneau puisque les temps augmentent jusqu'à la distance maximale entre deux processeurs, qui correspond au diamètre de l'anneau ($\lfloor \frac{NbP}{2} \rfloor$). Néanmoins, les temps absolus sont bien plus importants que ceux du graphe complet.

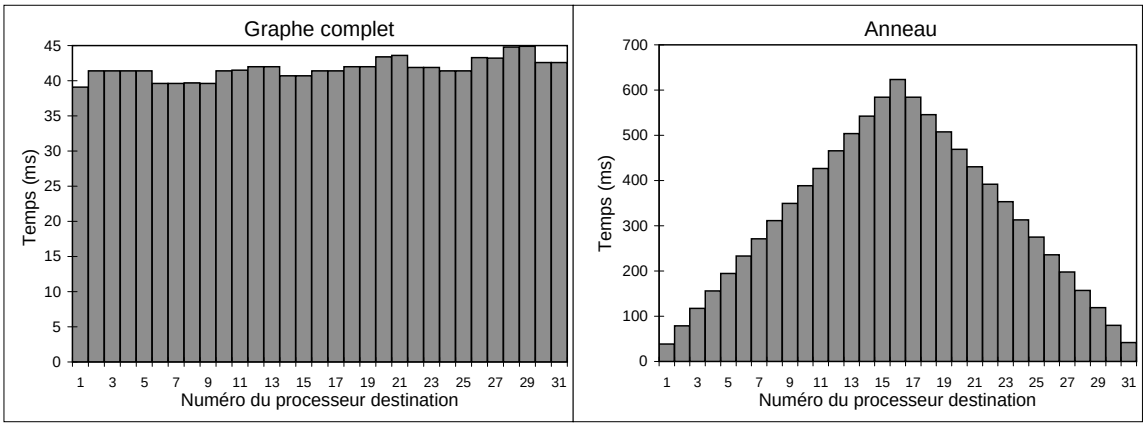

FIG. 5.16 - *Temps de transfert d'un octet à partir du processeur 0, en fonction du nœud destination*

Ces observations peuvent être expliquées de la manière suivante : dans le cas du graphe complet, le message est directement envoyé vers sa destination, et c'est le routage interne de la machine qui se charge de le faire parvenir le plus rapidement possible. Dans le cas de l'anneau, nous forçons le message à passer par les processeurs qui sont entre la source et la destination dans le schéma virtuel. De plus, l'assignation logique des processeurs n'a aucun lien direct avec leur disposition réelle car nous n'avons pas de contrôle sur celui-ci. Donc, deux processeurs logiquement voisins peuvent ne pas l'être physiquement. Ainsi, le message peut être dévié de sa route optimale.

Une dernière observation que l'on peut faire sur ces histogrammes est que leur forme ne dépend pas directement de la taille des messages. Lorsque la taille est plus importante, les temps absolus sont logiquement plus élevés mais la forme des histogrammes ne change pas.

Maintenant que nous avons vu une communication simple, nous devons compléter notre étude par des tests sur une communication globale.

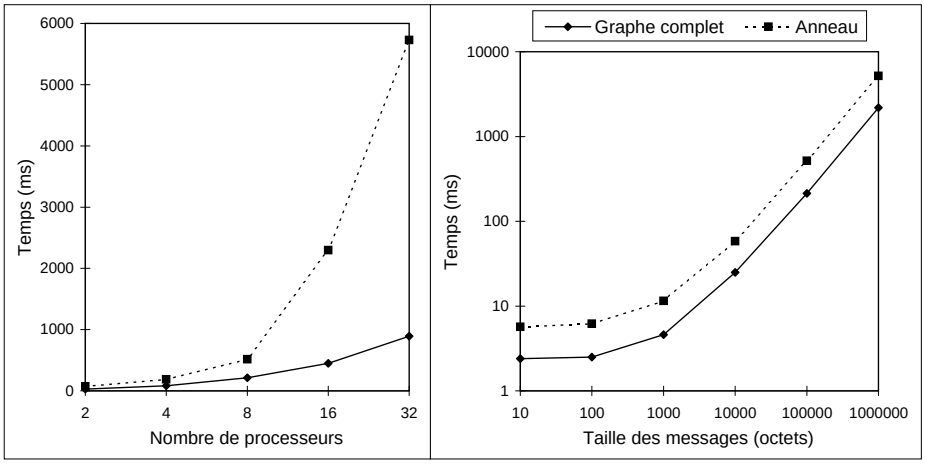

FIG. 5.17 - *Temps pour une multi-distribution. En fonction du nombre de processeurs et pour un message de 100000 octets (à gauche). En fonction de la taille du message et sur 8 processeurs (à droite)*

Sur la figure 5.17, deux types de mesures sont représentées. Sur la gauche, on a mesuré les temps de communication en fonction du nombre de processeurs et pour une taille de message fixe. La courbe quadratique correspond bien à la complexité de la multi-distribution en $O(NbP^2)$, en termes d'envois/réceptions. Encore une fois, et pour les mêmes raisons, la topologie en anneau donne de moins bons résultats.

Sur le graphique de droite, on peut voir l'évolution des temps de communication en fonction de la taille des messages et pour 8 processeurs. Il est intéressant de voir que les deux topologies ont le même comportement à une translation près. La translation vient, encore une fois, des différences d'efficacités des deux routages.

5.4.1 Conclusion

On se rend compte que les temps de communications ne sont pas les mêmes selon la topologie virtuelle que l'on choisi. De plus, l'écart important entre les temps obtenus avec les deux topologies testées nous prouve que ce choix se révèle être primordial sur les performances globales de l'algorithme. On a d'ailleurs déjà pu constater cette influence sur notre algorithme parallèle de reconstruction de terrains.

Concernant le choix en lui-même, il doit privilégier au maximum les routages internes et les communications locales. Il faut donc utiliser la topologie se rapprochant au maximum de l'architecture réelle de la machine cible.

5.5 Conclusion

Dans ce chapitre, nous avons étudié un algorithme parallèle de transformation géométrique d'image. Nous avons présenté trois variantes du schéma algorithmique initial. Comme dans les chapitres précédents, nous nous sommes particulièrement intéressé à l'équilibrage des charges et à la représentation des données pour obtenir des communications réduites. Les différences entre ces trois variantes se situent donc au niveau de la gestion de la mémoire (duplication ou distribution des données) et au niveau du calcul de l'image (algorithme en une passe ou en deux passes).

La comparaison des ces algorithmes au travers de résultats expérimentaux nous a permis de statuer sur l'efficacité de chacun d'eux. Il apparaît clairement que la duplication des données doit être évitée au maximum. Par contre, les deux versions distribuées présentent de bonnes performances et peuvent être utilisées conjointement en fonction de la transformation à effectuer.

Cette étude a aussi permis de confirmer les observations faites dans les chapitres précédents, concernant le rôle clé de certains paramètres par rapport aux performances des algorithmes parallèles. Dans le cas des transformations d'image, on retiendra le rapport entre les tailles des images initiale et finale.

Enfin, nous avons comparé le comportement de deux topologies virtuelles (anneau et graphe complet) sur une machine Cray T3D organisée selon un tors 3D. Il en est ressorti l'importance

du choix de la topologie utilisée sur les performances des communications et donc sur celles de l'algorithme général.

En conclusion, les transformations géométriques d'images peuvent être réalisées efficacement de manière parallèle. Les performances obtenues sont proches de l'interactivité. Mais le parallélisme offre aussi la possibilité (souvent sous-estimée) de traiter des images plus grandes, ce qui est essentiel dans le cadre de notre chaîne de traitement des images satellites. En effet, les algorithmes distribués présentés ici vont notamment être utilisés pour redresser les images SPOT en géométrie épipolaire.

CHAPITRE 6

Conclusion et Perspectives

6.1 Conclusion

Notre travail, tout au long de cette thèse, a donc abouti à une chaîne complète d'algorithmes parallèles pour le traitement des images satellites SPOT, dans le but de la reconstruction et de la visualisation de terrains. Nous avons présenté trois algorithmes s'insérant chacun dans un domaine particulier de l'imagerie : la vision, la synthèse et le traitement des images.

Finalement, ce travail de thèse a permis de mettre en évidence plusieurs caractéristiques importantes du parallélisme. Les résultats que nous avons obtenus avec nos différents algorithmes tendent à confirmer que cette voie est très prometteuse et même nécessaire pour certains problèmes de grande taille demandant des calculs intensifs. De plus, le parallélisme ne se contente pas de rester une simple voie de recherche uniquement utilisable en laboratoire mais commence bel et bien à s'étendre au grand public au travers des premières machines de type PC équipées de 2 ou 4 processeurs.

Néanmoins, tout au long de nos travaux, nous avons pu constater que ce type de programmation pose aussi des problèmes importants. En effet, la programmation parallèle n'est pas quelque chose de naturel pour nous qui sommes habitués à agir de manière séquentielle. De plus, le fait de faire coopérer plusieurs processeurs requiert un travail supplémentaire de coordination et aussi de communication des données. Ces surcoûts spécifiques au parallélisme peuvent réduire considérablement son efficacité lorsqu'ils sont mal gérés. C'est pourquoi il faut redoubler d'efforts dans les domaines particuliers de l'équilibrage des charges de travail, mais aussi dans la mise au point de structures de données permettant d'obtenir des communications réduites. Nous avons montré dans nos différents travaux que l'on pouvait tirer parti des spécificités du problème traité pour obtenir de telles structures. De même, la méthode d'équilibrage basée sur une répartition judicieuse des données s'est montrée efficace.

Notre étude a aussi été l'occasion de montrer une certaine dualité entre le parallélisme de tâche et le parallélisme de données. En effet, nous nous sommes aperçus que les algorithmes écrits selon un modèle de parallélisme à gros grain ou grain moyen (SPMD) s'adaptent très mal à un contexte données/processeurs à grain fin (SIMD). On a pu constater une chute des performances lorsque le nombre de processeurs devenait trop important et impliquait un nombre réduit de données à traiter par processeur. Il y a donc un équilibre fragile à trouver entre la quantité totale de données

et le nombre de processeurs à utiliser pour obtenir des performances optimales. De même, les communications entre processeurs se sont révélées être primordiales. En particulier, il est clairement apparu que l'on ne peut pas faire une abstraction totale du réseau de communication de la machine cible.

Dans le domaine particulier de l'imagerie numérique (vision, synthèse et traitement) et du traitement des MNT, nous avons montré que la régularité des structures des images et des terrains jouent un rôle important et positif sur les échanges de données. La similitude des données entre les trois problèmes étudiés nous a permis d'utiliser la même stratégie de parallélisation. De même, nous avons mis en évidence certains paramètres importants parmi lesquels on peut citer notamment, le rapport entre la taille de données lues (entrées) et générées (sorties), ou encore le nombre de données à échanger qui conditionne le volume des communications.

Pour mettre au point un algorithme parallèle efficace, il faut donc prendre en compte ces éléments et faire certains compromis entre le temps et l'espace mémoire. Ainsi, dans le domaine de l'imagerie parallèle, on va privilégier la distribution des données plutôt que leur duplication, diviser l'espace des données générées plutôt que celui des données en entrée, lorsque cela est possible, et enfin, utiliser au mieux la régularité des structures de données pour coder les messages lors des communications entre processeurs.

6.2 Perspectives

Bien que les algorithmes que nous avons mis au point présentent une efficacité relativement bonne, certaines améliorations sont envisageables.

6.2.1 Reconstruction des terrains

Concernant cet algorithme, on peut effectuer des modifications à plusieurs niveaux.

Si l'on reprend la première corrélation sur des images réduites, il serait intéressant d'augmenter le nombre d'images intermédiaires à des résolutions différentes. On pourrait donc construire un pyramide d'images avec un rapport 2 entre chaque étage. Cela permettrait de réduire les erreurs toujours possibles lors de la phase d'interpolation. La rapidité de la corrélation nous permet d'envisager cette modification sans rendre l'algorithme final trop long.

Si l'on utilise la pyramide précédente, on peut alors modifier notre fonction pour effectuer l'interpolation aussi bien en vertical qu'en horizontal. Notre problème avec la direction verticale était la taille indéterminée des tampons à cause des voisins non valides. Mais, on peut toutefois modifier le traitement d'un point valide dont les deux voisins directs ne sont pas valides pour contraindre le tampon à trois lignes de disparités.

Toujours dans le but d'améliorer la qualité et la robustesse de notre mise en correspondance, une idée intéressante serait de prendre en compte la cohérence entre les lignes consécutives de l'image. Nous avons déjà utilisé la cohérence au sein d'une ligne mais pas entre les lignes. Néanmoins,

cela modifierait considérablement la gestion mémoire de notre chaîne de traitement. De plus, des vérifications supplémentaires devraient être faites pour ne pas propager des erreurs d'une ligne à l'autre. Il faut donc être très prudent avec ce type de modifications.

Un autre point important concerne le type des images traitées. À l'heure actuelle, nous traitons des images provenant du satellite SPOT dont la géométrie du capteur est très particulière. Il serait très intéressant de pouvoir aussi traiter des images aériennes prises d'avion avec des caméras plus classiques à géométrie conique. Les modifications à apporter se situent aux niveaux du redressement en géométrie épipolaire et de la reconstruction tridimensionnelle des points. Bien que le premier point n'ait pas été traité directement dans nos travaux, notre algorithme de transformation d'images a été conçu dans ce but. Pour le deuxième point, il suffit de modifier la reconstruction en reprenant le modèle de déformation appliqué sur les images lors du redressement en géométrie épipolaire. Cette modification est très importante car elle permettrait d'étendre notre algorithme à un champ d'application encore plus vaste.

Enfin, le dernier point d'étude complémentaire se situe au niveau de notre modèle théorique d'exécution de l'algorithme parallèle. En effet, nous avons donné une première esquisse se basant sur les complexités de chaque étape mais nous avons pu constater que cela ne nous donne pas une estimation très précise. Il serait donc intéressant d'affiner ce modèle pour mieux comprendre le comportement de notre algorithme lorsque les paramètres changent (données initiales, nombre de processeurs, ...).

6.2.2 Visualisation des terrains

Concernant la visualisation des terrains, les modifications fondamentales à apporter ont déjà été discutées à la fin du chapitre 4. Il s'agit de l'utilisation de la multirésolution, de la cohérence spatiale entre les images d'une animation et de la méthode de division de l'image. De plus, certaines optimisations, notamment lors de la construction des messages et lors du dessin des polygones, peuvent être envisagées pour réduire les temps de calcul.

6.2.3 Transformation géométrique d'image

Actuellement, on ne voit pas de modifications importantes à apporter aux variantes distribuées de l'algorithme général. De plus, cet algorithme a été implanté en prenant en compte les optimisations évoquées précédemment pour la construction des messages et le dessin des polygones. Néanmoins, comme on a pu le voir, il pourrait être intéressant de combiner les deux types de calcul, en une ou deux passes, pour toujours obtenir les meilleures performances en fonction de la transformation voulue.

6.2.4 Méthodologie générale

Enfin, si l'on se replace dans le contexte global de la conception d'algorithmes parallèles, nos résultats et conclusions peuvent ouvrir la voie à de nouvelles recherches. On a pu constater que l'utilisation d'un seul modèle de parallélisme limitait le champ d'utilisation de l'algorithme. Il apparaît donc particulièrement intéressant d'utiliser deux algorithmes réalisant le même travail mais conçus selon les deux modèles de parallélisme: de données ou de tâches. En effet, on peut penser qu'une telle stratégie permettrait d'obtenir de bonnes performances quelle que soit la taille des données à traiter et le nombre de processeurs utilisés. Elle requiert cependant une double programmation. L'idéal serait donc de mettre au point un système de programmation qui permette de mettre en œuvre ces deux modèles de parallélisme.

Liste des publications personnelles

La liste des publications faite ici ne tient pas compte de la chronologie des travaux effectués au cours de ma thèse mais de la classification par média.

Dépôts de Logiciels

[CVS96a] Contassot-Vivier Sylvain. Volter. *Agence pour la Protection des Programmes.* No : IIDN.FR.001.460013.00.S.P.1996.000.21000. 1996.

Ce logiciel de visualisation de terrains texturés est le résultat d'une étude menée lors de mon stage de licence à l'École Normale Supérieure de Lyon. Il intègre une interface graphique développée sous les environnements X11 et OpenWindows et un module de génération d'animation sous forme d'une commande Unix. Il a été réalisé en collaboration avec le département des Sciences de la Terre de l'École.

Revues d'audience internationale

[CVS97] Contassot-Vivier Sylvain. A Load Balanced Parallel Ground Visualization Tool. *International Journal of Pattern Recognition and Artifical Intelligence*, 11(7):1113-1127, 1997.

Cette publication reprend et étend les résultats présentés dans [CVM95] et [CVS96]. Elle est parue dans un numéro spécial du journal IJPRAI, consacré à l'imagerie parallèle, à la suite d'une issue spéciale de la conférence IWPIA'95.

Congrès avec comité de lecture d'audience internationale, dont les actes sont publiés

[CVM95] Contassot-Vivier Sylvain et Serge Miguet. Parallel Visualization of Texture-Mapped Digital Elevation Models. Dans *Fourth International Workshop on Parallel Image Analysis (IWPIA '95)*, pages 269–282. Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France, Décembre 1995.

Cette communication est une introduction à l'étude de l'algorithme parallèle de visualisation de terrains texturés avec équilibrage de la charge de travail. Elle a permit de fixer le cadre de l'étude et de montrer les premiers résultats expérimentaux obtenus sur machine Volvox.

[CVS96b] Contassot-Vivier Sylvain. Calculs Parallèles pour la Visualisation de Terrains avec Textures. Dans *Huitièmes Rencontres Francophone du Parallélisme (RenPar'8), Edition Spéciale, GDR-PRC Parallélisme, Réseaux et Systèmes*, pages 93–96, Bordeaux, France, Mai 1996.

Dans cette publication, nous montrons nos résultats sur l'étude de l'algorithme parallèle de visualisation de terrains texturés. Cette étude est étayée par deux séries de tests sur machine Volvox et Cray T3D.

[CVM97] Contassot-Vivier Sylvain et Serge Miguet. Parallel Geometric Transformations of Images. Dans *Fifth International Workshop on Parallel Image Analysis (IWPIA'97)*, pages 13–30. Université d'Hiroshima , Japon, Septembre 1997.

Dans cet article, nous proposons un algorithme parallèle et générique pour réaliser n'importe quelle transformation géométrique d'images. La transformation est décrite sous forme d'équations paramétriques sur les coordonnées des pixels de l'image. Nous nous focalisons sur l'équilibrage des charges, la gestion optimale de la mémoire et les structures de données utilisées pour obtenir des communications minimales. Les tests expérimentaux ont été réalisés sur machine Cray T3E.

[CVM98] Contassot-Vivier Sylvain et Serge Miguet. Optimization and Parallelization of a Geographical Stereo Vision Code. Dans *The Sixth International Conference in Central Europe on Computer Graphics and Visualization'98 (WSCG'98)*. Université de Bohème de l'Ouest, Campus Bory, Pilsen, République Tchèque, Février 1998.

Cette article présente le travail central de ma thèse. Nous y décrivons toutes les étapes de la conception de notre algorithme parallèle de reconstruction tridimensionnelle de terrains à partir d'un algorithme séquentiel préexistant. Nous présentons aussi la phase d'optimisation de l'algorithme initial, portant sur les temps de calculs, la gestion mémoire et la qualité des résultats fournis. L'étude est complétée par des expérimentations sur machine Cray T3E.

Communication à un groupe de travail

[CVBN96] Sylvain Contassot-Vivier, Giosué Lo Bosco, Chanh Dao Nguyen. Multiresolution Approach for Image Processing. Dans *Intensive Program on Image Analysis Topics*, Erasmus ICP-A-2007. Leiden, Pays Bas, Avril 1996.

Cet article est le résultat d'une étude bibliographique portant sur l'utilisation de la multirésolution pour le traitement d'images. Il a été réalisé dans le cadre du projet ERASMUS de la Communauté Européenne, réunissant dans des groupes de travail des chercheurs et étudiants de plusieurs pays de la Communauté.

Rapports de recherche LIP

[RR96] Contassot-Vivier Sylvain. Parallel Visualization of Textured-Mapped Digital Elevation Models. Rapport de Recherche 96-02, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, 1996.

Bibliographie

[All91] M. Allison. *Private Communication*, July 1991.

[BB91] H. Harlyn Baker and Thomas O. Binford. Depth from edge and intensity based stereo. In Patrick J. Hayes, editor, *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, pages 631–636, Los Altos, CA, 24–28 August 1991. William Kaufmann.

[BB93] L. Anne Breene and Jack Bryant. Image warping by scanline operations. *Computers and Graphics*, 17(2):127–130, March–April 1993.

[BF82] Stephen T. Barnard and Martin A. Fischler. Computational stereo. *ACM Computing Surveys*, 14(4):553–572, December 1982.

[BL84] M. Berthod and P. Long. Graph matching by parallel optimization methods: an application to stereo vision. In *Seventh International Conference on Pattern Recognition (Montreal, Canada, July 30-August 2, 1984)*, IEEE Publ. 84CH2046-1, pages 841–843. IEEE, IEEE, 1984.

[Bro92] Lisa Gottesfeld Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, December 1992.

[BS72] D. I. Barnea and H. F. Silverman. A class of algorithms for fast digital image registration. *IEEE Transactions on Computer*, 21:179–186, 1972.

[BT80] S. T. Barnard and W. B. Thompson. Disparity analysis of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-2(4):333–340, july 1980.

[CBA93] L. Cohen, E. Bardinet, and N. Ayache. Surface reconstruction using active contour models. Technical Report 1824, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1993.

[CBF+92] Henri-Pierre Charles, Olivier Baby, Anne Fouilloux, Serge Miguet, Laurent Perroton, Yves Robert, and Stéphane Ubéda. Ppcm : A portable parallel communication module. Technical Report 92-04, LIP-IMAG, ENS-Lyon, 46 allée d'Italie, 69364 Lyon CEDEX 07, 1992.

[CDH+88] F. C. Crow, G. Demos, J. Hardy, J. McLaughlin, and K. Sims. 3D image synthesis on the connection machine. *Proceedings of the International Conference on Parallel Processing for Computer Vision and Display*, January 1988.

[Chu96] Kuo-Liang Chung. Image template matching on reconfigurable meshes. *Parallel Processing Letters*, 6(3):345–353, September 1996.

[CJ81] P. Chang and R. Jain. A multi-processor system for hidden surface removal. *Computer Graphics*, 15(4):405–436, December 1981.

[CK91] Y. Choe and R. Kashyap. 3-d shape from a shaded and textural surface image. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13:907–919, 1991.

[CLM92] Henri-Pierre Charles, Jian-Jin Li, and Serge Miguet. A portable parallel toolkit for 3D image pro cessing. *The European Transactions on Telecommunications*, 3(6):31–42, 1992.

[CLM95] Henri-Pierre Charles, Laurent Lefèvre, and Serge Miguet. An optimized and load-balanced portable parallel ZBuffer. In *SPIE Symposium on Electronic Imaging: Science and Technology*, 1995.

[CR74] E. Catmull and R. Rom. A class of local interpolating splines. *Computer Aided Geometric Design*, pages 317–326, 1974.

[CS80] E. Catmull and A. R. Smith. 3-D transformations of images in scanline order. *Computer Graphics*, 14(3):279–285, July 1980.

[CSWW97] B. Cahoon, S. Singhai, G. Weaver, and E. Wright. A parallel implementation of a correspondence-finder for uncalibrated stereo image pairs. Technical Report UM-CS-1997-013, University of Massachusetts, Amherst, Computer Science, October, 1997.

[CV93] Sylvain Contassot-Vivier. Reconstruction 3D de terrains texturés. Rapport de stage de licence, August 1993.

[CVSG89] L. Cohen, L. Vinet, P. T. Sander, and A. Gagalowicz. Hierarchical region based stereo matching. In *CVPR'89 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Diego, CA, June 4–8, 1989)*, pages 416–421, Washington, DC., June 1989. Computer Society Press.

[DA89] U. R. Dhond and J. K. Aggarwal. Structure from stereo – A review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, November-December 1989.

[DCM92] T. Day, A. C. Cook, and J. P. Muller. Automated digital topographic mapping techniques for mars. In *XVII Int.Society of Photogrammetry and Remote Sensing (ISPRS 92) Congress, Comission II, Washington DC*, august 1992.

[DD95] Jack J. Dongarra and Tom Dunigan. Message-passing performance of various computers. Technical report, University of Tennessee and Oak Ridge National Laboratory, August 1995.

[DM89] T. Day and J. P. Muller. Digital elevation model production by stereo-matching SPOT image pairs: a comparison of algorithms. *Image and Vision Computing*, 7:95–101, 1989.

[Ell94] D. A. Ellsworth. A new algorithm for interactive graphics on multicomputers. *IEEE COMPUTER*, July 1994.

[FFR83] E. Fiume, A. Fournier, and L. Rudolph. A parallel scan conversion algorithm with anti-aliasing for a general-purpose ultracomputer. *ACM Computer Graphics*, 17(3):141–149, July 1983.

[FHM+93] O. Faugeras, B. Hotz, H. Mathieu, T. Viéville, Z. Zhang, P. Fua, E. Théron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real time correlation-based stereo: algorithm, implementatinos and applications. Technical Report 2013, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1993.

[FMP95] Fabien Feschet, Serge Miguet, and Laurent Perroton. *Parallélisme et Applications Irrégulières*, chapter 7 : ParList : une structure de donnée parallèle pour l'équilibrage des charges, pages 177–201. HERMES, 1995.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990.

[GGB84] A. Goshtasby, S. H. Gage, and J. F. Bartholic. A two-stage cross correlation approach to template matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:374–378, 1984.

[Gib50] J. J. Gibson. *The Perception of the Visual World.* Houghton Mifflin, Boston, Massachusetts, 1950.

[GL94] G. Giraudon and JL. Lotti. Adaptive window algorithm for aerial image stereo. In Jean luc Lotti, editor, *International Conference on Computer Vision and Pattern Recognition*, pages 701–703, Jerusalem, October 1994.

[Gra90] G. H. Granlund. Hierarchical computer vision. In L. Torres, E. Masgrau, and M. A. Lagunas, editors, *Signal Processing V: Theories and Applications*, pages 73–84. Elsevier Science, 1990.

[Gru85] A. W. Gruen. Adaptive least squares correlation : a powerful images matching algorithm. *S. Afr. J. of Photogrammetry Remote Sensing and Carthography*, 14(3):175–187, 1985.

[GW93] George Gusciora and Jon A. Webb. Parallel affine image warping. In L. N. Kanal, editor, *Parallel Processing for Artificial Intelligence.* North-Holland, 1993.

[HA89] W. Hoff and N. Ahuja. Surfaces from stereo: Integrating feature matching, disparity estimation, and contour detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-11(2):121–136, February 1989.

[Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Technical report, CS Division, EECS Dept, UC Berkeley, May 1989.

[Hen97] R. D. Henkel. Stereovision by coherence detection. Technical report, Institute for Theoretical Neurophysics, University of Bremen, 1997.

[HMP92] Y. C. Hsieh, D. M. McKeown, and F. P. Perlant. Performance evaluation of scene registration and stereo matching for cartographic feature extraction. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):214–238, February 1992.

[HOE94] Yacov Hel-Or and Shimon Edelman. A new approach to qualitative stereo. In *Proc. ICPR*, pages 316–320, 1994.

[Hor75] B. K. P. Horn. Obtaining shape from shading information. *The Psychology of Computer Vision*, 1975.

[HZM91] M. Holden, M. J. Zemerly, and J-P. Muller. Using a transputer array for parallel stereo matching of SPOT satellite images. In T. S. Durrani, W. A. Sandham, J. J. Soraghan, and J. Hulskamp, editors, *Applications of Transputer 3*, pages 154–159, Amsterdam, 1991. IOS Press.

[IB94] Stephen S. Intille and Aaron F. Bobick. Disparity-space images and large occlusion stereo. In Jan-Olof Eklundh, editor, *European Conference on Computer Vision*, pages 179–186, Vol B, Stockholm, Sweden, May 1994. Springer-Verlag.

[Jac93] F. Jacquis. Amélioration d'une méthode de corrélation en vue de construire un modèle numérique de terrain. Master's thesis, Ecole Normale Supérieure de Lyon, June 1993.

[JKMC94] F. Jacquis, T. Kalinke, P. Martin, and J.M. Chassery. Approches pour la mise en correspondance d'images satellite. In *RFIA'94 - Paris - France*, 1994.

[JM91] David G. Jones and Jitendra Malik. A computational framework for determining stereo correspondence from a set of linear spatial filters. Technical Report CSD-91-655, University of California Berkeley, CS, October 91.

[KB91] B. Kim and P. Burger. Depth and shape from shading using the photometric stereo method. *Computer Vision, Graphics, and Image Processing*, 54(3):416–427, November 1991.

[KD94] K. Sunil Kumar and U. B. Desai. New algorithms for 3D depth estimation from binocular stereo. *Journal of the Franklin Institute*, 331B(5):531–554, 1994.

[KDKA92] K. Kakusho, S. Dan, T. Kitahashi, and N. Abe. Computer vision based on hypothesization and verification scheme by parallel relaxation. *International Journal of Computer Vision*, 9(1):13–30, 1992.

[Ken80] J. R. Kender. Shape from texture. Technical Report CMU-CS-81-102, Carnegie-Mellon University, 1980.

[KG79] M. Kaplan and D. P. Greenberg. Parallel processing techniques for hidden surface removal. *Computer Graphics, Proceedings for Siggraph*, 13(2):300–307, July 1979.

[KS96] Sing Bing Kang and Richard Szeliski. 3-D scene recovery using omnidirectional multibaseline stereo. In *International Conference on Computer Vision and Pattern Recognition*, pages 364–370, San Francisco, CA, June 1996.

[LB95] An Luo and Hans Burkhardt. An intensity-based direct method for depth estimation from image sequences with kalman filtering. Technical Report TR-402-95-021, Technical University of Hamburg-Harburg, June 1995.

[Lef93] L. Lefèvre. Communications intensives sur hypercube pour l'algorithme du zbuffer. Technical report, Laboratoire de l'Informatique du Parallélisme (LIP), ENS Lyon, 69007, France, 1993.

[LH96] Z. N. Li and G. Hu. Analysis of disparity gradient based cooperative stereo. *IEEE Transactions on Image Processing*, 5:1493–1506, November 1996.

[LHB87] S. A. Lloyd, E. R. Haddow, and J. F. Boyce. A parallel binocular stereo algorithm utilizing dynamic programming and relaxation labelling. *Computer Vision, Graphics, and Image Processing*, 39(2):202–225, August 1987.

[Li96] Zi-Cai Li. Analysis of discrete techniques for image transformation. *Numerical Algorithms*, 13(3-4):225–263, 1996.

[LM95] Z. D. Lan and R. Mohr. Robust matching by partial correlation. Technical Report 2643, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1995.

[LR91] A. F. Laine and G.-C. Roman. A parallel algorithm for incremental stereo matching on SIMD machines. *IEEE Transactions on Robotics and Automation*, 7(1):123–134, February 1991.

[Mat92] L. Matthies. Stereo vision for planetary rovers: Stochasting modeling to near real-time implementation. *International Journal of Computer Vision*, 8(1):71–91, 1992.

[MCEF94] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications*, July 1994.

[Mem91] M. Memier. *Stéréophotogrammétrie numérique : calcul de MNT par corrélation automatique d'images SPOT.* PhD thesis, Université Joseph Fourier Grenoble I, 1991.

[MP76] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194(4262):283–287, October 1976.

[MP79] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proc. Royal Soc. London Bulletin*, B204:301–328, 1979.

[MR91] Serge Miguet and Yves Robert. Elastic load balancing for image processing algorithms. In H. P. Zima, editor, *Parallel Computation*, Lecture Notes in Computer Science, pages 438–451, Salzburg, Austria, September 1991. 1st International ACPC Conference, Springer Verlag.

[Nas92] N. M. Nasrabadi. A stereo vision technique using curve-segments and relaxation matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(5):566–572, May 1992.

[OC89] G. P. Otto and T. K. W. Chau. "region-growing" algorithm for matching of terrain images. *Image and Vision Computing*, 7:83–94, 1989.

[OK85a] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7:139–154, 1985.

[OK85b] Y. Ohta and T. Kanade. Stereo by two-level dynamic programming. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (Los Angeles, CA, August 18–23, 1985)*, pages 1120–1126, 1985.

[OK93] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–63, April 1993.

[Ols90] Soren I. Olsen. Stereo correspondence by surface reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(3):309–315, March 1990.

[OTI87] Y. Ohta, K. Takano, and K. Ikeda. A highspeed stereo matching system based on dynamic programming. In *First International Conference on Computer Vision, (London, England, June 8–11, 1987)*, pages 335–342, Washington, DC., 1987. IEEE Computer Society Press.

[PD96] D. V. Papadimitriou and T. J. Dennis. Epipolar line estimation and rectification for stereo image pairs. *IEEE Transactions on Image Processing*, 5(4):672–676, April 1996.

[Pie96] J. M. Pierson. *Équilibrage de Charge Dirigé par les Données : Applications à la Synthèse d'Images*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, 1996.

[PJ95] S. Pankanti and A. K. Jain. Integrating vision modules : Stereo, shading, grouping, and line labeling. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(8):831–842, September 1995.

[PMF85] S. Pollard, J. Mayhew, and J. Frisby. PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14, 1985.

[RA90] J. J. Rodriguez and J. K. Aggarwal. Stochastic analysis of stereo quantization error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-12(5):467–470, May 1990.

[Rob88] D. R. Roble. A load balanced parallel scanline Z-Buffer algorithm for the iPSC hypercube. *Proceedings of Pixim'88, Paris, France*, October 1988.

[She92] F. Shevlin. Geometric transformation of images. Technical Report TCD-CS-92-33, Department of Computer Science, University of Dublin, Trinity College, 30 September 1992.

[Sil94] G. Silber. Un z-buffer parallèle : Coupage et optimisation. Technical report, Laboratoire de l'Informatique du Parallélisme (LIP), ENS Lyon, 69007, France, 1994.

[SL97] A. J. Stewart and M. S. Langer. Toward accurate recovery of shape from shading under diffuse lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1020–1025, September 1997.

[SN96] P. W. Smith and N. Nandhakumar. An improved power cepstrum based stereo correspondence method for textured scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(3):338–348, March 1996.

[SP90] D. Sherman and S. Peleg. Stereo by incremental matching of contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(11):1102–1106, November 1990.

[Td91] Hemant D. Tagare and Rui J. P. deFigueiredo. A theory of photometric stereo for a class of diffuse non-lambertian surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-13(2):133–152, February 1991.

[TD92] H. D. Tagare and R. J. P. DeFigueiredo. Simultaneous estimation of shape and reflectance map from photometric stereo. *Computer Vision, Graphics, and Image Processing*, 55(3):275–286, May 1992.

[TKKW86] A. Tanaka, M. Kameyama, S. Kazama, and O. Watanabe. A rotation method for raster image using skew transformation. In *Proceedings, CVPR '86 (IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, June 22–26, 1986)*, IEEE Publ.86CH2290-5, pages 272–277. IEEE, 1986.

[VY93] O. E. Vega and Y. H. Yang. Shading logic : A heuristic approach to recover shape from shading. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 15(6):592–597, June 1993.

[WAH92] J. Wueng, N. Ahuja, and T. S. Huang. Matching two perspective views. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(8):806–825, August 1992.

[WH97] G. Q. W. Wei and G. Hirzinger. Parametric shape-from-shading by radial basis functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):353–365, April 1997.

[Whe85] D. S. Whelan. *Animac: A Multiprocessor Architecture for Real-Time Computer Animation.* PhD thesis, California Institute of Technology, 1985.

[Whi92] S. Whitman. *Multiprocessor Methods for Computer Graphics Rendering.* Jones and Bartlett, 1992.

[Whi94] S. Whitman. Dynamic load balancing for parallel polygon rendering. *IEEE Computer Graphics and Applications*, July 1994.

[WIB91] R. J. Woodham, Y. Iwahori, and R. A. Barman. Photometric stereo: Lambertian reflectance and light sources with unknown direction and strength. Technical Report TR-91-18, University of British Columbia Computer Science Department., August 1991.

[Wol90] George Wolberg. *Digital Image Warping.* IEEE Computer Society Press, 10662 Los Vaqueros Circle, Los Alamitos, CA, 1990. IEEE Computer Society Press Monograph.

[Woo80] R. Woodham. Photometric method for determining surface orientation from multiple images. *Opt. Eng.*, 19:139–144, 1980.

[WR94] Jon A. Webb and Bill Ross. Real-time parallel stereo vision. Sampler page, CMU Robotics Institute, August 1994.

[Yam85] M. Yamashita. Parallel and sequential transformations on digital images. *Pattern Recognition*, 18:31–41, 1985.

[YPZC95] J. You, E. Pissaloux, W. P. Zhu, and H. A. Cohen. Efficient image matching: a hierarchical Chamfer matching scheme via distributed system. *Real-Time Imaging*, 1(4):245–259, October 1995.

[YS89] Y. Yeshurun and E. L. Schwartz. Cepstral filtering on a columnar image architecture: A fast algorithm for binocular stereo segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):759–767, July 1989.

[ZDFL95] Zhengyou Zhang, Rachid Deriche, Olivier Faugeras, and Quang-Tuan Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry,. *Artificial Intelligence, December 1995*, 78:87–119, 1995.

[Zha93] Z. Zhang. Le problème de la mise en correspondance : L'état de l'art. Technical Report 2146, Institut National De Recherche en Informatique et en Automatique (INRIA), 06902 Sophia Antipolis, France, 1993.

[Zha96] Yuefeng Zhang. A fuzzy approach to digital image warping. *IEEE Computer Graphics and Applications*, 16(4):34–41, July 1996.