CEA-R-6047

ISSN 0429 - 3460

COMMISSARIAT À L'ÉNERGIE ATOMIQUE

ETAT DE L'ART DES APPLICATIONS DE VISUALISATION SCIENTIFIQUE PARALLÈLES SUR GRAPPES DE PC

par

Manuel JULIACHS

CEA/DAM - DIRECTION ILE-DE-FRANCE

DÉPARTEMENT SCIENCES DE LA SIMULATION ET DE L'INFORMATION

SERVICE NUMÉRIQUE ENVIRONNEMENT ET CONSTANTES

DIRECTION DES SYSTEMES D'INFORMATION

CEA / SACLAY 91191 GIF-SUR-YVETTE CEDEX FRANCE

CEA/DAM ILE-DE-FRANCE

2004

RAPPORT

RAPPORT CEA-R-6047

- Rapport CEA-R-6047 -

CEA/DAM – Direction Ile-de-France
Département Sciences de la Simulation et de l'Information
Service Numérique Environnement et Constantes

ÉTAT DE L'ART DES APPLICATIONS DE VISUALISATION SCIENTIFIQUE PARALLÈLES SUR GRAPPES DE PC

par

Manuel JULIACHS

- Mars 2004 -

RAPPORT CEA-R-6047 – Manuel JULIACHS

«Etat de l'art des applications de visualisation scientifique parallèles sur grappes de PC»

Résumé **-** Ce rapport présente un état de l'art des applications de visualisation scientifique parallèles sur grappes de PC, classées par la méthode de rendu graphique : surfacique ou volumique. Au préalable, les configurations de grappes de PC pour le graphique sont examinées, ainsi que les logiciels parallèles pour le rendu graphique parallèle sur grappe.
Cette étude fait suite à des expériences menées en 2001 au CEA/DIF et accompagne la mise en service d'une nouvelle plate-forme d'étude. Cette plate-forme, constituée d'une grappe de huit PC sous Linux pilotant un mur d'images, est réalisée en collaboration avec l'Université de Versailles-Saint-Quentin et fonctionne depuis août 2003.

2004 – Commissariat à l'Énergie Atomique – France

RAPPORT CEA-R-6047 – Manuel JULIACHS

«State of the art of parallel scientific visualization applications on PC clusters»

Abstract **-** In this state of the art on parallel scientific visualization applications on PC clusters, we deal with both surface and volume rendering approaches. We first analyze available PC cluster configurations and existing parallel rendering software components for parallel graphics rendering.
CEA/DIF has been studying cluster visualization since 2001. This report is part of a study to set up a new visualization research platform. This platform consisting of an eight-node PC cluster under Linux and a tiled display was installed in collaboration with Versailles-Saint-Qentin University in August 2003.

2004 – Commissariat à l'Énergie Atomique – France

Etat de l’art des applications de visualisation scientifique parallèles sur grappe de PC

Manuel Juliachs

Etat de l'art des applications de visualisation scientifique parallèles sur grappe de PC

State of the art of parallel scientific visualization applications on PC clusters

Résumé
Ce rapport présente un état de l'art des applications de visualisation scientifique parallèles sur grappes de PC, classées par la méthode de rendu graphique : surfacique ou volumique. Au préalable, les configurations de grappes de PC pour le graphique sont examinées, ainsi que les logiciels parallèles pour le rendu graphique parallèle sur grappe.
Cette étude fait suite à des expériences menées en 2001 au CEA/DIF et accompagne la mise en service d'une nouvelle plate-forme d'étude. Cette plate-forme, constituée d'une grappe de huit PC sous Linux pilotant un mur d'images, est réalisée en collaboration avec l'Université de Versailles-Saint-Quentin et fonctionne depuis août 2003.

Abstract
In this state of the art on parallel scientific visualization applications on PC clusters, we deal with both surface and volume rendering approaches. We first analyze available PC cluster configurations and existing parallel rendering software components for parallel graphics rendering.
CEA/DIF has been studying cluster visualization since 2001. This report is part of a study to set up a new visualization research platform. This platform consisting of an eight-node PC cluster under Linux and a tiled display was installed in collaboration with Versailles-Saint-Quentin University in August 2003.

Mots clés
Grappe de PC, cluster, visualisation scientifique, rendu graphique parallèle, mur d'images

Keywords
PC cluster, scientific visualization, parallel graphics rendering, tiled displays

Cette étude est suivie par Guillaume Colin de Verdière et Jean-Philippe Nominé du service DSSI/SNEC du CEA/DIF. Ce rapport a été rédigé par Manuel Juliachs, du laboratoire PRISM de l'Université de Versailles-Saint-Quentin, dans le cadre du contrat d'études CEA 4600057753.

Sommaire

Liste des figures

Liste des tableaux

1. Introduction

La simulation numérique de phénomènes physiques instationnaires produit des jeux de données de plus en plus précis et volumineux. Le calculateur Tera du CEA/DIF permet la production en flux de tels jeux grâce à sa puissance de calcul de l'ordre de 5 Tflops.

L'outil de la visualisation scientifique fournit une aide cruciale à l'exploitation efficace de tels jeux de données, qui sont fournis sous des formats très variables (bi ou tri-dimensionnels). Pour cela, le service NEC du CEA/DIF a mis en place en 2001 une solution de visualisation collaborative « haute performance » basée sur un mur d'images à haute résolution (8 millions de pixels) relié à un calculateur graphique haut de gamme. De plus, une étude a été engagée sur les architectures de visualisation à base de grappes de PC ([COS02]). Le but de cette étude préliminaire était d'évaluer les performances des architectures de rendu à faible coût. Il a ensuite été décidé de poursuivre cette étude afin d'évaluer l'utilisation des grappes de PC pour les besoins en visualisation de jeux de données dynamiques de très grande taille du CEA/DIF. Pour cela, le CEA/DIF travaille en collaboration avec l'Université de Versailles-Saint-Quentin (UVSQ) sur le couplage d'une grappe de visualisation à un mur d'images, le tout installé au CEA/DIF (3 millions de pixels et 2 m par 1,5 m).

Ce rapport fait partie d'une étude préliminaire à la prise en main de ce système. Cette étude se compose de deux parties : une synthèse bibliographique sur la visualisation scientifique sur grappes de PC et une étude technique des composants logiciels de visualisation sur grappes de PC. Ce document est le résultat de la première partie de l'étude, synthèse bibliographique ayant valeur d'état de l'art sur le domaine de la visualisation «haute performance » sur grappes de PC.

Ce rapport comporte quatre parties.
Premièrement, nous décrivons la problématique du parallélisme du rendu dans le processus de visualisation scientifique. Nous présentons alors une classification des différentes approches du rendu parallèle et décrivons comment celles-ci peuvent être implantées dans des architectures initialement spécialisées, et maintenant dans des architectures de type grappes de PC.
Deuxièmement, nous décrivons les aspects matériels des grappes de PC pour la visualisation. Nous présentons l'architecture générale d'une grappe de PC et des sous-systèmes d'un PC individuel, les systèmes d'affichage haute résolution utilisés avec les grappes, les systèmes d'interconnexion à haut débit permettant d'agréger les différents nœuds d'une grappe et les composants et architectures permettant d'unifier les images partielles des nœuds individuels en une image finale.
Troisièmement, nous présentons les composants logiciels permettant d'exploiter le parallélisme d'une grappe. Nous considérons les composants de parallélisation du rendu graphique (implantations parallèles de la bibliothèque graphique OpenGL et systèmes de rendu parallèle par duplication des instances d'application). Nous décrivons également différentes méthodes algorithmiques pouvant améliorer les performances des composants logiciels de rendu parallèle.
Dans la dernière partie, nous présentons différentes applications de la visualisation scientifique sur grappes de PC. Pour cela, nous classons les applications de visualisation selon la méthode de rendu qu'elles utilisent : rendu volumique ou rendu surfacique.

Quelques rappels fondamentaux sur les techniques et algorithmes du rendu graphique surfacique et volumique sont effectués au cours du texte ou en annexe. On a laissé dans le fil du texte ce qui a trait aux techniques plus avancées (paragraphe 5.2.1 sur le rendu volumique), tandis que le rendu surfacique, plus fondamental, est détaillé en annexe (Annexe A : Rappels sur les principes et algorithmes du rendu graphique complétant le paragraphe 2.2.1 Le pipeline graphique). Les lecteurs familiers de ces techniques peuvent omettre la lecture détaillée de 5.2.1 et de l'annexe de 2.2.1.

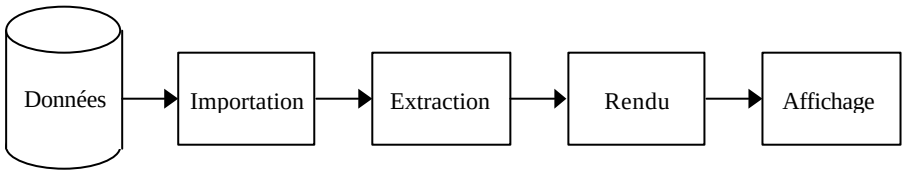
2. Visualisation scientifique et rendu graphique parallèle

Dans cette partie, nous présentons la problématique du rendu graphique parallèle en visualisation scientifique. Tout d'abord nous décrivons le processus de visualisation scientifique, de l'importation des données à l'affichage et nous focalisons sur l'étape de rendu, précédant l'affichage, qui est cruciale en visualisation scientifique interactive. Nous décrivons comment les performances du rendu peuvent être améliorées grâce au parallélisme. Nous présentons alors la classification par tri des méthodes de rendu parallèle, et une méthode de classification alternative. Puis, nous décrivons l'intégration du rendu parallèle dans un processus d'application plus générale. Nous décrivons alors les différents goulots d'étranglement grevant les performances. Ensuite, nous décrivons les architectures de rendu traditionnellement utilisées pour réduire les goulots d'étranglement du rendu graphique et introduisons les architectures de rendu distribué à base de grappes de PC.

2.1. Rappels sur le processus de visualisation scientifique

La visualisation scientifique haute performance requiert des ressources de calcul très importantes. En effet, les jeux de données scientifiques, que l'on cherche à représenter de manière graphique, ont une taille potentiellement illimitée. Ils peuvent provenir de nombreuses sources différentes : par exemple de simulations numériques ou bien de mesures physiques (imagerie médicale, mesures astrophysiques, etc.). En simulation numérique l'augmentation constante de la puissance des calculateurs permet de générer des données de plus en plus détaillées, ce qui est souhaitable pour pouvoir modéliser de plus en plus finement les phénomènes que l'on veut étudier. Afin de représenter graphiquement ces données de manière interactive, des ressources de rendu graphique de plus en plus grandes sont ainsi nécessaires. De plus, dans le cas de phénomènes dynamiques, les données sont représentées en fonction du temps, la visualisation d'un tel phénomène demande alors d'autant plus de ressources graphiques. La Figure 1 ci-dessous [NOM01] représente de manière simplifiée le processus de visualisation d'un jeu de données, obtenu au préalable et stocké sous forme non traitée pour la visualisation.

Figure 1 : processus de visualisation



- Importation : conversion des données du format natif vers un format exploitable.
- Extraction : extraction (isosurfaces, coupes, etc.) à partir des données converties de caractéristiques importantes sous forme de primitives.
- Rendu : calcul d'une image à partir des primitives extraites.
- Affichage : affichage de l'image rendue par un dispositif matériel (écran).

Chacune des étapes peut être effectuée sur des calculateurs différents ou bien sur une seule machine. Les premières étapes (importation et extraction) peuvent être effectuées de manière découplée du rendu. Ainsi, si par exemple la phase d'extraction de données est longue comparée à la phase de rendu, le découplage de ces deux phases permettra d'afficher les données extraites sans que la vitesse de rendu soit limitée par la vitesse d'extraction. A l'opposé, l'extraction et le rendu peuvent s'effectuer de manière « pipelinée » (on peut les enchaîner directement).
Une caractéristique majeure du rendu en visualisation scientifique est la résolution de l'image rendue. En effet, pour visualiser correctement des scènes correspondant à des jeux de données de très grande taille, il est nécessaire de rendre à haute résolution, sinon les détails de la scène ne seront pas correctement restitués. De plus, dans le cas des jeux de données exprimées en fonction du temps, il doit être possible de rendre l'évolution temporelle du phénomène qui a été simulé ou observé. Pour cela, nous considérons que l'on doit pouvoir rendre à une vitesse supérieure ou égale à 5 images/s la scène évoluant au cours du temps, chaque pas de temps donnant une image rendue distincte. Cela requiert ainsi des ressources de rendu d'autant plus grandes. En effet, le rendu de données statiques (par exemple un pas de temps unique d'une simulation numérique) requiert une vitesse de

rendu moins importante, puisque que l'on ne cherche pas à observer l'évolution d'un phénomène au cours du temps. Une vitesse de rendu moindre est acceptable, même si celle-ci doit être raisonnable en cas de changement de point de vue d'observation de la scène.
Pour augmenter les ressources de calculs de rendu, nous considérons deux possibilités majeures. Premièrement, on peut utiliser un calculateur mono-processeur plus puissant. Cependant, comme nous l'avons dit, la taille des jeux de données à visualiser est potentiellement illimitée, les performances d'un seul calculateur graphique ne peuvent égaler les besoins d'une scène de très grande taille. La seconde possibilité est l'utilisation de plusieurs processeurs de rendu travaillant en parallèle. Chaque processeur (ou unité de rendu) rend une partie de la scène totale, ce qui produit une image partielle. L'image rendue de la scène totale est obtenue à partir des images (partielles) des différentes unités de rendu. C'est le rendu parallèle. Les processeurs de rendu peuvent appartenir à la même machine (machine de type SMP, ou « Symmetric MultiProcessing »), ou bien être répartis sur des machines distinctes (un processeur de rendu par machine). Notons que dans ce dernier cas, chaque machine peut théoriquement avoir plus d'un processeur de rendu.
Nous décrivons dans la sous-partie suivante une classification des systèmes de rendu parallèle.

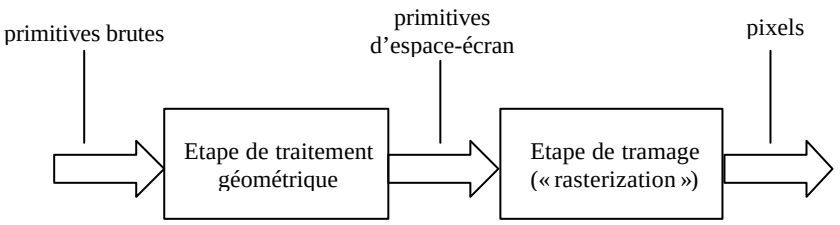
2.2. Classification des approches algorithmiques du rendu parallèle

Tout d'abord nous présentons le pipeline de rendu. Ensuite, nous présentons dans cette sous-partie deux classifications des systèmes et méthodes de rendu parallèle. Premièrement, la classification par la méthode de tri (« sort ») du pipeline de rendu, que nous utiliserons comme base de discussion pour le reste de ce document. Ensuite, nous décrivons une classification alternative, pouvant servir à décrire certains systèmes de visualisation parallèle. Puis, nous décrivons comment une application de visualisation doit elle-même être parallélisée pour pouvoir exploiter les capacités d'un système de rendu parallèle, et les goulots d'étranglement à surmonter pour cela.

2.2.1. Le pipeline graphique

Le pipeline graphique (simplifié) se compose de deux étapes : le traitement géométrique et la phase de tramage. La Figure 2 ci-dessous représente le pipeline graphique « feed-forward » : il prend en entrée des primitives géométriques (en général des triangles) et produit en sortie des pixels. L'étape de traitement géométrique projette les sommets des primitives de l'espace global (3D) vers l'espace écran (2D) et effectue des calculs d'éclairage par sommet. L'étape de tramage (« rasterization ») prend en entrée les primitives projetées, et les discrétise en un ensemble de pixels, suivant une grille régulière. La couleur de chaque pixel est déterminée en fonction d'un modèle d'interpolation de couleurs et/ou de placage de textures. Voir Annexe A : Rappels sur les principes et algorithmes du rendu graphique ainsi que [FOL90] pour plus de détails.

Figure 2 : pipeline de rendu



Dans ce modèle simplifié sur lequel nous nous baserons pour la suite de ce document, le débit du pipeline graphique (en nombre de pixels tramés par seconde), dépend de la vitesse de traitement des étapes de traitement géométrique et de tramage. Nous donnerons fréquemment les performances du pipeline en nombre de primitives brutes (généralement des triangles) traitées par seconde. Dans ce modèle simplifié, une primitive brute k subit en traitement géométrique alors qu'en parallèle, la primitive d'espace-écran k-1 (obtenue par transformation de la primitive brute k-1) est traitée par l'étape de tramage. Le débit du pipeline (en nombre de primitives brutes pouvant être traitées par seconde) est égal à l'inverse du maximum des temps de traitement de l'étape géométrique et de l'étape de tramage. Si les temps de traitements des 2 unités sont très différents, alors le pipeline est déséquilibré et son débit est égal au débit de l'unité qui a le plus grand temps de traitement. Les accélérateurs de rendu

implantent généralement la totalité du pipeline en matériel. La classification « sort », que nous présentons ci-dessous, utilise ce modèle de pipeline de rendu à deux étages.

2.2.2. Classification par le tri (« sort »)

2.2.2.1. Description

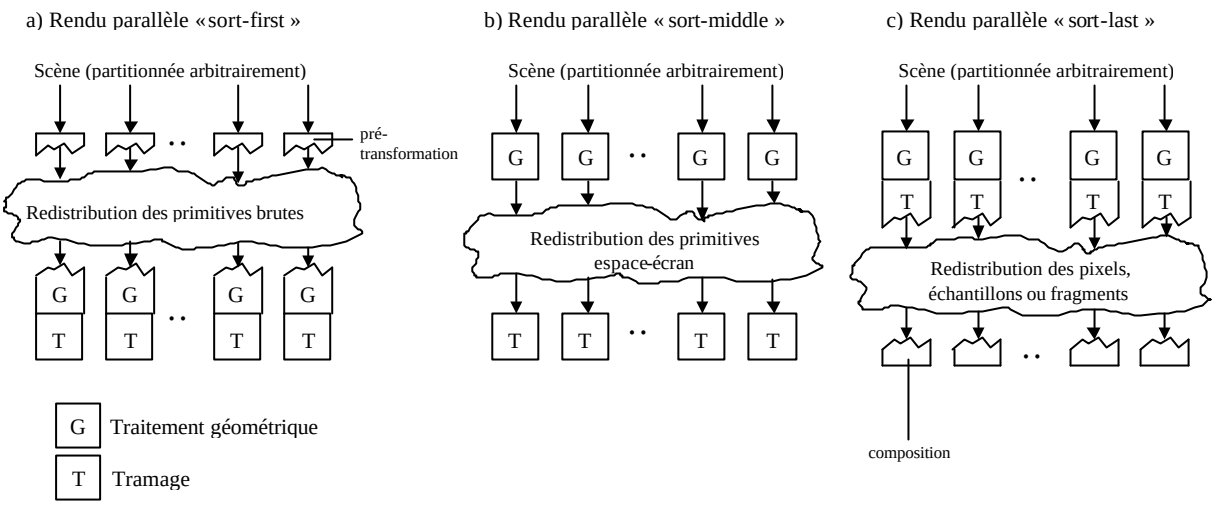
Molnar et al. [MOLN94] ont proposé une classification des systèmes de rendu parallèle suivant l'endroit du pipeline de traitement graphique qui est parallélisé. La classification « sort » considère l'endroit du pipeline où a lieu le tri (« sort ») des primitives (dites primitives brutes) à partir des coordonnées dans l'espace objet (3D) vers les coordonnées dans l'espace écran (2D). En effet, le problème du rendu équivaut à un problème de tri de primitives dans l'espace écran : une primitive pouvant avoir n'importe quelles coordonnées dans l'espace global, déterminer l'endroit de l'espace-écran où elle va se trouver revient à effectuer un tri (spatial) sur les primitives successives. Les systèmes de rendu totalement parallèles sont ceux qui parallélisent à la fois les étapes de traitement géométrique et de tramage. On considère alors trois grandes catégories de rendu parallèle : « sort-first », « sort-middle » et « sort-last ».

a) Méthode « sort-first » (SF) :
Dans le pipeline SF, la redistribution a lieu durant la phase géométrique. Les primitives brutes sont affectées arbitrairement aux unités de rendu (processeurs). Les primitives sont alors pré-transformées, puis, en fonction de leur position dans l'espace-écran, elles sont redistribuées, via un réseau d'interconnexion, aux unités de rendu qui effectuent le reste des calculs de rendu (transformations géométriques et tramage), chaque unité ayant en charge une portion de l'écran disjointe.

b) Méthode « sort-middle » (SM) :
Les primitives sont affectées arbitrairement à des processeurs géométriques et transformées. Elles sont ensuite redistribuées à des processeurs de tramage (comme dans SF, chacun gère une région distincte de l'écran) en fonction de leurs coordonnées écran (ce sont des primitives dites « d'affichage », ou d'espace-écran). Chaque processeur de tramage rend alors sa région d'écran.

c) Méthode « sort-last » (SL) :
Le tri a lieu en fin de pipeline. Les primitives sont affectées arbitrairement aux processeurs. Chacun calcule des valeurs de pixels (rendu) et les envoie à un réseau de processeurs de composition qui déterminent la visibilité des pixels et produisent l'image finale. Deux grandes variantes de SL existent. La méthode « SL-sparse » ne redistribue que les pixels effectivement rendus, alors que la méthode « SL-full » redistribue la totalité de chaque image rendue (incluant les pixels de fond non modifiés).
La Figure 3 ci-dessous représente le pipeline graphique parallèle pour chacune des trois catégories décrites ci-dessus.

Figure 3 : pipeline de rendu parallèle



2.2.2.2. Comparaison des méthodes

Chacune de ces méthodes a des avantages et inconvénients distincts. Un problème important des méthodes « sort-last » et « sort-middle » est celui du chevauchement. En effet, une primitive peut chevaucher plusieurs régions de l'espace-écran. Elle est dans ce cas transmise à tous les processeurs dont elle chevauche la région d'écran. Cela induit un surcoût de traitement par rapport au pipeline de rendu séquentiel : la primitive est traitée autant de fois que le nombre de processeurs à qui elle est transmise. Un autre problème est celui du déséquilibre de charge de travail : si les primitives sont concentrées dans très peu de régions d'écran (voire une seule), elles seront redistribuées à seulement quelques processeurs. Le reste des processeurs sera alors sous-alimenté en travail, ce qui nuit à l'efficacité du système parallèle. Le Tableau 1 ci-dessous résume les grandes caractéristiques des trois méthodes :

Tableau 1 : caractéristiques des méthodes de rendu parallèle

	Avantages	**Inconvénients**
Sort-first	• Chaque processeur implante tout le pipeline graphique pour sa région d'écran • Débit d'interconnexion faible quand suréchantillonnage, ou exploitation de la cohérence image-à-image.	• Sensibilité au déséquilibre de charge de travail • L'utilisation de la cohérence image-à-image nécessite l'emploi d'un mode non-immédiat et un code de gestion des données complexe.
Sort-middle	• Généralité et simplicité : la redistribution a lieu à un endroit naturel du pipeline.	• Coût de communication élevé si le rapport de subdivision est important • Sensible aux déséquilibre de charge entre processeurs de tramage quand les primitives ne sont pas également réparties à travers l'écran.
Sort-last	• Chaque processeur de rendu implante le pipeline entier et est indépendant jusqu'au fusionnement des pixels • Moins sujet au déséquilibre de charge.	• Le trafic en pixels peut être très important (débit requis élevé).

Molnar et al. comparent les trois méthodes en fonction du surcoût de communication induit par la redistribution des primitives lors du tri. Ils supposent dans ce cas que le chevauchement des primitives en rendu SF et SM est négligeable. Le Tableau 2 ci-dessous donne le surcoût de redistribution pour les trois méthodes :

Tableau 2 : surcoût de redistribution du rendu parallèle

	Sort-first	**Sort-middle**	**SL-sparse**	**SL-full**
Coût de redistribution	cNr[primR]	SNr[primD]	NrArE[echant]	NAE[echant]

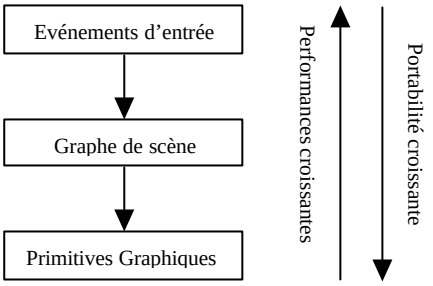
- [primR] et [primD] : coûts de redistribution respectivement d'une primitive brute et d'une primitive d'affichage, proportionnels aux tailles respectives de leurs structures de données.
- [echant] : coût de redistribution d'un échantillon de pixel.
- c : fraction de cohérence image (varie entre 0 et 1), plus elle est faible, plus la cohérence entre deux images successives est forte et moins les primitives doivent être redistribuées aux processeurs.
- S : rapport de subdivision (« tesselation ») entre primitives brutes et d'affichage.
- Nr,d : respectivement nombre de primitives brutes et d'affichage.
- Ar,d : taille moyenne (en pixels) respectivement d'une primitive brute et d'affichage.
- N : nombre d'unités de rendu.
- A : résolution de l'écran (pixels).
- E : nombre d'échantillons par pixel (facteur de suréchantillonnage).

On considère qu'une méthode est préférable à une autre quand son coût de redistribution est inférieur. Si il n'y a pas de subdivision (S = 1) et la cohérence image nulle (c = 1), SM est préférable à SF (en général, [primD] < [primR]). Par contre, si S est important, et/ou c faible, SF est préférable à SM.
Dans le cas d'une cohérence nulle (c = 1), SF est favorisé par rapport à SLS si [primR] < ArE[echant], c'est-à-dire si le coût de redistribution d'une primitive brute est inférieur au coût de redistribution des pixels de la primitive tramée. SM est préférable si [primD] < AdE[echant]. Si les primitives sont simples (petites structures de données) mais couvrent une grande partie de l'écran ou en cas de suréchantillonnage, SF ou SM sont utilisés. Si les primitives sont complexes (grande taille de structure de données) mais de faible taille, SLS est favorisé.
SLS est préférable à SLF quand NrAr < NA, c'est-à-dire quand la complexité en profondeur de l'image est inférieure au nombre de processeurs.
Nous verrons par la suite les conséquences des caractéristiques des trois grandes méthodes de rendu parallèle sur les architectures de rendu parallèle.

2.2.3. Classification plus générale du parallélisme

En visualisation distribuée, on peut considérer une autre classification de parallélisme [RAF03], en fonction du niveau de parallélisation (du plus haut niveau vers le plus bas niveau) : au niveau des événements utilisateur, du graphe de scène, et des primitives graphiques. Selon cette classification, les performances croient vers l'amont, tandis que la portabilité croît vers l'aval. Plus précisément, cette classification considère que les performances croient suivant l'inverse d'une fonction de la quantité de données réparties entre les unités de rendu. Le parallélisme des événements d'entrée consiste à répartir les événements d'entrée entre les différentes unités de rendu. Cela requiert principalement une distribution des paramètres de vue, ce qui représente une quantité de données faible. Le parallélisme du graphe de scène consiste à répartir le rendu des nœuds d'un graphe stockant la scène parmi les unités de rendu. Le parallélisme au niveau des primitives équivaut au rendu parallèle de type SF ou SM (redistribution des primitives graphiques). La Figure 4 ci-dessous représente les différents niveaux de parallélisme, du plus haut niveau vers le plus bas.

Figure 4 : niveaux de parallélisme

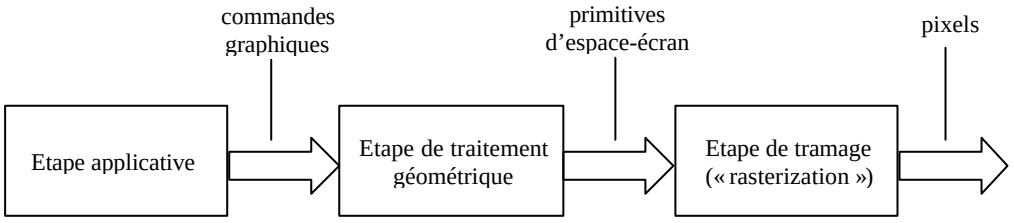


Comme nous le verrons par la suite, cette classification permet de décrire les systèmes de rendu parallèle distribué, tels les grappes de PC qui utilisent un parallélisme de type « sort », et également les systèmes de type « réalité virtuelle » qui utilisent plutôt des méthodes de parallélisme de haut niveau (parallélisme d'événements et de graphe de scène). Néanmoins, nous montrerons dans les parties suivantes que les performances globales d'un système de rendu parallèle utilisant un parallélisme au niveau des primitives graphiques sont potentiellement meilleures que celles d'un système parallèle de plus haut niveau (une taille de la scène plus importante et des performances agrégées de rendu supérieures).

2.3. Le pipeline d'une application de rendu

De manière plus générale que ce que nous avons décrit en 2.2.1, le pipeline d'une application de visualisation 3D comporte trois étapes : les 2 étapes de rendu décrites précédemment (traitement géométrique et tramage) et une étape d'application. La Figure 5 ci-dessous représente le pipeline complet . L'étape initiale de l'application se charge de transmettre les données géométriques et commandes de rendu à l'étape de traitement géométrique de la partie graphique du pipeline. Les commandes de rendu sont celle de l'API (« Application Programming Interface ») d'une bibliothèque graphique (telle qu'OpenGL) pour laquelle existe généralement une implantation matérielle.

Figure 5 : pipeline d'une application de visualisation



Notons que dans le cas de la visualisation scientifique, l'étape « applicative » peut effectuer des opérations nettement plus complexes qu'un simple envoi de commandes graphiques, et comporter notamment des algorithmes de pré-rendu. Elle peut recouvrir dans ce cas les étapes d'importation et d'extraction de la Figure 1. Selon Humphreys et al. [HUM01], on peut classifier les applications selon l'endroit du pipeline où se forme un goulot d'étranglement :

- Rendu : ce sont les applications qui consomment beaucoup de puissance de calcul graphique (de quelque nature qu'il soit). Par exemple une application effectuant du rendu à base de textures transparentes requiert une vitesse de remplissage (tramage) importante.
- Interface : ce type d'applications est limité par la vitesse à laquelle elles peuvent envoyer des commandes de rendu et transmettre les données géométriques, via l'interface du matériel de rendu. Cette vitesse dépend généralement du débit des systèmes d'interconnexion du matériel graphique (bus). Pour

réduire cette limitation, il est possible d'utiliser des particularités des bibliothèques graphiques : listes d'affichage, tableaux de sommets compilés. D'une autre manière, l'utilisation d'applications parallèles (envoyant les données et commandes de rendu en parallèle) permet de réduire ce goulot.

- Résolution : les performances de ces applications ne sont pas limitées, mais elles ne peuvent pas rendre leurs données géométriques à la résolution nécessaire pour en percevoir les détails fins. Pour réduire ce type de goulot, il est nécessaire d'utiliser un système d'affichage à haute résolution.

2.4. Systèmes de rendu parallèle

Dans cette sous-partie, nous présentons les architectures de rendu parallèle dédiées (généralement à mémoire partagée) et les architectures de rendu distribué, qui ont fortement progressé depuis peu.

2.4.1. Architectures parallèles spécialisées

Le rendu est traditionnellement gros consommateur de puissance de calcul. Dans le domaine de la visualisation temps réel, afin de rendre à des vitesses interactives (au moins 5 Hz en visualisation scientifique « interactive », ce qui est peu exigeant, la réalité virtuelle ou les jeux sont souvent calibrés autour de 50-60 Hz pour un confort raisonnable de l'utilisateur sur des durées significatives d'utilisation), des ressources importantes sont nécessaires. De plus, la taille des ressources nécessaires croît avec la taille de la scène à rendre. Pour cela on utilise généralement des architectures spécialisées plutôt que des architectures généralistes. Ces architectures implantent généralement une des méthodes de parallélisme décrites en 2.2.2.1. Jusqu'à récemment, les machines spécialisées dans le rendu utilisaient des architectures propriétaires (généralement de type SMP). Ainsi, une machine type, telle que l'Infinite Reality 2 de SGI, implante matériellement une méthode de rendu plutôt « sort-middle ». Un certain nombre de processeurs géométriques traite la scène à rendre et redistribue les primitives transformées à un ensemble de processeurs de tramage, via un réseau d'interconnexion partagé et à haute vitesse. Les processeurs de tramage effectuent les calculs de visibilité et de couleur [MON97]. Ce type d'architecture, propriétaire, utilise des composants spécialisés. Son coût de développement (en ressources matérielles et en temps) est ainsi important, ce qui rend de telles machines coûteuses à l'achat. Cependant, de telles architectures offrent des performances très élevées, et permettent l'utilisation de modes de rendu avancés (anticrénelage, ou tampon d'accumulation) sans trop grever les performances. La programmation d'applications pour ces architectures requiert souvent l'utilisation d'une API particulière (et propriétaire) pour utiliser la totalité des ressources graphiques de la machine. Par exemple, les architectures SMP de SGI utilisent une version « multi-pipe » (ou multi-canaux) d'OpenGL pour utiliser en parallèle les différents canaux de rendu.

2.4.2. Architectures de rendu distribué

Ces dernières années, les performances du matériel de rendu pour ordinateurs personnels (PC) ont crû de manière spectaculaire. Jusqu'à il y a environ 6-7 ans, la plupart des applications de visualisation sur PC étaient exécutées par le processeur central. La totalité du pipeline graphique était exécuté par le processeur, ce qui rendait les performances des applications médiocres. Pourtant, il existe depuis plus longtemps des cartes graphiques comprenant des circuits dédiés au tramage et (éventuellement) au traitement géométrique. Leurs performances étaient cependant limitées par rapport à celles des stations de travail de visualisation et des machines SMP haut de gamme : au niveau de la vitesse de traitement des circuits et de celle de transmission des données à la carte via le bus graphique. De plus leur coût était généralement prohibitif, ce qui les réservaient à un petit nombre d'applications (CAO, etc.). Le développement important du marché du jeu informatique sur PC a stimulé le développement de nouvelles générations de cartes de rendu 3D, dédiées aux applications grand public. De plus, le développement des bus graphiques spécialisés (le bus AGP) et l'augmentation des performances des processeurs et de la mémoire des PC a contribué à rendre ceux-ci très attractifs pour les applications de visualisation séquentielles (grand public, comme le jeu vidéo, ou professionnelles : CAO, synthèse d'image artistique, etc.).

L'augmentation des performances des PC permet de les utiliser dans des domaines de la visualisation auparavant réservés aux architectures haut de gamme. Ainsi, depuis environ 5 ans, plusieurs laboratoires et centres académiques travaillent à la mise en place d'architectures de rendu distribué sur grappes de PC, une grappe se composant d'un certain nombre de nœuds (au moins 2) interconnectés. Initialement, les grappes ont été utilisées pour effectuer des calculs parallèles de manière distribuée. Cette alternative aux systèmes multiprocesseurs à

mémoire partagée est préférable en terme de coût et d'ouverture (composants très standard du commerce et pas propriétaires). L'augmentation des performances des cartes graphiques PC a donc permis l'utilisation des grappes en rendu distribué, synonyme de rendu à distance (« networked rendering »).
Dans la partie suivante, nous décrivons les différentes architectures et configurations matérielles de rendu distribué sur grappe.

3. Matériel

Les systèmes de rendu parallèle à base de grappes de PC utilisent des nœuds constitués de composants standard (ou COTS, pour « Commodity Off-The-Shelf »). Ainsi, les constituants de chaque nœud (processeur, carte-mère, mémoire, carte graphique, etc.) peuvent être acquis auprès d'un détaillant en PC. Les systèmes d'interconnexion utilisés sont eux aussi des composants standard (cartes d'interface et commutateurs Ethernet Gbit, Myrinet, etc.). Dans cette partie, nous présentons les caractéristiques générales des différents sous-systèmes des configurations de rendu distribué sur grappe. Premièrement, nous décrivons les composants généralement utilisés pour le rendu parallèle sur grappe, et plus particulièrement les caractéristiques des cartes graphiques standard. Deuxièmement nous décrivons les différents systèmes d'affichage utilisés en rendu haute résolution, en insistant sur les systèmes pouvant être facilement couplés avec des grappes. Nous décrivons ensuite les systèmes d'interconnexion puis les systèmes de composition matérielle de sous-images. Enfin, nous présentons plusieurs architectures de rendu distribué à base de grappes.

3.1. Généralités sur les grappes de PC pour le rendu

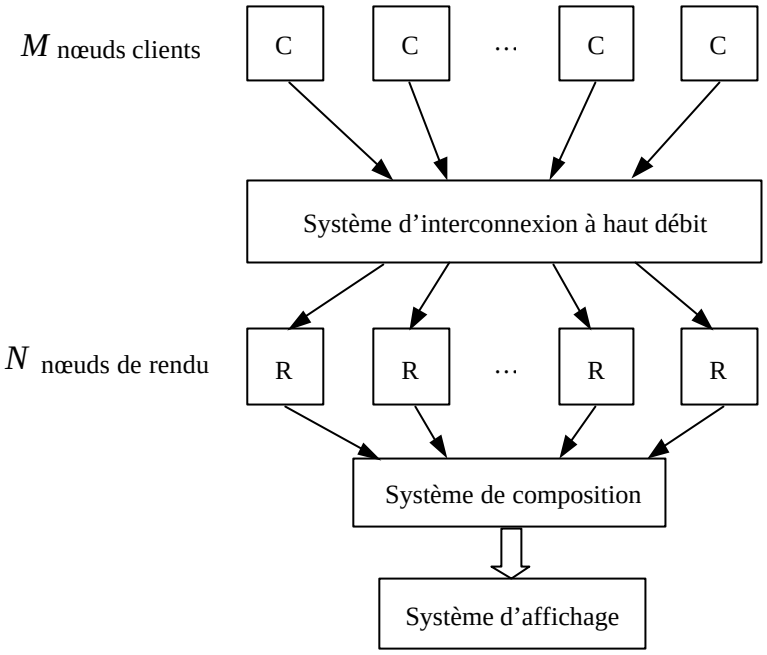
3.1.1. Architecture générale d'une grappe

Un système de rendu parallèle sur grappe se compose des sous-systèmes suivants :

- Un ensemble de nœuds exécutant une application de visualisation.
- Un ensemble de nœuds de rendu, pouvant être commun ou non avec les nœuds de calcul, comportant chacun un accélérateur matériel qui effectue le rendu.
- Un système d'interconnexion rapide reliant les nœuds d'application aux nœuds de rendu et/ou les nœuds de même type entre eux. Ce système est généralement de type réseau local haut débit : Ethernet Gigabit, Myrinet 2000, etc.
- Un système d'affichage relié aux sorties vidéo des cartes graphiques des nœuds de rendu.
- Un système de composition matérielle relié aux sorties vidéo des nœuds de rendu et composant les images individuelles des nœuds de rendu pour produire l'image finale. Ce sous-système est optionnel mais particulièrement utile dans les configurations faisant du rendu SL.

Le principe du rendu distribué est le suivant : la scène à rendre est répartie parmi M clients, ce qui permet de stocker une scène de taille M fois supérieure à la scène pouvant être stockée sur un seul PC. Les nœuds clients soumettent leur partie de scène en transmettant des commandes de rendu aux nœuds de rendu via le système d'interconnexion à haut débit. Chaque nœud de rendu comprend une carte graphique standard chargée d'accélérer le rendu des commandes reçues. Une architecture de rendu distribué sur grappe permet d'agréger les performances de plusieurs PC et ainsi d'atteindre les performances d'un système SMP pour un coût moindre. En effet, si un nœud est capable de rendre k primitives par seconde, alors une grappe de N nœuds (de rendu) peut idéalement rendre Nk primitives par seconde. La Figure 6 ci-dessous représente une architecture générale de grappe de rendu.

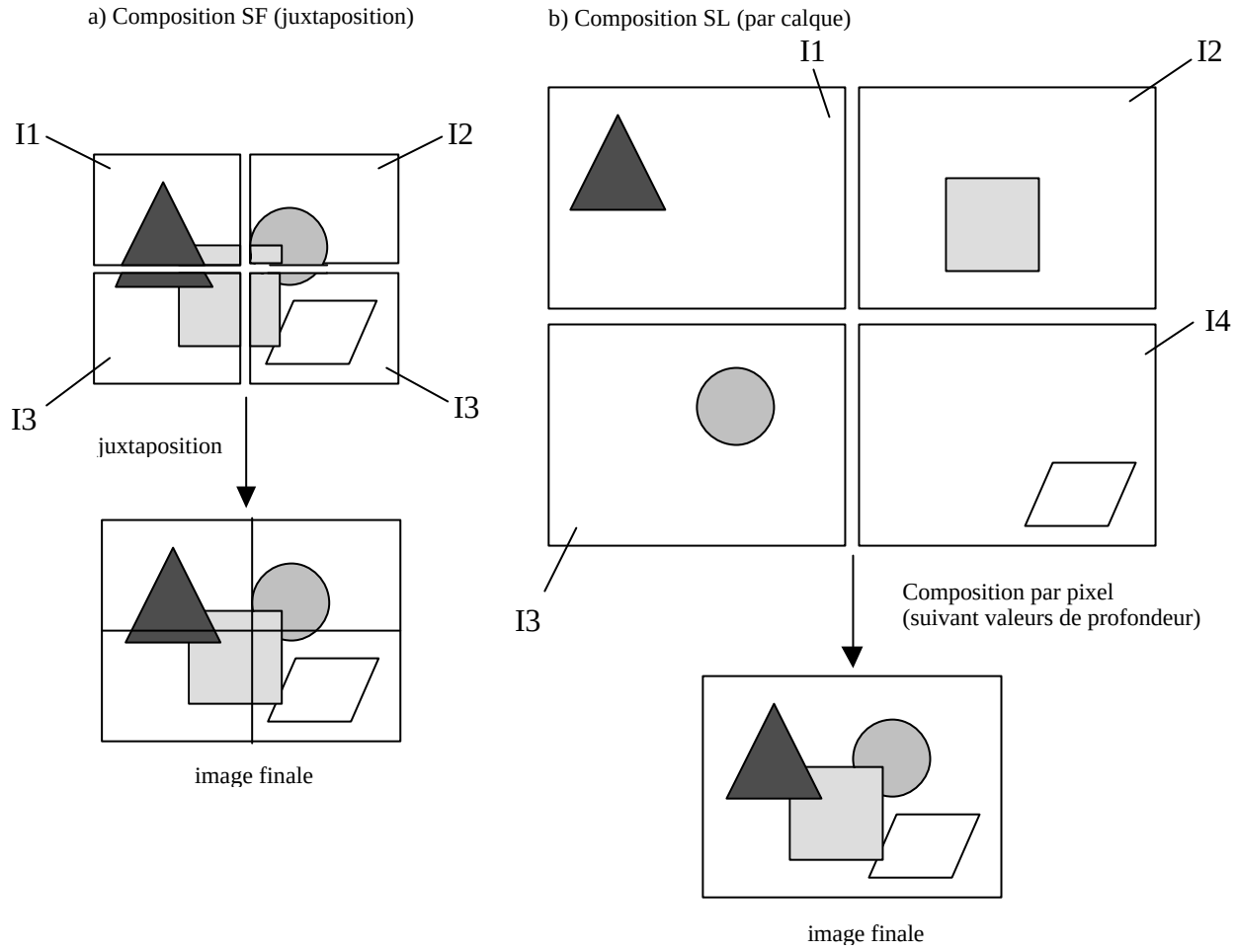
Figure 6 : grappe de rendu distribué



Notons que les nœuds clients et nœuds de rendu peuvent être confondus (par exemple dans le cas de certaines configurations de rendu SL). Afin d'exploiter le potentiel agrégé de la grappe, il est nécessaire d'utiliser une des méthodes de rendu parallèle décrites en 2.2.2. Etant donné les contraintes de l'architecture des PC, la méthode « sort-middle » est difficilement implantable. En effet, SM redistribue les primitives d'affichage entre les étapes de traitement géométrique et de tramage. Or, les cartes graphiques standards n'offrent pas d'accès aux primitives intermédiaires du pipeline, étant donné qu'elles en implantent la totalité. Bien que SM apparaisse comme la méthode de rendu parallèle la plus naturelle (selon [MOLN94]), elle est difficilement implantable sur grappe. Par contre, les méthodes de rendu SF et SL sont bien adaptées au rendu sur grappe. En effet, dans ces 2 méthodes, chacune des unités de rendu implante la totalité du pipeline graphique. Chaque nœud de la grappe rend donc une sous-partie de la scène, et les images partielles résultantes sont assemblées de manière à produire l'image finale.

Par exemple, dans une grappe de rendu « sort-first », chaque nœud est relié à un écran physique correspondant à la partie de l'espace écran vers laquelle il affiche. L'ensemble des écrans forme un pavage correspondant à la surface d'affichage totale. Notons qu'il n'y a pas forcément une correspondance 1 à 1 entre les pavés « logiques » (zones logiques de l'espace écran affectées à chaque nœud de rendu) et les pavés physiques (écrans reliés aux nœuds de rendu). La Figure 7 ci-dessous montre la subdivision entre les N nœuds de rendu d'une grappe pour le rendu SF et SL (pour N=4). Dans un système de rendu SF, chaque nœud est responsable d'une partie disjointe de l'espace écran. Un nœud rend donc toutes les primitives dont la projection recouvre la zone gérée (voir Figure 7 a)). L'image finale est construite par juxtaposition des zones disjointes de tous les nœuds. Dans un système de rendu SL, chaque nœud gère une zone de l'espace-écran de même dimensions que l'image finale. Les primitives sont également réparties parmi tous les nœuds, chaque nœud rend ses primitives dans sa zone-image. L'image finale est obtenue par superposition des différentes image partielles (composition par calque). La valeur d'un pixel (i,j) de l'image finale est calculée à partir des N pixels (i,j) des N images partielles. Typiquement, cette valeur est déterminée suivant les valeurs de profondeur des N pixels (comme dans un z-buffer classique) ou bien en effectuant des opérations de transparence sur les N valeurs. La valeur finale dépend alors de l'ordre des opérations. La Figure 7 b) montre une image résultant de la composition suivant les valeurs de profondeurs. La composition peut se faire de manière logicielle, matérielle (voir 3.4), physique, ou bien par une méthode hybride. Notons que la composition (juxtaposition) physique consiste à utiliser un écran distinct par pavé de l'espace-écran. Les écrans distincts forment un pavage physique de telle manière que l'observateur a l'impression d'observer une image unique et continue.

Figure 7 : division de l'espace-écran en fonction du type de parallélisme



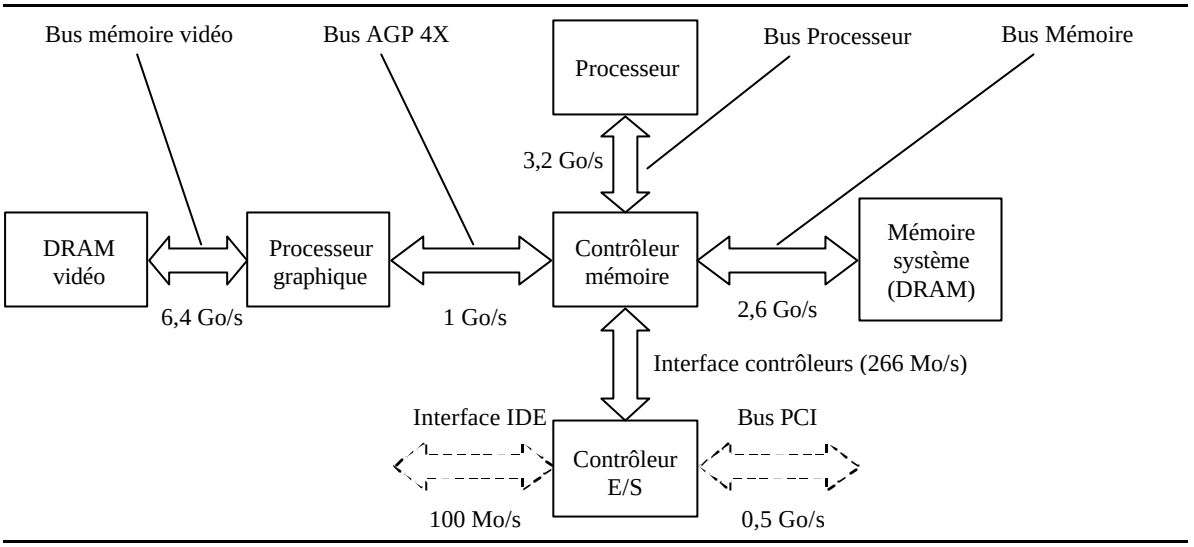
3.1.2. Architecture de base des PC et cartes graphiques COTS

Le cœur d'un système de rendu parallèle sur grappe est constitué par l'ensemble des nœuds de rendu qui sont identiques. Chaque nœud a donc la même architecture de base, celle d'un PC. Les principaux composants d'un PC sont les suivants :

- Carte-mère pour processeur Intel ou compatible (en général IA-32).
- Processeur : généralement architecture de type IA 32 (typiquement AMD Athlon ou Intel Pentium 3 ou 4).
- Mémoire : SDRAM, RDRAM ou DDR-SDRAM.
- Carte d'interface hôte-réseau (Ethernet, Myrinet, etc.).
- Un ensemble de bus, dont le bus graphique AGP et le bus PCI.
- Une carte graphique implantant tout ou partie du pipeline de rendu.

La Figure 8 ci-dessous (voir [NVI02] et [INT02]) représente de manière simplifiée l'architecture d'un PC et les débits associés aux différents bus d'interconnexion (les valeurs indiquées ne sont pas représentatives des technologies les plus récentes mais donnent un ordre de grandeur de débit).

Figure 8 : architecture d'un PC



La carte graphique est représentée de manière simplifiée par un processeur graphique et la mémoire vidéo, le processeur graphique étant relié à un contrôleur mémoire par le bus dédié AGP (« Accelerated Graphics Port »). Le débit du bus graphique (AGP) est généralement très inférieur au débit entre le processeur graphique et la mémoire vidéo (la dernière version du bus AGP a un débit théorique maximum de 2,1 Go/s). Les nœuds de calcul et nœuds de rendu utilisent généralement les mêmes composants. Ces derniers comprennent en plus une carte graphique « haute performance ». Les cartes graphiques pour PC peuvent se répartir dans deux grandes catégories : les cartes destinées aux applications grand public, et les cartes prévues pour les applications professionnelles. Jusqu'à il y a quelques années, les cartes professionnelles utilisaient des processeurs spécialisés. Cependant, les performances des processeurs des cartes grand public leur permettent d'être utilisés à leur tour dans les cartes professionnelles (avec quelques modifications dues aux plus grandes exigences des applications professionnelles).

Une carte graphique grand public a généralement les caractéristiques suivantes :

- Processeur graphique (parfois désigné GPU pour « Graphics Processing Unit ») à haute fréquence et programmable.
- Relativement peu de mémoire vidéo (64-128 Mo sur les cartes actuelles), qui est actuellement du type DDR (Double Data Rate) et avec une fréquence d'horloge élevée.
- Vitesse de texturage (en texels/s) très élevée (de l'ordre de 1 Gtexel/s).
- Vitesse de remplissage (« fill-rate », en pixels/s) très élevée (de l'ordre de 1 Gpixel/s).
- Nombre de triangles rendus/s très élevé (de l'ordre de 100 Mtriangles/s).
- Opérations de traitement géométrique et de traitement de pixels programmables (sur les cartes les plus récentes).
- Gestion de l'anticrénelage de scène (ou FSAA : « Full-Scene Anti-Aliasing »).
- Pas de gestion de la stéréoscopie (par synchronisation externe).
- Lecture des tampons de trame/profondeur par le bus graphique généralement lente.
- Interface d'affichage VGA (analogique) et DVI (numérique).

Les cartes professionnelles étant souvent des versions améliorées des cartes grand public, elles en reprennent les caractéristiques et proposent des fonctionnalités supplémentaires :

- Rendu de lignes anticrénelées.
- Capacité mémoire étendue (capacité de 256 Mo courante).
- Gestion de la stéréo : présence d'un connecteur de synchronisation externe (« genlocking »), et rendu stéréo géré par les pilotes logiciels (« drivers »).

Les caractéristiques les plus importantes sont la vitesse de rendu en triangles/s et la vitesse de remplissage (« fill-rate », en nombre de pixels/s). Le Tableau 3 ci-dessous donne les performances et caractéristiques principales (telles qu'indiquées par leurs fabricants) de cartes standard récentes (voir [EXT02]).

Tableau 3 : caractéristiques de cartes graphiques COTS

Fabricant	ATI	ATI	Nvidia
Modèle	Radeon 9700 Pro	Radeon 8500	GeForce Ti 4600
Modèle du processeur graphique	R300	R250	NV25
Fréquence du processeur graphique (MHz)	325	275	300
Nombre d'unités programmables de traitement de sommets	4	1	2
Vitesse de rendu de triangles (Mtriangles/s)	325	69	136
Nombre d'unités de traitement de pixels	8	4	4
Nombre d'unités de texturage	8	8	8
Vitesse de remplissage de pixels (Gpixels/s)	2,52	1,1	1,2
Vitesse de remplissage de texture (Gtexels/s)	2,52	2,2	2,4
Largeur du bus mémoire vidéo (bits)	256	128	128
Fréquence horloge mémoire (DDR) max (MHz)	650	550	650
Débit mémoire max (Go/s)	20,8	8,8	10,4
Capacité mémoire maximum (Mo)	256	128	128

Les performances indiquées dans le Tableau 3 ci-dessus sont des chiffres théoriques. Les performances réelles dépendent généralement du type d'application de visualisation utilisée (c'est-à-dire de la structure de la scène : taille et nombre des primitives, utilisation d'effets de rendu complexes, etc.). Par exemple, les débits de triangles donnés considèrent généralement le rendu de triangle de 1 pixel, l'étape de tramage n'influe alors quasiment pas sur les performances globales, ce qui n'est pas le cas dans une application réelle, où les triangles sont de tailles très variées.
[HOU02] donne des mesures des performances (sur des applications réelles) de 2 modèles répandus de cartes standard, présentées dans le Tableau 4 ci-dessous :

Tableau 4 : caractéristiques mesurées de cartes graphiques COTS

Modèle de carte	Nvidia GeForce 3	Nvidia GeForce 4
Vitesse de rendu en Mtris/s	25	60
Vitesse de remplissage Mpixels/s	680	800
Vitesse de lecture du tampon de trame (Mpixels/s)	35	135
Vitesse de lecture du tampon Z (Mpixels/s)	22	22
Vitesse d'écriture du tampon de trame (Mpixels/s)	45	45
Vitesse d'écriture du tampon Z (Mpixels/s)	21	21

Les performances d'une carte donnée dépendent de : son architecture, sa fréquence de processeur, sa fréquence mémoire, la qualité de son pilote. Les performances d'écriture et de lecture sont particulièrement importantes pour

les applications de rendu parallèle à base de composition par calque d'images individuelles (notamment en rendu parallèle « sort-last »). En effet, cette méthode de composition nécessite de récupérer le tampon de profondeur (ou « Z-buffer ») de chaque image. Pour cela on utilise des fonctions de lecture du tampon de profondeur fournies par la bibliothèque graphique utilisée (par exemple OpenGL). La vitesse de remplissage est également critique pour le rendu volumique par composition de textures, où des primitives de grande taille sont rendues. La vitesse de rendu en triangles/s est quant à elle plutôt importante dans les applications de rendu plus classique (de type rendu surfacique), lorsque la taille de la scène (en nombre de primitives) est très importante.
Les processeurs graphiques programmables permettent d'outrepasser les fonctions fixes (transformation, éclairage, texturage) implantées dans l'architecture. Cela permet à l'utilisateur de remplacer les opérations du pipeline graphique par un programme qu'il spécifie. Ainsi, il est possible d'implanter des modèles d'éclairage plus sophistiqués (par exemple éclairage anisotrope), d'utiliser un mode d'éclairage par pixel, etc. Cependant ces caractéristiques (récentes) semblent encore peu utilisées en visualisation scientifique.
Enfin, le genlocking est une caractéristique permettant d'asservir la vitesse de rafraîchissement d'une carte graphique à un signal externe. En effet, une carte graphique rend une image aussi vite que possible. Si la fréquence de rendu (en images/s) d'une carte est différente de la fréquence d'affichage du moniteur auquel elle est reliée, on observe un déchirement (« tearing ») dans les images affichées. Toutes les cartes actuelles permettent de synchroniser leur fréquence de rendu à la fréquence de rafraîchissement verticale du moniteur (c'est la synchronisation verticale). Cependant, lorsqu'un système d'affichage en mosaïque est utilisé (voir 3.2.2.3), même si chaque carte est synchronisée avec son moniteur, et si toutes les cartes démarrent leur affichage en même temps, les différents écrans ne sont pas synchrones entre eux. On observe alors des effets de tearing, ce qui nuit à la perception d'une image continue et unique. Le genlocking permet de synchroniser le rafraîchissement de plusieurs cartes, via un signal externe, ce qui élimine le cisaillement.
Dans la sous-partie suivante, nous décrivons les systèmes d'affichage utilisés dans les grappes de rendu.

3.2. Les systèmes d'affichage

Tout d'abord nous décrivons brièvement quelques grandes technologies d'affichage. Nous insistons particulièrement sur les technologies par projection, qui sont particulièrement utilisées dans les systèmes d'affichage à haute résolution. Nous décrivons également l'interface vidéo numérique DVI, qui tend à se généraliser sur les systèmes d'affichage pour ordinateurs, notamment les projecteurs. Puis, nous décrivons les systèmes d'affichage utilisés dans les grappes de visualisation en nous focalisant sur les dispositifs de type écrans en mosaïque (tiled displays). Les détails des technologies et systèmes d'affichage ne sont présentés qu'à titre d'information, le lecteur déjà informé pourra omettre la lecture de la sous-partie présente.

3.2.1. Technologies d'affichage

3.2.1.1. Généralités

Les systèmes d'affichage vidéo permettant d'afficher une image générée par ordinateur présentent une grande variété. Les technologies d'affichage existantes sont ainsi nombreuses et leurs caractéristiques sont très variables. Nous considérons ici deux grands types de technologies suivant la manière dont l'image est générée : de manière directe ou bien par projection. Dans les technologies à affichage direct, la surface d'affichage émet de la lumière de manière directe. Ce sont les technologies CRT (tube cathodique), LCD (« Liquid Crystal Display ») à matrice passive et à matrice active, OLED (« Organic Light Emitting Diode Display »), etc. Dans les technologies d'affichage par projection, la surface d'affichage réfléchit (de manière diffuse) de la lumière projetée grâce à un système optique. Ce sont les technologies de projection LCD transmissive, LCoS, DMD (« Digital Micromirror Device »), CRT tri-tubes, etc.
Un grand nombre de caractéristiques permettent d'évaluer et comparer les différentes technologies d'affichage, cela inclut :

- La résolution spatiale (en points par unité de longueur).
- La luminosité.
- L'angle de vue.
- La vitesse de rafraîchissement.

- La durée de vie.
- La netteté de l'image.
- La taille d'une surface d'affichage unique.
- etc.

3.2.1.2. Technologies d'affichage direct

Les technologies à affichage direct sont utilisées dans les moniteurs classiques pour ordinateurs de bureau, les téléviseurs grand public, etc. Cependant la taille (physique) de la surface d'affichage qu'elles offrent est souvent limitée : à cause, par exemple de problèmes de dimensionnement des tubes cathodiques (CRT), des coûts et contraintes technologiques pour les écrans LCD ou OLED, etc.

3.2.1.3. Technologies d'affichage par projection

Les technologies à affichage par projection permettent généralement une surface d'affichage plus grande qu'avec les technologies directes : en effet il suffit d'augmenter la distance de projection pour que la taille de l'image projetée augmente. Il existe une grande variété de technologies d'affichage par projection. Par exemple, les projecteurs de type CRT tri-tubes : ils utilisent trois tubes cathodiques pour générer trois images (R,V et B). Un filtre différent pour chaque tube permet de générer l'image voulue (R,V ou B). Chaque tube est couplé à un système optique qui permet de projeter l'image sur l'écran. L'image couleur finale est reconstituée par le système visuel humain, de façon analogue à l'image affichée par un téléviseur CRT classique. Ce type de projecteur est bien adapté aux applications grand public (cinéma) mais est encombrant. Nous nous intéressons ici plus particulièrement aux systèmes de projection par micro-affichage. Ces technologies utilisent un système d'agrandissement optique d'une image générée par un micro-système d'affichage [UND00]. Au cœur de ces systèmes on trouve un micro-afficheur : c'est un composant électronique de petite dimension physique (diagonale d'image inférieure à 25 mm et souvent en dessous de 10 mm). Il est basé sur un support de silicium et constitué d'une matrice d'éléments individuellement adressables, que nous nommons ici pixels. Il reçoit en entrée un signal vidéo (souvent numérique) et modifie l'état de ses pixels en fonction du signal grâce à des circuits de contrôle. Nous considérons ici deux classes de technologies de projection par micro-affichage : transmissive et réflective ; il existe des microafficheurs émissifs mais leur luminosité est encore insuffisante pour les applications projectives. Nous insistons particulièrement sur les technologies réflectives. En effet, ces technologies ont été choisies par le CEA pour son mur d'images haute résolution MIRAGE et pour le mur d'images moyenne résolution, que nous avons mentionné en 1, utilisé pour la recherche en Espace Nord.

a) Technologies transmissives

Dans les technologies transmissives, le faisceau de lumière traverse une matrice de points, qui modulent individuellement le faisceau par modulation de leur transmissivité. Les technologies transmissives sont de type LCD : la technologie la plus répandue est HTPS (High Temperature Poly Silicon) : une matrice poly-silicium transparente porte les circuits de contrôle et est recouverte d'une couche de verre. La couche de cristaux liquides est intercalée entre les deux. L'inconvénient des technologies transmissives est que les circuits de contrôle se trouvent dans le chemin optique. La quantité de lumière transmise est donc réduite, et la luminosité de l'image projetée inférieure à celle des technologies réflectives (voir ci-dessous). L'image générée souffre notamment de l'effet de grillage (« screen-door »).

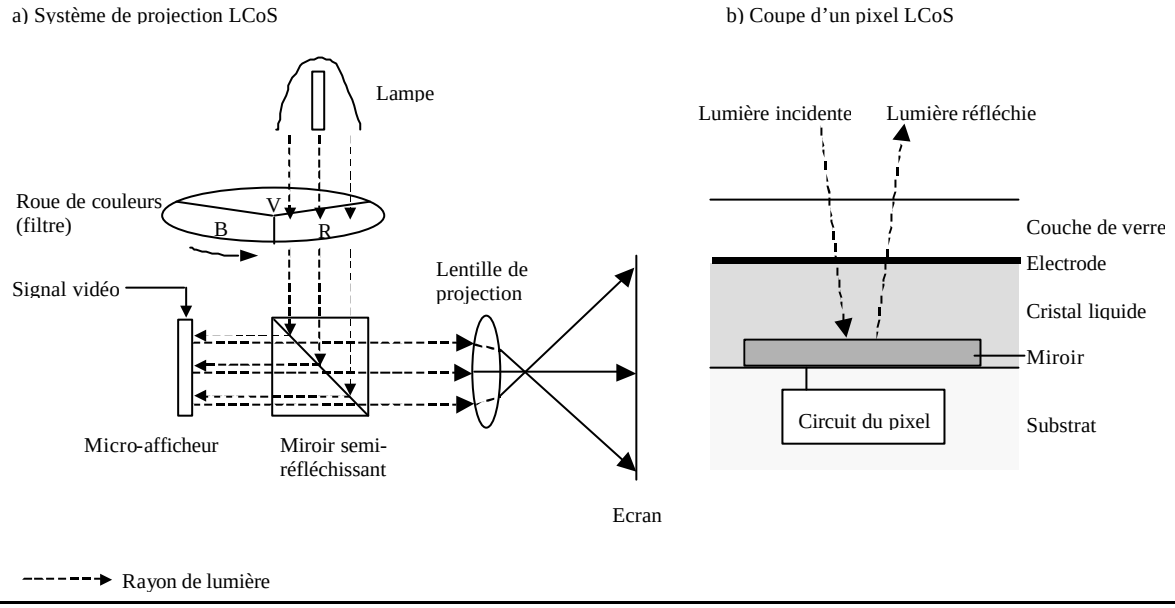
b) Technologies réflectives

Dans ces technologies, chaque pixel comprend un miroir qui lui permet de réfléchir de la lumière. Un faisceau lumineux est projeté sur le micro-afficheur par un système optique. Le micro-afficheur module (suivant le signal vidéo) le faisceau lumineux qui est réfléchi et dirigé via une lentille vers une surface de projection diffuse. Nous décrivons ici 2 grandes technologies, utilisées dans les systèmes par projection : LCoS (« Liquid Crystal on Silicon ») et DMD (« Digital Micromirror Device »).

- LCoS : le principe de fonctionnement des micro-afficheurs LCoS est très proche des afficheurs LCD à matrice active, à deux exceptions : l'affichage est par réflexion, ce qui exige une puce de silicium d'excellente planéité et qualité optique, et la couleur est obtenue par combinaison RVB séquentielle dans le temps (au lieu d'utiliser une triade RVB physique). Un point (pixel) est illuminé par de la lumière polarisée venant d'au-dessus. Une tension entre le miroir du pixel et une électrode contrôle la quantité de lumière réfléchie par le

pixel. Un système NLC (« Nematic LC ») affiche une image couleur en écrivant séquentiellement dans son support CMOS les sous-images R,V et B (technique FSC ou « Field Sequential Colour »). Pour chaque sous-image, l'appareil est éclairé par une impulsion lumineuse de couleur correspondante (respectivement R,V et B) ; cette technique requiert un plus petit nombre de pixels (d'un facteur 3), ce qui réduit la taille et le coût du support CMOS et ne montre aucune séparation spatiale RVB même sous fort agrandissement. Les systèmes FLC (« Ferroelectric LC ») fonctionnent différemment. Chaque sous-image RVB est découpée en plans de bits (8 plans par couleur pour RVB 24 bits). Les plans sont transmis au support qui est illuminé par impulsion de couleur. Le plan le plus significatif (MSBP) reçoit une impulsion 2 fois plus intense que le plan suivant et ainsi de suite. Comme le temps de commutation des FLC est très inférieur à celui des NLC, un plus grand nombre de plans de bits peut être contenu, par exemple, dans la durée d'une image vidéo. Les afficheurs FLC sont bien adaptés à la projection. La Figure 9 ci-dessous représente de manière simplifiée un système de projection de type LCoS et une coupe d'un pixel d'un tel système. Une roue de couleurs RVB permet de moduler la lumière émise par la lampe source. Cela permet de produire séquentiellement les sous-images R,V et B. Le système visuel de l'observateur reconstruit l'image finale par intégration temporelle.

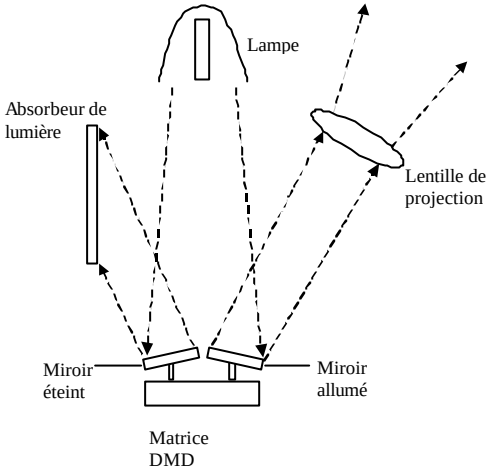
Figure 9 : structure d'un projecteur LCoS et détail d'un pixel individuel



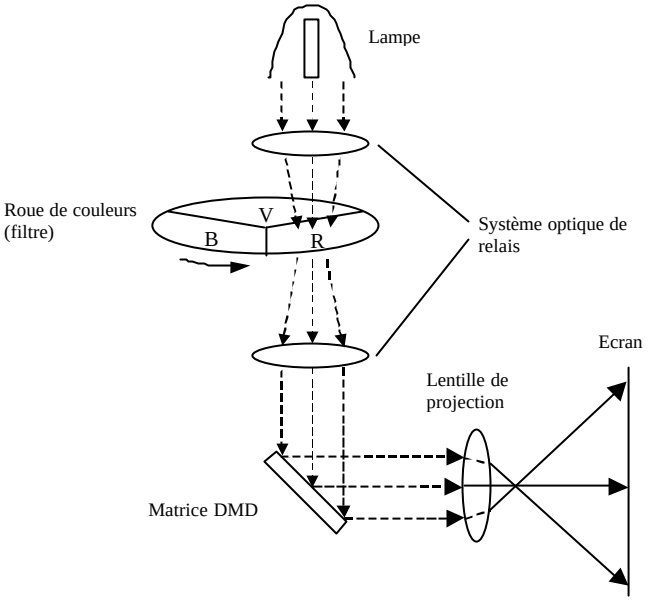
- DMD : cette technologie est la propriété de Texas Instruments et est utilisée commercialement dans les projecteurs DLP (Digital Light Processing) [YOD97]. Elle utilise une matrice de circuits SRAM 1 bit pour incliner (par attraction électrostatique) un miroir micro-usiné placé au-dessus de chaque élément de pixel. Le miroir dirige de manière binaire la lumière vers ou hors du système optique de projection, en aval. Les afficheurs DMD fonctionnent à base de plans de bits (de même manière que les afficheurs FLCoS). La Figure 10 a) ci-dessous représente le principe de fonctionnement d'une matrice DMD. Deux miroirs sont représentés (l'échelle n'est pas respectée) : un miroir est dans l'état éteint, il réfléchit la lumière de la lampe vers un absorbeur de lumière, tandis que l'autre est dans l'état allumé et réfléchit la lumière vers le système optique de projection. La Figure 10 b) décrit la structure (simplifiée) d'un système de projection DMD. Comme dans le système LCoS que nous avons décrit (voir Figure 9 ci-dessus), une roue de couleur RVB est utilisée pour moduler la lumière provenant de la lampe : pour chaque image la roue permet de produire séquentiellement 3 sous-images R, V et B. Le système visuel de l'observateur reconstitue l'image couleur finale.

Figure 10 : principe de fonctionnement et structure d'un système de projection DMD

a) Principe de fonctionnement d'un projecteur DMD



b) Structure d'un système de projection DMD



Notons qu'il existe dans le cas des technologies LCoS et DMD des systèmes utilisant 3 micro-afficheurs distincts (un par primaire R,V, B). Cela permet de maximiser l'efficacité lumineuse du système (qui dans le cas des systèmes FSC que nous avons décrits ci-dessous peut-être réduite au plus de 66 %) et la saturation des couleurs. Cependant, cela a l'inconvénient d'augmenter le coût du système et son encombrement.

Le Tableau 5 ci-dessous résume (de manière qualitative) les grandes caractéristiques des technologies de projection LCD, LCoS et DMD.

Tableau 5 : caractéristiques des technologies de projection

Technologie	**LCD**	**LCoS**	**DMD**
Type de technologie	Transmissive	Réflective	Réflective
Résolution typique	1280 x 1024	1280 x 1024	1280 x 1024
Synthèse de la couleur	Analogique	• Analogique (NLC) • Numérique (FLC)	Numérique
Contraste	Mauvais	Moyen	Bon
Luminosité	Bonne	Bonne	Moyenne
Avantages	• Coût bas • Image nette	• Bonne luminosité	• Effet de grillage réduit
Inconvénients	• Effet de grillage prononcé	• Coût assez élevé	• Moins bonne fidélité de couleur

3.2.1.4. L'interface DVI

L'interface DVI [DDW99] est une interface de connexion numérique de systèmes d'affichage qui est indépendante des différentes technologies d'affichage. Elle utilise un protocole de transmission des signaux vidéo sous forme numérique (TMDS ou « Transition Minimized Differential Signaling »). Jusqu'à récemment, les signaux (numériques) des images générées par une carte graphique étaient convertis sous forme analogique par un convertisseur sur la carte et transmis à un dispositif d'affichage analogique (typiquement un écran à tube cathodique) via le connecteur VGA. Depuis peu, il existe des dispositifs d'affichage (écrans plats) dans lesquels

chaque point peut être adressé individuellement et qui prennent en entrée des signaux numériques. Ainsi, l'interface DVI s'est généralisée afin de permettre une chaîne d'affichage entièrement numérique (du tampon image de la carte graphique à l'écran). Les avantages des signaux vidéo numériques par rapport aux signaux analogiques sont multiples :

- Moins grande susceptibilité au bruit.
- Pas de conversion/reconversion avec un dispositif d'affichage numérique (tel qu'un écran plat).

L'interface DVI utilise jusqu'à 2 liens TMDS, qui relient un transmetteur à un récepteur. Chaque lien comporte 4 canaux : 3 canaux pour les données image RVB et les données de contrôle et un canal d'horloge (fréquence maximale de 165 MHz), chaque canal transmettant un caractère de 10 bits (8 bits de données et 2 bits de drapeaux). Les signaux sont codés avant transmission pour minimiser le nombre de transitions binaires de chaque caractère. Un lien TMDS a un débit maximum de 165 Mpixels/s, ce qui permet de transmettre une image de 1600 x 1200 à 60 Hz (pour un moniteur CRT). Deux liens permettent de transmettre une image 2048 x 1536 à 75 Hz pour un moniteur CRT. L'interface DVI tend à se développer, sa présence maintenant généralisée sur les cartes graphiques COTS permet d'effectuer des traitements (informatiques) post-rendu sur les données image qui étaient impossibles à faire sur des signaux analogiques (par exemple de la composition, voir 3.4).

3.2.2. Les différents systèmes d'affichage

3.2.2.1. Généralités

Comme nous l'avons dit en 2.1, un des buts de la visualisation scientifique haute performance est de visualiser des jeux de données de très grande taille. Cela nécessite de pouvoir afficher des images très détaillées (résolution élevée) sur une surface de grande taille. En effet une image avec une résolution standard (par exemple 1280 x 1024) ne permet pas d'observer des détails d'une grande finesse. Un système d'affichage de très haute résolution permet d'afficher des images détaillées. Cependant, la taille de la surface d'affichage est importante : si la résolution est grande, mais la surface d'affichage trop petite, la séparation angulaire entre deux éléments d'affichage de la surface (à une distance d'observation donnée) peut être très nettement inférieure à ce que peut percevoir un observateur humain. Un bon système d'affichage doit donc effectuer un bon compromis entre ces deux caractéristiques. Nous classons donc les différents systèmes d'affichage en fonction de leur résolution et de la taille de leur surface d'affichage : nous considérons alors 2 catégories : les écrans individuels et les écrans à affichage par mosaïque (ou pavage).

3.2.2.2. Les écrans individuels

Ce sont les systèmes que l'on trouve sur la plupart des ordinateurs personnels. Les technologies qui y sont le plus utilisées sont : la technologie à tube cathodique (CRT) et la technologie à cristaux liquides (LCD) qui, avec la généralisation de l'interface DVI, tend à supplanter la technologie CRT. Les écrans grand public ont généralement des caractéristiques limitées (principalement la résolution) mais acceptables pour certaines applications de visualisation (lorsque la scène n'est pas trop détaillée). Il existe cependant des écrans pouvant afficher des images à haute résolution (supérieure à 1600 x 1200). L'écran T221 d'IBM (utilisé dans la grappe de visualisation DeepView d'IBM, [KLO02]) est un écran LCD à matrice active qui permet d'afficher des images d'environ 9,2 Mpixels (3840 x 2400) à une fréquence de rafraîchissement de 41 Hz. Une de ses caractéristiques notable est la présence de 4 entrées DVI synchrones, ce qui permet d'utiliser plusieurs sources vidéo. Chaque source peut provenir soit d'une même carte graphique (pourvue de plusieurs sorties vidéo), ou bien d'une carte graphique indépendante. Dans ce dernier cas, il est possible d'utiliser plusieurs ordinateurs en parallèle, chacun générant une sous-partie de l'image totale. Cependant, la taille physique d'un tel écran (environ 56 cm de diagonale visible) limite son utilité pour la visualisation en collaboration (plusieurs observateurs pour un même système d'affichage, et observant à des distances différentes).

3.2.2.3. Les écrans à affichage en mosaïque (pavage)

Comme nous l'avons dit en 3.2.1.1, une des caractéristiques des dispositifs d'affichage est la taille maximum de la surface d'affichage. Par exemple, la surface maximum de la technologie CRT est limitée par la taille du tube cathodique, elle-même contrainte par la pression atmosphérique (notons que les technologies CRT à rétro-projection permettent d'avoir une surface d'affichage un peu plus grande). D'autres technologies (directes) de types LCD ou OLED ont des contraintes technologiques qui empêchent leur utilisation pour fabriquer des

surfaces d'affichage de très grande taille. A défaut d'avoir une surface d'affichage unie de très grande taille (ce qui sera probablement possible dans le futur), il est possible d'avoir une surface d'affichage de grande taille en combinant des écrans individuels. De tels systèmes d'affichage sont nommés écrans en mosaïque (« tiled displays ») : ces systèmes sont composés d'une matrice d'écrans MxN idéalement disposés de manière à ce que les jointures (physiques) entre les écrans individuels ne soient pas apparentes. Les MxN écrans constituent alors une surface d'affichage logique unifiée. Nous distinguons deux grandes méthodes pour afficher des signaux vidéo sur des écrans en mosaïque :

- Utilisation d'une seule source : le même signal vidéo est découpé dans le temps en MxN sous parties, chaque partie du signal est étirée et pilote un écran individuel. Cette méthode est utilisée dans les murs vidéo que l'on trouve dans les espaces publics.
- Utilisation d'une source différente par écran : une source vidéo indépendante est utilisée pour piloter chaque écran.

La première méthode est simple à mettre en œuvre : elle ne requiert en plus des MxN écrans qu'une source vidéo unique et un matériel éléctronique permettant de répartir le signal. Cependant, comme la source est unique, la résolution de l'image (en nombre de lignes) affichée sur la matrice d'écrans est identique à celle d'une image affichée sur un seul écran sans répartition. L'image de chaque écran individuel a donc une résolution efficace de H/M lignes (H le nombre de lignes du signal source). Cette méthode ne permet donc que d'augmenter la taille de la surface d'affichage mais pas la résolution de l'image totale. La seconde méthode permet d'augmenter la résolution de l'image totale en fonction du nombre d'écrans. Si chaque source individuelle a une résolution de LxH pixels (L et H étant respectivement le nombre de colonnes et le nombre de lignes de l'image source), alors l'image totale a une résolution de MxHxNxL pixels. De plus, un tel système d'affichage se prête naturellement à l'utilisation conjointe avec un système de rendu parallèle. En effet si chaque source vidéo indépendante est rattachée à un nœud d'une grappe de rendu (via la carte graphique de ce nœud), alors l'image de chaque écran peut être générée par un PC différent. Cela permet d'utiliser le parallélisme d'espace-écran tel qu'il est utilisé dans l'algorithme de rendu parallèle « sort-first » : pour chaque écran individuel, les pixels de sa source vidéo sont générés en parallèle avec les autres écrans. Notons cependant que dans les méthodes de rendu « sort-first » les pavés de subdivision logique ne sont pas nécessairement identiques (en adresse et taille dans l'espace-écran) aux pavés physiques.

3.2.2.4. Les difficultés de la mise en place de systèmes d'affichage par pavage

Les systèmes d'affichage en mosaïque ont néanmoins des contraintes propres importantes. Leurs écrans individuels sont généralement à base de technologies projectives, le plus souvent par projection arrière : un projecteur (relié à une source vidéo) est situé derrière chaque écran et projette dessus son image. [HER00a] décrit les différents contraintes et problèmes technologiques à résoudre pour développer un système d'affichage en mosaïque à base de projecteurs. Selon les auteurs, les problèmes (matériels) à résoudre pour développer un système d'affichage en mosaïque fonctionnel se répartissent dans 3 grandes catégories :

- Choix des matériaux des écrans.
- Choix des projecteurs.
- Fusion des images individuelles, alignement et calibrage.

a) Choix des matériaux des écrans

Les critères de choix principaux du matériau d'affichage sont notamment la qualité de l'image (luminosité, résolution, angle de vue), la rigidité, la masse, l'aire maximale d'une surface continue, le type de montage et le coût. Les systèmes non fixes imposent des contraintes propres : faibles masses de l'écran et du support, montage par tension, flexibilité de l'écran, etc. Ils sont généralement utilisés pour les applications de présentation. Les systèmes fixes sont moins contraignants. Cependant, une méthode de pavage physique doit être utilisée pour des matériaux non disponibles avec une taille arbitraire. Des structures de montages très rigides sont alors nécessaires pour assurer une continuité maximale du pavage et une bonne stabilité de l'écran. Notons que les systèmes fixes sont plus adaptés aux applications de type visualisation scientifique collaborative (qui nécessite une infrastructure matérielle importante).

b) Choix des projecteurs :

La qualité d'un projecteur dépend de nombreux critères : résolution, luminosité, gamme de couleurs, qualité optique, interfaces de données, contrôles de calibrage et configuration, stabilité, taux de rafraîchissement, alignement et ajustement de l'image, planéité du champ d'illumination, etc. La planéité du champ d'illumination est

particulièrement critique pour les écrans en mosaïque : la baisse d'intensité (idéalement nulle) du centre de l'image vers les bords doit être corrigée pour une bonne continuité visuelle du pavage. L'uniformité de couleur est également importante : une égalisation colorimétrique des différents projecteurs doit être effectuée. Les interfaces de données sont aussi importantes pour la qualité : les interfaces analogiques RVB sont sensibles au bruit et posent des problèmes de formatage vidéo. Les interfaces numériques (telles que DVI) permettent de résoudre ces problèmes. Le taux de rafraîchissement est également important pour permettre l'affichage stéréo.

c) Fusion des images, alignement et calibrage :
L'unification d'un ensemble de pavés distincts en un seul espace d'affichage pose trois types de problèmes différents :

- **Fusion (« blending ») des bords des images :** Pour fusionner plusieurs images disjointes en une seule image continue, on utilise des images chevauchantes. Les zones chevauchantes sont alors composées pour obtenir une zone de transition inter-image continue. L'utilisation de zones de chevauchement de taille fixe (par alignement précis des projecteurs) permet la modification du signal d'un projecteur par un « masque d'ombre » virtuel. Cet effet peut être électronique ou logiciel. Cependant, il induit un coût de traitement supplémentaire. De plus, pour les projecteurs LCD, la technique logicielle induit un grisage supplémentaire de la zone de chevauchement. L'utilisation d'un masque physique (cadre modifiant le faisceau de projection) permet d'ombrer l'image projetée de manière ajustable dans la zone de chevauchement, sans modification de l'image source. La structure interne (parcours de la lumière) du système projecteur-écran doit alors être plus complexe.
- **Alignement des images et projecteurs :** Dans un écran en mosaïque, les pavés doivent être alignés de manière précise pour ne pas déformer sa géométrie globale. Chaque projecteur doit donc disposer de 6 degrés de liberté (DDL) de contrôle. Pour cela, un système mécanique de montage et positionnement à six DDL est généralement utilisé. Cependant, un alignement sous-pixel est difficile, même par ajustement fin. Des techniques de traitement d'image permettent d'ajuster les bords des images par déformation mais induisent généralement des baisses de performances de rendu. A partir de 15 projecteurs, une méthode d'alignement automatique est probablement nécessaire.
- **Egalisation des couleurs et luminosité :** La variation des couleurs et luminosité des projecteurs est problématique (surtout pour les modèles bas de gamme). Une sélection grossière permet de déterminer un ensemble de projecteurs dont la variabilité est la plus faible possible (choix d'un même modèle). Un ajustement fin reste alors nécessaire. Des techniques automatiques utilisent des colorimètres ou des caméras numériques en entrée avec un algorithme d'optimisation pour calibrer et corriger l'illumination de l'écran. La baisse d'intensité des pavés individuels peut être traitée par application d'un filtre inverse obtenu par calcul, par exemple sur le canal alpha de la source vidéo.

[HER00b] présente le développement de 2 systèmes d'affichage en mosaïque par projection à l'aide de matériel à faible coût (projecteurs, structure de montage). Ces 2 systèmes sont respectivement à base de projecteurs DMD et LCD. Les auteurs ont développé un système à faible coût de positionnement de projecteur avec 6 DDL. Il permet un ajustement de 0,1 mm à 1,8 m de distance de projection. Une méthode d'atténuation physique du faisceau lumineux (au moyen d'un cadre) est utilisée pour graduer le faisceau sur les bords de l'image afin d'obtenir une transition continue. Le calibrage colorimétrique est effectué automatiquement en temps réel sur tout l'écran avec une acquisition par caméra vidéo standard. Cela donne des mesures précises des composants RVB d'une image pondérés sur plusieurs pixels et en fonction du temps, ce qui permet d'effectuer des corrections via l'interface série de chaque projecteur. La baisse d'intensité a été mesurée sur 2 types de projecteurs (LCD et DMD) au moyen d'une caméra : les auteurs observent que la baisse d'intensité du projecteur DMD est plus forte. En effet, l'optique de projection alignée hors-axe géométrique modifie le comportement de la baisse d'intensité, rendant sa correction plus difficile. Enfin, les auteurs ont développé une méthode logicielle de correction automatique. Elle mesure la distorsion de chaque projecteur (à un facteur de zoom donné) et génère une transformation qui annule la distorsion de chaque image.

Nous décrivons dans la sous-partie suivante les systèmes d'interconnexion utilisés dans les systèmes de rendu sur grappe.

3.3. Systèmes d'interconnexion inter-nœuds

Les systèmes d'interconnexion de grappes de rendu parallèle sont un des composants critiques du système. Comme nous l'avons vu en 3.1.2, dans un nœud individuel, l'interface entre le système processeur-mémoire et

l'accélérateur graphique est assurée par le bus AGP, qui a un débit élevé (2,1 Go/s dans sa version AGP 8x). Cependant, dans les systèmes de rendu parallèle, les différents nœuds doivent être interconnectés pour interéchange de données, que ce soient des données géométriques, des données image (pixels) ou bien des commandes graphiques. Dans un système implantant la méthode de rendu parallèle SF, des nœuds d'application envoient des commandes graphiques (code des commandes et données géométriques) aux nœuds de rendu qui exécutent ces commandes via leur accélérateur graphique local. Le système d'interconnexion transmet ces commandes, la vitesse de rendu agrégée du système (en triangles/s) dépend donc du débit du système d'interconnexion. Si ce dernier est trop faible, alors le débit de rendu du système est limité par l'interface entre les nœuds. Un système d'interconnexion à débit élevé est donc nécessaire. Nous considérons qu'il doit avoir les caractéristiques suivantes :

- Débit maximum par lien de connexion élevé.
- Doit offrir un débit extensible en fonction du nombre de nœuds (si le débit est fixe, alors cela crée un goulot d'étranglement quand le nombre de nœuds augmente).
- Latence de transmission faible.
- Coût modeste.
- Compatibilité matérielle avec l'architecture des PC (c'est-à-dire existence d'un adaptateur hôte prévu pour PC).
- Portable sur de nombreux systèmes d'exploitation.

Depuis peu, les réseaux locaux à haut débit ont atteint des performances qui les rendent bien adaptés à l'utilisation en tant que systèmes d'interconnexion de grappes de rendu. Leurs caractéristiques principales sont :

- Liens bidirectionnels ayant un débit de l'ordre de 1 Gbit/s.
- Extensibilité du débit en utilisant plusieurs liens et des commutateurs (ce qui est impossible dans les réseaux locaux de type Ethernet partagé).
- Faible coût.

Les deux réseaux locaux les plus utilisés dans les grappes de rendu parallèle à base de PC sont : Ethernet Gbit commuté et Myrinet. Le réseau Ethernet Gbit est l'option la plus simple, en effet Ethernet est un réseau local très répandu et sa version à haut débit est devenue plus accessible (en coût) depuis quelques années. Myrinet [BOD95] semble a priori plus adapté aux besoins du rendu à distance sur grappe. En effet, alors qu'Ethernet a originalement été développé pour des réseaux de postes de travail de type bureautique, Myrinet a été développé à partir de réseaux d'interconnexion pour les ordinateurs multiprocesseurs à passage de message. Il reprend donc un grand nombre de caractéristiques de ce type de réseaux spécialisés :

- Débit maximum élevé : 1,28 Gb/s par lien.
- Fiabilité très importante (taux d'erreur de 10^{-15} pour 25 m).
- Protocole de routage simplifié (par rapport au routage « store-and-forward » des réseaux locaux), ce qui augmente le débit moyen.
- Extensibilité importante (en regroupant plusieurs liens sur un commutateur).

Le débit maximum de Myrinet est sensiblement supérieur à celui de l'Ethernet Gbit. Cependant le débit du réseau Ethernet commuté (« switched ») peut également être étendu en utilisant plusieurs liens par commutateur et plusieurs commutateurs si besoin est. Le réseau QsNet [QUA03] est un réseau d'interconnexion similaire aux deux premiers mais offre le débit le plus important (2720 Mb/s par lien). [HOU02] répartit les systèmes d'interconnexion en trois catégories de performances : bas de gamme, moyenne gamme et haut de gamme, dans lesquelles ils situent respectivement les réseaux Gigabit Ethernet, Myrinet 2000 et Quadrics.

Il est important de noter que le bus système de chaque nœud, auquel est relié l'adaptateur hôte du système d'interconnexion, est critique pour de bonnes performances. En effet, sur les PC grand public, le bus système est le bus PCI 32 bit à 33 MHz, qui offre un débit maximum de 1 Gbit/s, égal au débit d'un lien Gbit Ethernet. L'utilisation de réseaux de type Myrinet ou Quadrics requiert donc l'utilisation de carte-mères haut de gamme, avec des bus systèmes dont le débit maximum permet d'exploiter au mieux les performances des réseaux les plus rapides. Ainsi la présence d'un bus PCI 64 bits (avec une fréquence d'horloge de 64 MHz) est nécessaire pour exploiter les performances d'un réseau de type Quadrics. [HOU02] décrit également suivant 3 types de configuration les composants (« chipsets ») requis sur les cartes-mères des nœuds suivant le type de réseau utilisé. Le Tableau 6 ci-dessous donne les caractéristiques principales d'adaptateurs-hôte de plusieurs technologies (voir [MYR03], [QUA03], [DIS03], [ALA03] et [HUG03]).

Tableau 6 : caractéristiques de systèmes d'interconnexion

Fabricant	Myricom Inc.	Quadrics	Dolphin Interconnect Solutions Inc.	Alacritech	Intel
Technologie d'interconnexion	Myrinet	QsNet	SCI (Scalable Coherent Interface)	Gigabit Ethernet	Gigabit Ethernet
Modèle	M3F-PCI64B, C	QM400	PCI-64/66 / D331	1000x1 Copper Gigabit Adapter	PRO/1000 XT
Interface hôte	PCI 64/32 bit à 66/33 MHz	PCI 64 bit, 66 MHz	PCI 64/32 bit à 66/33 MHz	PCI 64/32 bit à 66/33 MHz	PCI 32bit à 66/33 MHz
Lien physique	Fibre optique	Cuivre	Cuivre	Cuivre	Cuivre
Débit bidirectionnel maximum (Gb/s)	4,0	5,44	10,66	2	1
Débit unidirectionnel réel maximum (Gb/s)	1,98	2,24	2,60	NC	0.91 à 0.95

Comme nous venons de le voir, le débit maximum unidirectionnel des meilleurs réseaux d'interconnexion pour PC est encore inférieur au débit maximum du bus graphique d'un seul PC (débit maximum unidirectionnel de 2700 Mb/s pour Quadrics comparé au débit maximum de 16800 Mb/s pour le bus AGP 8x). Les performances technologiques ne cessent de croître et des réseaux encore plus haut de gamme, comme Ethernet 10Gbit, seront probablement abordables dans quelque temps. Cependant les performances des bus graphiques ne cessent également d'augmenter, ainsi le bus PCI Express x16, qui va remplacer à court terme le bus AGP, devrait offrir un débit maximum de 40 Gb/s. L'idéal étant qu'un lien d'interconnexion offre un débit proche du débit maximum du bus graphique d'un PC unique, la marge de progression est encore importante.
Dans la sous-partie suivante, nous décrivons plusieurs matériels de composition d'image et la manière dont ils peuvent être utilisés pour implanter le rendu parallèle.

3.4. Les systèmes de composition matérielle

3.4.1. La composition

Nous décrivons dans cette partie les matériels de composition. Comme nous l'avons vu en 3.1.1, l'image finale rendue en parallèle par une grappe de N PC doit être composée à partir des images partielles des N nœuds. En rendu parallèle SF, les images partielles (disjointes) doivent être juxtaposées pour donner une image finale. Cette juxtaposition peut être physique (utilisation d'un écran par image disjointe, c'est-à-dire un système d'affichage en mosaïque) ou bien l'affichage peut se faire sur un écran unique. Dans ce cas, la composition doit être faite avant affichage de l'image finale, par des techniques logicielles ou matérielles. L'algorithme de rendu parallèle SL répartit la scène à rendre entre différentes unités de rendu. Chaque unité de rendu rend sa partie de scène vers une image partielle. Les images partielles sont ensuite transmises à un réseau de processeurs qui calcule par composition (par calque) l'image finale. Les opérations de composition consistent typiquement à comparer deux pixels et d'affecter à la valeur du pixel résultat la valeur du pixel avec la valeur de profondeur la moins élevée (composition dite en profondeur ou en Z) ou bien à calculer la valeur du pixel final en faisant une somme pondérée des 2 valeurs des 2 pixels multipliées par des coefficients de transparence (ou alpha) : c'est la composition par transparence. Ces opérations sont simples et répétitives mais leur nombre peut être très important, notamment pour la méthode « SL-full » où chaque image partielle a la même taille que l'image finale. Cela nécessite : un réseau d'interconnexion des processeurs de composition qui soit rapide, et des processeurs de composition eux-même rapides afin de diminuer la durée de cette phase. La composition logicielle (exécutée par le processeur central) est typiquement facile à implanter, mais est coûteuse en temps de calcul.

Il existe des composants matériels spécialisés permettant d'accélérer cette phase de composition, ce qui réduit sa durée au minimum. Ces composants prennent en entrée un ou plusieurs flux vidéo, effectuent un traitement dessus et produisent en sortie un ou plusieurs flux vidéo.

3.4.2. Tampons de trame matériels externes

3.4.2.1. Lightning-2

Lightning-2 [STO01] est un sous-système de composition conçu pour fonctionner avec un système de visualisation parallèle à base de grappe de PC. Il relie plusieurs nœuds de rendu à plusieurs écrans par interface DVI et permet à un pixel généré par un nœud quelconque d'être dynamiquement affecté à un endroit quelconque de n'importe quel écran. Chaque pixel peut être traité par une fonction de composition d'image (Z, couleur-clé).

L'unité de base de Lightning-2 (L2) est un module, constitué d'un empilement de 4 tranches indépendantes, reliées verticalement par une chaîne de composition. Une tranche fonctionne de la manière suivante : un contrôleur d'entrée récupère le signal vidéo par l'entrée DVI, le répète sur une sortie vers un éventuel autre module L2, le décode en fonction d'informations de contrôle empaquetées et transmet les pixels à un contrôleur mémoire qui les écrit dans un tampon-image. Une unité de composition (UC) lit le tampon-image et le compose avec le tampon de la tranche précédente puis transmet le résultat à la tranche suivante. La chaîne de composition d'un module fonctionne de manière pipelinée, les données y étant injectées dans l'ordre de trame. Elle peut être reliée à d'autres chaînes (colonnes) via les répéteurs DVI de chaque module.
Chaque pixel est codé sur 32 bits : 24 bits RVB et 8 bits de contrôle de composition. La chaîne a une largeur de 8 pixels et un débit maximum de 533 Mpixels/s. Les données sont multiplexées dans le temps avant composition puis démultiplexées en fin de chaîne pour être dirigées sur (au plus) 8 sorties DVI. La correspondance aval entre entrée et sorties pour chaque pixel est donnée par l'utilisateur, avec une granularité par fragment de ligne.
Chaque fragment est contrôlé par un en-tête de 48 bits qui donne : la taille du fragment (implicitement la position du prochain en-tête/fragment), son adresse cible, l'indice de colonne destination dans la matrice L2, et 8 bits de paramètres de composition. La correspondance entre entrées et sorties s'effectue donc avec une granularité arbitraire et de manière synchrone, les données pixels et de correspondance étant transférées sur le même canal.
De plus, le surcoût de préparation des informations de correspondance est distribué sur les différents nœuds de rendu. Chaque tranche L2 utilise un double-tampon et doit recevoir une image complète avant de basculer et d'accepter l'image suivante des nœuds de rendu. L2 utilise un canal retour (via une interface RS-232) pour indiquer aux nœuds de la grappe qu'il a reçu l'image n et est prêt à accepter l'image n+1. Ce protocole induit cependant une latence d'affichage d'une image. L2 implante plusieurs opérations basiques de composition d'images, dont :

- Assemblage de zones disjointes d'une même image (affichage en mosaïque).
- Transparence par couleur-clé : les pixels d'une couleur-clé (spécifiée par l'utilisateur) sont ignorés lors de la composition. Cela permet de gérer par exemple des éléments complexes d'interfaces graphiques.
- Composition par profondeur : elle nécessite 2 valeurs par pixel (profondeur et couleur), on doit récupérer le tampon-Z sur la sortie DVI de chaque accélérateur. Cela requiert une lecture du tampon Z en mémoire de chaque nœud, suivie d'une réécriture des données Z dans le tampon image de la carte graphique. La carte stocke donc dans son tampon image son image couleur et l'image Z correspondante. De plus, la valeur Z de chaque pixel doit être disponible en même temps que sa valeur de couleur pour l'unité de composition. Pour cela, on place l'image de profondeur sur l'afficheur 0 (c'est-à-dire vers la sortie DVI 0) et l'image couleur sur l'afficheur 1. Grâce au multiplexage temporel, les informations de couleur et Z arrivent en succession rapide temps à une UC donnée. L'UC compare les valeurs pour l'afficheur 0 (valeurs de Z) et conserve les valeurs de couleur (sur l'afficheur 1) en fonction du résultat.

Les auteurs présentent ainsi des mesures de performances en rendu « sort-last » avec composition Z sur une grappe de 8 nœuds, avec une scène de 0,9 Mtriangles, répartie statiquement sur les nœuds et rendue 10 fois par image. Ils observent une vitesse de rendu de 106 Mtriangles/s et une fréquence d'affichage de 11,4 Hz (à une résolution finale de 800 x 600, chaque nœud utilise un tampon de trame de 1280 x 1024 pour pouvoir y écrire ses images Z et couleur). L'accélération est de 6,8 pour 8 nœuds.

3.4.2.2. Metabuffer

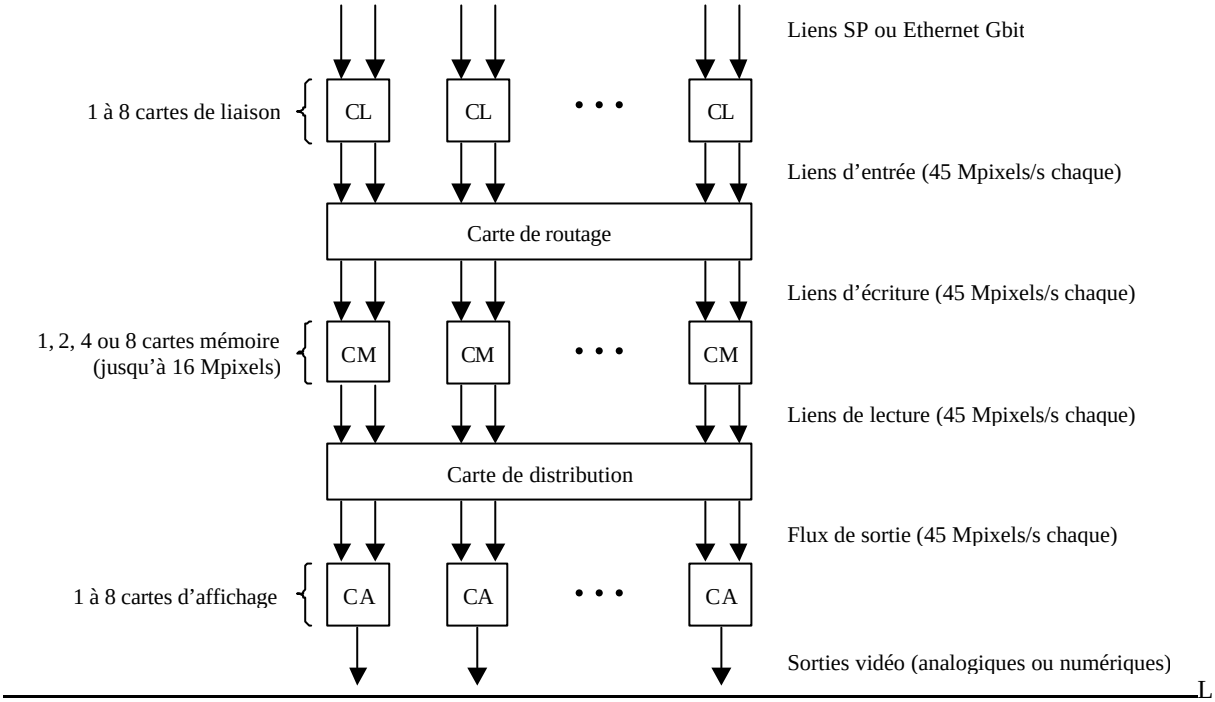
[BLA00] décrit l'architecture d'un appareil de composition d'images multi-sources : le Metabuffer. Il prend en entrée plusieurs flux vidéo venant de cartes graphiques standard, les compose et envoie le résultat sur plusieurs écrans. Il supporte plusieurs PC et plusieurs écrans, en nombres indépendants. Chaque PC (unité de rendu) gère une région de dimensions arbitraires alignée dans l'espace d'affichage global. Un PC a donc accès à tout l'espace global, qui se comporte comme un seul tampon image unifié (« Metabuffer » ou MB). Le MB relie les unités de rendu aux écrans au moyen de chemins de données en pipeline. Chaque unité de rendu est reliée à un tampon image d'entrée (TE), relié à un pipeline de M unités de compositions (UC), M le nombre d'écrans. Les UC équivalentes des N unités de rendu sont également reliées en pipeline, permettant de composer un écran entier (on a une matrice de MxN UC). Quand une unité de rendu transmet son image, chaque UC ne récupère que les données intersectant l'écran qu'elle gère. Chaque UC envoie ensuite ses données dans l'UC de la ligne suivante, où s'effectue une composition en Z. En bas de chaque colonne, un tampon image stocke les données résultantes, effectue un post-traitement dessus (antialiasing) et transmet l'image par sortie DVI. Dans la première ligne de l'image de chaque unité de rendu sont codées les informations indiquant au TE à quelles UC doivent être transmises les données. Ces informations décrivent chaque segment de l'image (position, dimensions, UC attribuée). Comme les UC sont pipelinées, un manque de données créera une erreur de rendu dans l'image finale.

Les auteurs montrent cependant que le débit requis par le MB est constant, quelles que soient l'orientation et la position des différentes fenêtres. Cependant, l'architecture n'a été implantée qu'en logiciel : les auteurs en ont réalisé un simulateur écrit en C++. Bien qu'il ne fonctionne pas en temps réel, il implante toutes les caractéristiques architecturales du MB. Les auteurs ont effectué plusieurs tests qualitatifs dont un test de rendu par multirésolution, en utilisant des images pré-rendues comme sources Z et couleur (les données proviendront de sorties DVI dans la machine réelle). Trois images (fenêtres) et 4 écrans sont utilisés. Ils montrent que l'image finale est correctement composée et que l'anticrénelage du MB (par suréchantillonnage) adoucit les transitions abruptes tout en gardant le reste des détails.

3.4.2.3. SGE

Le système SGE (Scaleable Graphics Engine), voir [HOC] et [PER01], est un tampon de trame matériel. Il prend en entrée des images partielles et les compose en une image unique et continue. Une image partielle est produite par un nœud de rendu. Jusqu'à 16 nœuds peuvent être reliés au SGE, via des liens Ethernet Gbit. Notons que dans une première version, les liens d'entrée étaient des liens SP (1,2 Gbit/s par lien), qui ne permettaient de connecter que des machines SP (architecture propriétaire d'IBM). Chaque nœud peut contribuer à n'importe quelle partie de l'espace-écran. Le partitionnement de l'espace-écran peut donc être totalement arbitraire (régions multiples, chevauchantes et non-rectangulaires). Le SGE peut donc effectuer un masquage précis au niveau de chaque pixel. Pour cela, il transmet les données sous formes de paquets de commandes : chaque commande spécifie une bande de pixels horizontale ou verticale et de taille arbitraire. La Figure 11 ci-dessous représente l'architecture du SGE.

Figure 11 : architecture du Scaleable Graphics Engine



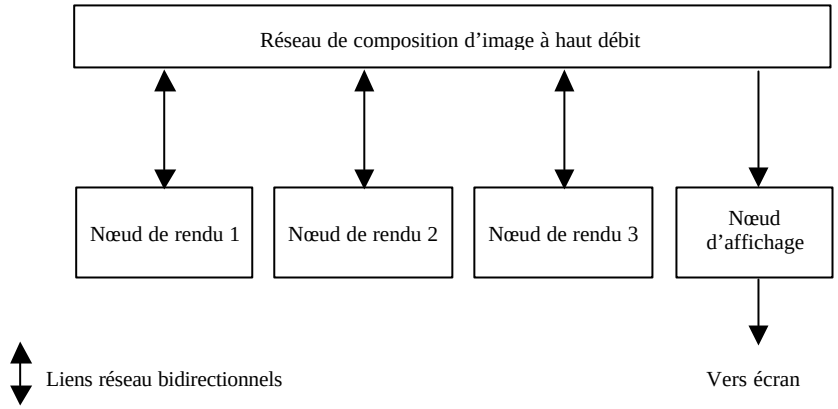
Les commandes (pixels) sont transmises des nœuds de rendu aux cartes de liaison par des liens réseau, chaque carte supportant 2 liens. La carte de routage distribue les pixels, générés à partir des commandes, vers les cartes mémoires grâce à des liens multiples. Chaque lien d'entrée a un débit de 45 Mpixels/s. Le tampon de trame est constitué de plusieurs carte mémoires (1, 2, 4 ou 8) et est mis à jour en parallèle par les cartes de liaisons. Chaque carte est constituée de 2 bancs mémoire : chaque banc stocke 1 Mpixel et peut être écrit et lu de manière concurrente à un débit de 45 Mpixels/s pour chacune des 2 opérations. Chaque pixel est codé sur 24 ou 16 bits, le tampon de trame a une capacité de 16 Mpixels (avec 8 cartes mémoire). Les cartes mémoire sont reliées à une carte de distribution qui dirige les pixels vers des cartes d'affichage. Chaque carte d'affichage est contrôlée par 2 liens de sortie (avec un débit de 45 Mpixels/s chacun). Une carte génère un flux vidéo analogique ou numérique, suivant le type d'écran utilisé. Des cartes avec sortie analogique ou numérique sont disponibles. Le SGE est ainsi capable de piloter 8 cartes d'affichage avec un débit de 720 Mpixels/s. Le SGE permet d'utiliser un schéma arbitraire de partitionnement de l'espace-écran ; de plus, il permet d'utiliser une configuration d'affichage arbitraire (écran en mosaïque, mono-écran, etc.). Cependant, à la différence de L2 et du Metabuffer, il ne peut effectuer de composition suivant des valeurs de profondeur, de transparence, ou bien d'anticrénelage.
Dans la sous-partie suivante, nous décrivons plusieurs configurations matérielles de rendu sur grappes.

3.5. Architectures de rendu

3.5.1. L'architecture Sepia

L'architecture Sepia [MOLL99] est une architecture de rendu distribué de composition matérielle d'images, qui implante le rendu parallèle « sort-last » d'un jeu de données partitionné. Elle utilise un réseau de transfert d'image à haut débit reliant les nœuds de rendu en pipeline. La Figure 12 ci-dessous représente une grappe de rendu Sepia.

Figure 12 : architecture Sepia



Les nœuds de rendu (PC) sont reliés en pipeline par un réseau de composition d'image (réseau spécialisé ServerNet). Chaque nœud de rendu génère une image partielle en rendant sa portion de scène, la compose avec l'image du nœud précédent et la transmet au nœud suivant. Un nœud utilise un accélérateur graphique OpenGL standard pour rendre sa scène et possède une carte Sepia, constituée d'un moteur de composition matérielle (implanté sur 4 circuits FPGA) et d'un composant d'interface ServerNet. La carte Sepia extrait l'image de son accélérateur local par DMA via le bus PCI. L'image partielle du nœud amont est reçue par le composant ServerNet. Les 2 images sont composées (en Z ou alpha) par le moteur de composition (1 circuit FPGA) et l'image résultante est envoyée vers le nœud aval via le composant ServerNet. La méthode de composition est programmable (par exemple en Z ou alpha). Les auteurs identifient plusieurs goulots d'étranglement dans leur architecture : le goulot principal est celui de l'interface réseau : les 2 liens ServerNet (entrée et sortie) ont un débit utile d'environ 30 Mo/s. Cela permet un débit d'environ 4 Mpixels/s en rendu SL (24 bits RVB et 24 bits Z).

[LOM01] décrit l'architecture de rendu par composition Sepia 2, qui reprend l'architecture Sepia et l'améliore en utilisant des composants plus performants. Son objectif est de rendre des données volumiques de 1024^3 voxels (voir 5.2) à des vitesses interactives. Pour cela, la carte de rendu volumique VolumePro est utilisée au lieu d'un accélérateur graphique OpenGL standard. Le jeu de données est partitionné sur 8 nœuds en sous-volumes de 512^3. Chaque nœud rend sa scène avec la carte VolumePro dans une image de 512^2 qui est lue via le bus PCI et copiée dans une super-image de 1024^2. Chaque super-image est composée avec celle provenant du nœud amont via le réseau ServerNet-2 (2 liens de 180 Mo/s). Le réseau de composition n'est plus en pipeline (comme Sepia) mais utilise une structure en crossbar (grâce à un commutateur) de manière à n'avoir une latence que d'1 à 2 images (composition par bloc, voir 5.2.2). Selon les auteurs, ce système permet des vitesses de rendu de 24 à 28 images/s. De plus, ils décrivent leur système comme extensible : l'utilisation de plusieurs commutateurs permet de créer des structures récursives multiétages, ce qui permettrait de dimensionner la scène en fonction du nombre de nœuds tout en gardant une latence faible.

[HEI02] décrit une méthode d'extraction de l'image des accélérateurs graphiques individuels, dans l'architecture Sepia 2, par l'interface DVI (au lieu d'utiliser pour cela le bus PCI). Cette méthode permet d'extraire le tampon Z d'un accélérateur et les données de transparence du tampon de trame, selon les besoins de l'application. Chaque nœud de rendu utilise une carte Sepia-2, reliée par interface DVI à l'accélérateur graphique. La procédure d'acquisition DVI est la suivante :

1. L'application rend ses primitives dans les tampons RVB et alpha/Z
2. La carte graphique envoie ses données RVB par le port DVI
3. La carte Sepia acquiert les données RVB par DVI
4. La carte graphique envoie les données alpha/Z par le port DVI
5. La carte Sepia acquiert les données alpha/Z par le port DVI

Puis l'application rend l'image suivante (retour à la 1ère étape). Pour acquérir les données Z et alpha par interface DVI, on les copie dans le tampon image. Pour copier les données Z, on utilise la fonction CopyPixels d'OpenGL avec une extension spécifique au modèle de carte graphique utilisé. Une extension permettant de sauvegarder les

données RVB est aussi utilisée. Pour écrire les données alpha dans le tampon image, la méthode varie selon utilisation ou non du double-tampon :

1. double-tampon : copie des données alpha du tampon avant vers dans le tampon arrière
2. simple-tampon : copie des données alpha du tampon avant dans le tampon avant.

Deux méthodes d'acquisition peuvent être utilisées : synchrone ou asynchrone. Les auteurs montrent que la méthode asynchrone permet de minimiser le temps de blocage de l'application de chaque nœud dû à l'acquisition. Cette méthode permet de rendre des jeux de données volumiques de 512^3 voxels sur 8 nœuds à 30 im/s (pour une résolution de 1024^2).

3.5.2. Deep View

L'architecture Deep View [KLO02] se constitue d'une grappe de 8 PC bi-processeurs à 866 MHz avec 1 Go de RAM. Les nœuds sont interconnectés par un commutateur Myrinet. Les nœuds sont également reliés à un compositeur matériel, le SGE (voir 3.4.2.3), via un commutateur Ethernet 1 Gbit. Le SGE accepte jusqu'à 16 liens Ethernet Gbit. Il est relié à l'écran T221 d'IBM (voir 3.2.2.2) par 4 sorties DVI synchrones. Comme le T221 affiche plus de 9,2 Mpixels à 41 Hz (soit un débit en entrée requis d'environ 377 Mpixels/s), le débit d'une seule sortie DVI (et donc d'une seule carte graphique) est insuffisant (165 Mpixels/s). L'utilisation du SGE permet de juxtaposer plusieurs flux vidéo permettant d'atteindre la résolution et le débit requis par l'écran T221.

3.5.3. DVG

[ORA02] présente l'architecture DVG de rendu distribué par composition. DVG utilise une architecture en chaîne similaire à celle décrite par la Figure 12. De une à N unités de rendu (PC avec une carte graphique standard) sont reliées par un bus dédié au trafic de pixels. Selon son constructeur, le système offre ainsi des performances linéairement extensibles en fonction du nombre d'unités chaînées. Cependant, les méthodes de parallélisme implantées par DVG diffèrent de celles que nous avons décrites jusqu'ici. Le mode de parallélisme (SL) par division de la scène est similaire au fonctionnement de l'architecture Sepia de base (voir 3.5). La totalité de la scène est répartie entre les unités, chaque unité génère une image partielle, la compose (composition Z) avec l'image de l'unité amont et transmet le résultat à l'unité aval. Cependant, trois autres méthodes de composition existent, dans lesquelles la totalité de la scène est répliquée sur chaque unité.

- **Division par échantillons :** cette méthode effectue un rendu avec anticrénelage. Chaque unité génère une sous-image dont tous les points sont décalés d'une distance sous-pixel par rapport aux autres images. Les images individuelles sont combinées pour produire l'image anticrénelée finale. Les cartes graphiques standard utilisées dans une unité DVG effectuent un rendu avec anticrénelage 2x sans baisse de performance : ainsi 2 unités chaînées permettent de rendre avec un anticrénelage 4x sans perte de performance, ou 8x avec une baisse de moitié des performances. La chaîne peut comprendre jusqu'à 4 unités dans ce mode, pour une meilleure qualité (plus d'échantillons) ou de meilleures performances.
- **Division temporelle :** chaque unité génère 60/N images/s, avec une fréquence vidéo de 60 Hz et N nœuds. Chaque unité a donc N/60 s (la durée de N trames vidéo) pour rendre son image et peut rendre une scène environ N fois plus complexe. A un temps donné chaque unité rend la scène dans un état différent. Le résultat final est un flux vidéo à 60 Hz, mais ce mode induit une latence de N-1 trames vidéo.
- **Division de l'espace-écran :** chaque unité effectue le rendu d'une sous-partie de l'image finale. Comme chaque unité rend la totalité de la scène, ce mode est adapté aux applications où les performances sont limitées par la vitesse de remplissage (l'étape géométrique du rendu n'est pas limitante). Des algorithmes d'équilibrage de charge permettent d'améliorer les performances, notamment par division de quadrant qui utilise une méthode d'élagage (« culling ») pour améliorer les performances géométriques.

3.5.4. Sv6

[HPC01] décrit le système sv6, un système de visualisation parallèle à base d'une grappe de stations de travail. Ces stations de travail utilisent une architecture propriétaire (notamment leur carte graphique et processeur central) et n'appartiennent donc pas à la catégorie des composants COTS. Un système sv6 à 4 nœuds de rendu est composé de :

- Un serveur d'application (station HP j6700), qui exécute les applications utilisateur basées sur la bibliothèque OpenGL (OGL).
- Un réseau d'interconnexion reliant le serveur aux 4 nœuds de rendu.

- 4 nœuds de rendu HP j6700 avec carte graphique HP fx10 qui produisent une image sur sortie DVI.
- Un compositeur DVI qui traite les résultats des sorties DVI des nœuds de rendu et fournit en sortie un fluxvidéo; connecté à un système d'affichage.

Un système à 4 nœuds de rendu fonctionne comme suit : le serveur d'application exécute une application utilisateur OpenGL. Il transmet les commandes OGL via le réseau aux 4 nœuds de rendu, qui à leur tour rendent leur image à partir des commandes reçues. Chaque nœud de rendu rend 1/4 de l'image totale, à partir de sa copie intégrale de la scène à rendre. Chacune des sous-images réside dans le tampon de trame du nœud qui la rend. Elles sont composées pour donner l'image finale. Les sorties DVI des 4 cartes graphiques sont reliées aux entrées DVI du compositeur, qui génère en sortie un flux vidéo unique, correspondant à une image continue générée à partir des 4 sous-images (disposées en pavage). Un des points particuliers du système est la présence de plusieurs modes de fonctionnement :

- **Mode performance :** c'est le mode de rendu parallèle décrit ci-dessus. Il s'apparente à un rendu parallèle « sort-first », bien que comme chaque nœud stocke l'intégralité de la scène, le tri s'effectue localement à chaque nœud.
- **Mode à haute qualité de rendu :** chaque nœud rend la même image, décalée (dans l'espace de l'image finale) par rapport aux autres d'une distance sous-pixel. Le compositeur génère l'image finale à partir des 4 images. Cette méthode (anticrénelage) permet de générer des images de plus haute qualité, avec 4 échantillons par pixel.
- **Mode intermédiaire :** un mode de rendu permet un compromis entre qualité et performance. L'image est divisée en 2 et 2 nœuds rendent chacun une moitié avec un anticrénelage à 2 échantillons par pixel.

Ce système permet, en utilisant un accélérateur graphique supportant un mode de rendu anticrénelé d'améliorer la qualité de rendu. Ainsi avec des accélérateurs effectuant un rendu anticrénelé x4, il est possible de rendre (en mode « qualité ») avec 16 échantillons par pixel. Le système sv6 est extensible et permet d'utiliser jusqu'à 16 nœuds de rendu, ce qui permet d'augmenter les performances et/ou la qualité de rendu.

3.5.5. Discussion

Nous avons présenté plusieurs types d'architectures et leurs caractéristiques principales. Tout d'abord, nous remarquons que les architectures DVG et sv6 sont similaires. Leurs méthodes de rendu à haute qualité par anticrénelage réparti sont équivalentes. Le compositeur matériel de l'architecture Sv6 est probablement proche d'un compositeur de type Lightning-2 (voir 3.4.2). Cependant, il permet de composer des images décalées d'une distance sous-pixel, ce qui ne semble pas être possible avec L2. En effet, cela demanderait probablement l'utilisation d'un en-tête par pixel, ce qui augmenterait le débit DVI requis et le surcoût de préparation par les applications de rendu des données de contrôle. Nous pensons que ces 2 architectures sont plus adaptées à la visualisation de type réalité virtuelle (simulation de vol, visualisation de prototypes virtuels). En effet, dans ce domaine, la qualité du rendu (anticrénelage de l'image et vitesse de rendu constante) importe plus que la précision des données d'origine, alors qu'en visualisation scientifique la fluidité du rendu et la qualité de l'image sont moins importantes que la résolution de l'image et le nombre de primitives rendues par seconde. De plus, les méthodes de subdivision de travail de ces 2 architectures dupliquent un grand nombre d'opérations : par exemple, dans le mode « subdivision de scène », chaque nœud ne rend qu'une partie de l'image finale. Comme chaque nœud stocke une copie de la scène, les opérations de traitement géométriques sont effectuées sur chacun des nœuds de manière redondante, c'est lors de la troncature (clipping) que sont sélectionnées les primitives devant être tramées. Il n'y a pas de redistribution des primitives à proprement parler.

Selon [HOU02], la qualité d'une configuration matérielle dépend fondamentalement de l'adéquation de ses différents sous-systèmes. En effet, la vitesse de rendu agrégée (en nombre de primitives par seconde) est limitée par la vitesse du composant le plus lent. Ainsi, une grappe utilisant des nœuds de rendu avec des accélérateurs graphiques très rapides mais un système d'interconnexion entre nœud clients et serveurs sous-dimensionné ne sera pas efficace. Suivant la méthode de rendu parallèle, on identifie différents goulots d'étranglement. Pour le rendu SF, ce sont :

- Regroupement des primitives en paquets (limité par le processeur central des nœuds clients).
- Distribution des primitives : limitée par le débit d'interconnexion (bus et réseau).
- Rendu : limité par la vitesse des processeurs graphiques.

Pour le rendu SL, ce sont :

- Rendu : limité par la vitesse des processeurs graphiques.

- Composition : limitée par le débit d'interconnexion (bus et réseau) et la vitesse de lecture/écriture des pixels dans le tampon de trame des cartes graphiques.

Ainsi, dans le cas d'une configuration haut de gamme, le rendu SF est bien équilibré, alors que le rendu SL est déséquilibré au niveau de l'étape de composition (précisément par la lecture/écriture des pixels des cartes graphiques). Dans ce dernier cas, seule une amélioration des performances des composants individuels peut permettre d'équilibrer les performances du rendu SL. La qualité d'un système de rendu distribué ne dépend pas seulement de sa vitesse de rendu (en primitives/s) mais également du système d'affichage. En effet, ce dernier peut être sous-dimensionné par rapport aux capacités de la grappe : une image standard (1280 x 1024) ne permet de rendre qu'un petit nombre de primitives distinctes (au plus 1,3M en supposant qu'une primitive ne contribue qu'à un pixel). Pour des scènes de très grande taille, il est nécessaire de rendre à haute résolution : ainsi le système d'affichage doit pouvoir afficher correctement (sans la sous-échantillonner) une image à haute résolution. Comme les performances des composants individuels progressent pour l'instant de manière ininterrompue, nous pensons que le développement des grappes de rendu prendra son ampleur dans les prochaines années.

Dans la partie suivante, nous recensons différents composants logiciels permettant d'exécuter des programmes de visualisation sur grappe. Nous nous focalisons sur deux catégories de composants : ceux qui permettent d'effectuer du rendu parallèle proprement dit et ceux qui permettent de paralléliser la partie pré-rendu d'une application de visualisation scientifique.

4. Composants logiciels

Dans cette partie, nous décrivons les composants logiciels qui permettent d'exécuter des applications parallèles de visualisation scientifique sur grappe. D'abord, nous introduisons deux grandes catégories de composants logiciels : les systèmes de rendu graphique parallèle (implantations parallèles de bibliothèques graphiques et systèmes de type réalité virtuelle) et les composants applicatifs parallèles de pré-rendu, utilisés pour paralléliser les parties de pré-rendu des applications de visualisation scientifique. Nous décrivons ensuite de manière plus détaillée ces deux grands types de logiciels. Enfin, nous décrivons certains algorithmes utilisés dans les logiciels de rendu parallèle, et qui permettent d'en améliorer les performances.

4.1. Généralités

Tout d'abord nous considérons les composants logiciels du rendu parallèle sur grappe. Pour cela, nous les répartissons en deux catégories distinctes :

- Les bibliothèques graphiques parallèles : ce sont des implantations spécifiques de bibliothèques graphiques standard (telle qu'OpenGL). Une bibliothèque graphique parallèle fonctionne en tant que surcouche de la bibliothèque graphique standard d'un système d'exploitation donné. Elle intercepte les commandes graphiques (appels de fonctions de la bibliothèque graphique) émises par les instances de l'application de visualisation, exécutées par un ou plusieurs nœuds d'application. Elle les redistribue ensuite aux nœuds de rendu adéquats, qui vont alors exécuter la commande correspondante de la bibliothèque graphique native, dont les fonctions sont généralement implantées sur l'accélérateur graphique du nœud de rendu.
- Les systèmes logiciels de réalité virtuelle (RV), qui dupliquent la scène sur tous les nœuds et parallélisent au niveau de la distribution d'événements. Chaque nœud rend sa copie de la scène en fonction de paramètres de point de vue propres qui lui sont transmis.

Ensuite, nous considérons les composants nécessaires au parallélisme des applications de visualisation : nous présentons ainsi pVTK, la version parallèle de la bibliothèque de visualisation scientifique VTK et la manière dont VTK peut s'intégrer à un système de rendu parallèle.

4.2. Les implantations parallèles d'OpenGL

La bibliothèque graphique OpenGL est un standard de rendu graphique dans le monde industriel (logiciels de CAO, de visualisation, etc.). Les implantations parallèles de bibliothèques graphiques que nous allons décrire utilisent ainsi les spécifications d'OpenGL, au lieu d'utiliser une bibliothèque graphique propriétaire. Les avantages d'une telle approche sont multiples :

- Les applications utilisant la bibliothèque graphique standard peuvent s'exécuter sans modification de leur code source sur grappes.
- Il n'est pas nécessaire de développer une nouvelle bibliothèque graphique.
- Les constructeurs d'accélérateurs graphiques fournissent leur propre version d'OpenGL, qui implante généralement la totalité (ou quasi-totalité) de l'API OpenGL. On peut ainsi exploiter facilement l'accélération du rendu tout en restant indépendant d'un accélérateur particulier.

A priori, on peut développer une surcouche parallèle pour n'importe quelle bibliothèque graphique, cependant comme OpenGL est un standard de fait, elle est donc un choix évident pour une telle parallélisation. Il existe une autre bibliothèque, Direct3D, mais celle-ci est utilisée principalement dans le développement de jeux pour PC sous Windows. De plus le standard OpenGL est ouvert, des modifications peuvent donc être intégrées (par exemple pour le parallélisme de l'API), alors que les spécifications d'une API comme Direct3D sont contrôlées par son développeur (Microsoft), ce qui la rend moins ouverte. Nous présentons maintenant plusieurs implantations parallèles d'OpenGL.

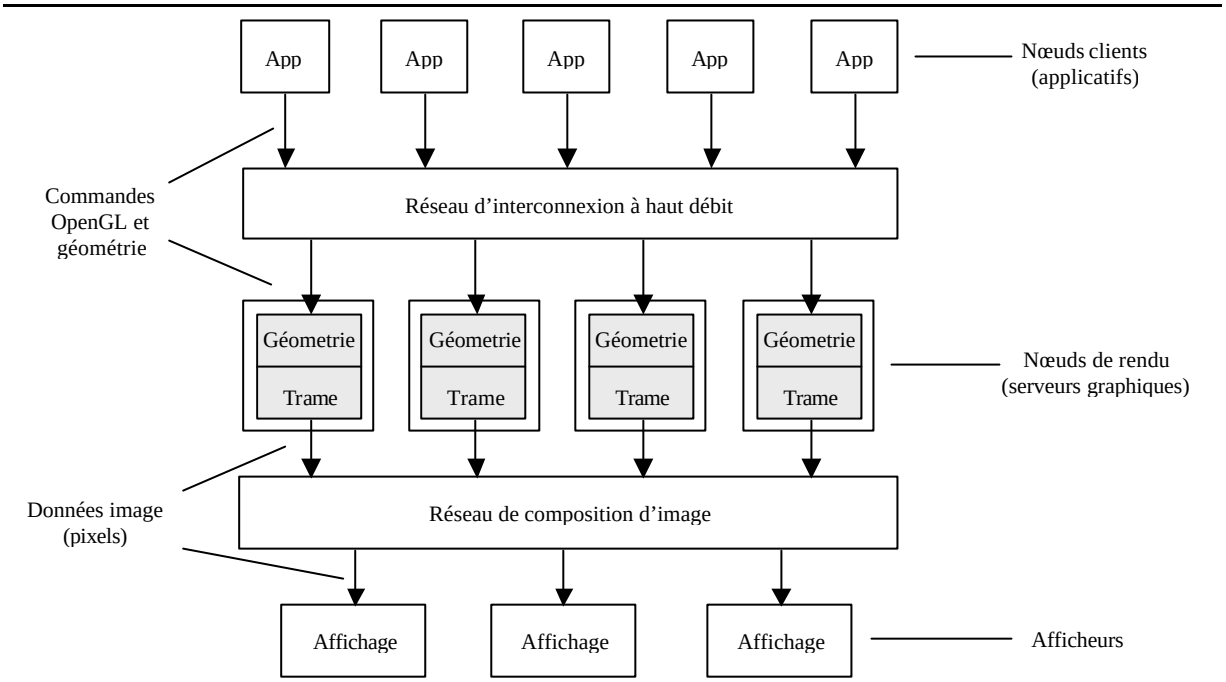
4.2.1. WireGL

4.2.1.1. Présentation

WireGL [HUM01] est un système logiciel extensible de rendu interactif sur grappes de PC. WireGL (WGL), qui fournit l'API OpenGL à chaque nœud d'un cluster, permet d'intégrer plusieurs accélérateurs graphiques dans un

système de rendu parallèle distribué de type « sort-first ». Un système WireGL utilise un ou plusieurs clients envoyant des commandes OpenGL à un ou plusieurs serveurs graphiques (« pipeservers »). Chaque serveur graphique gère une portion contiguë de l'espace-écran. Les serveurs graphiques rendent leur partie de l'image finale. Cette dernière est alors assemblée à partir des images de chaque serveur puis répartie entre les différents systèmes d'affichages. La Figure 13 ci-dessous représente l'architecture d'une grappe utilisant WireGL.

Figure 13 : architecture de WireGL



Un réseau à haut débit relie nœuds clients et nœuds serveurs. WireGL utilise une couche d'abstraction de réseau à base de connexions, ce qui permet d'utiliser différents types de réseaux (TCP/IP, Myrinet). Le nombre de nœuds clients, de nœuds serveurs (c'est-à-dire le nombre de zones logiques de subdivision de l'espace-écran) et le nombre de systèmes d'affichage (c'est-à-dire le nombre de flux vidéo indépendants) sont a priori indépendants entre eux, et ne sont fixés que par les besoins de l'application.
Nous décrivons maintenant le fonctionnement des programmes gérant les nœuds clients et les serveurs graphiques.

4.2.1.2. Fonctionnement des nœuds clients

Dans sa première version [HUM00], WireGL ne permettait pas de paralléliser l'étape d'application d'un programme de visualisation, seul le rendu pouvait être parallèle (avec la méthode sort-first). Pour les applications dont les performances sont limitées par l'interface hôte-accélérateur graphique (voir 2.3) l'utilisation de plusieurs nœuds client permet d'exploiter de manière plus efficace les ressources de plusieurs serveurs graphiques. Comme nous l'avons dit en 2.2.2.2, le surcoût en communication de la méthode de rendu « sort-first » est proportionnel au nombre de primitives redistribuées après pré-transformation. Pour diminuer le débit de redistribution, plusieurs méthodes sont utilisées. Dans un système WireGL, les nœuds clients implantent l'algorithme de distribution « sort-first » : WGL utilise un protocole réseau en flux pour transmettre les données. Les données et commandes OpenGL (représentées par un code) sont alignés et empaquetées séparément dans un même tampon. WGL n'envoie à chaque serveur que le minimum de commandes requises pour un rendu OpenGL correct. Pour chaque serveur, WGL effectue un tri des données géométriques en calculant la zone de chevauchement entre la projection de la boîte englobante 3D des données et la portion d'affichage d'un serveur. Les primitives et commandes sont envoyées lorsque le tampon est plein. La localité des primitives (en général) en visualisation

rend cet algorithme efficace. Cependant la proportion de primitives envoyées plus d'une fois augmente avec le nombre de portions d'affichage (chevauchement), ce qui limite l'extensibilité du système. Comme OpenGL est une machine à état, WGL utilise un système de suivi d'état pour minimiser le débit de commandes de changement d'état. Les états des clients et des serveurs (représentés de manière hiérarchique) sont suivis par WGL, qui calcule pour chaque client la différence d'état avec chacun des serveurs. Lors de l'envoi d'un flux géométrique à plusieurs serveurs, cette différence est envoyée à chaque serveur concerné afin de les synchroniser à l'application. Puis les données sont transmises.
Pour une application parallèle, chaque nœud client envoie des commandes et données à plusieurs serveurs. WireGL fournit donc une interface OpenGL parallèle : des commandes particulières (barrières et sémaphores), émises par l'application parallèle, permettent d'imposer aux serveurs graphiques un ordre de traitement des commandes OGL sans bloquer l'application. Cela permet de faire respecter la sémantique OpenGL par les serveurs. Cependant, tous les serveurs doivent recevoir la même commande d'ordre.

4.2.1.3. Fonctionnement des serveurs graphiques

Pour chaque client connecté (auquel est associé un contexte OGL), un serveur stocke une file de commandes en attente. Ces files de commandes sont placées dans une file circulaire d'exécution de contextes. Un serveur exécute une file particulière jusqu'à ce qu'il n'y ait plus de commandes ou que la file devienne bloquante (barrière ou sémaphore). Dans ce dernier cas, la file est placée dans une file d'attente. A chaque changement de file, un changement de contexte graphique doit être effectué. Comme la fréquence de changement matériel de contexte est limitée (10000 fois/s), WGL utilise l'algorithme de suivi d'état pour l'augmenter. L'algorithme de représentation hiérarchique permet de ne modifier que les variables d'états nécessaires pour passer d'un contexte à l'autre (en général cohérents du fait du parallélisme). Il est ainsi possible de changer de contexte 200000 fois/s (pour des différences de matrice de transformation et de couleur courante) et jusqu'à 5 millions de fois/s pour des contextes identiques. Un algorithme d'ordonnancement de type « round-robin » est utilisé.

4.2.1.4. Assemblage de l'image finale.

Une méthode matérielle d'assemblage utilise le compositeur Lightning-2, décrit en 3.4.2.1, qui prend en entrée les sorties vidéo (DVI) des serveurs et assemble l'image finale. Une méthode logicielle utilise un serveur de visualisation relié aux serveurs de rendu par réseau (dédié ou partie du réseau clients-serveurs). Ce serveur reçoit les sous-images extraites par chaque serveur graphique et les assemble dans une image finale. Cette méthode a cependant un surcoût plus important que la méthode matérielle.

4.2.1.5. Performances

D'après les auteurs, les performances de WireGL sont extensibles jusqu'à une configuration de 16 clients et 16 serveurs. Ils concluent qu'un bon dimensionnement nécessite une reconstruction matérielle de l'image finale, et que l'utilisation d'un très grand nombre de nœuds (128) nécessite l'augmentation du débit réseau.

Nous décrivons maintenant Chromium, un système logiciel de rendu distribué similaire à WireGL.

4.2.2. Chromium

4.2.2.1. Présentation

Le système logiciel Chromium (CR) a été développé à la suite de WireGL, en partie par les mêmes auteurs [HUM02]. Chromium est un système logiciel de manipulation de flux de commandes d'API graphique sur grappes de stations de travail. La différence fondamentale de Chromium par rapport à WireGL est qu'il fournit un mécanisme de traitement plutôt qu'un algorithme particulier (par exemple le rendu parallèle « sort-first »). Alors que l'architecture de WireGL implante uniquement l'algorithme de rendu parallèle « sort-first », les filtres de flux de Chromium peuvent être ordonnés pour mettre en oeuvre les architectures de rendu « sort-first » et « sort-last », en utilisant comme WireGL des accélérateurs graphiques à faible coût.

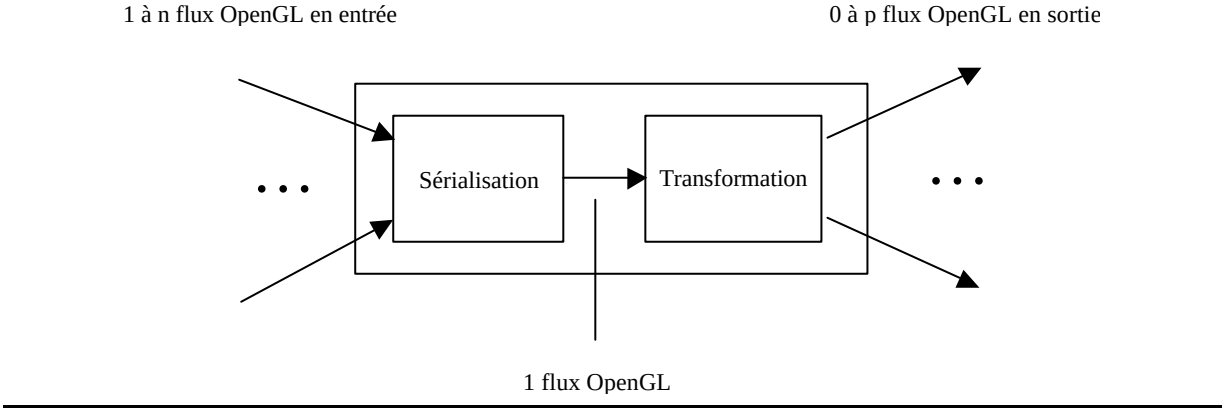
4.2.2.2. Architecture du système

Dans WGL, le protocole utilisé pour transférer les images des serveurs au compositeur est le même que le protocole réseau OpenGL utilisé pour la transmission de la géométrie des clients aux serveurs, ce qui a été repris dans Chromium. Chromium décrit l'architecture du système de rendu par graphe orienté acyclique (Direct Acyclic Graph ou DAG). Chaque nœud peut avoir des flux de données en entrée ou sortie et effectue des traitements dessus au moyen de SPU (« Stream Processing Unit »). Un nœud comprend 2 parties logiques distinctes : une partie de transformation et une partie de sérialisation. La partie de transformation d'un nœud prend en entrée un flux de commandes OpenGL et produit en sortie de 0 à plusieurs flux de commandes OpenGL. La correspondance entre flux d'entrée et de sortie est arbitraire. Les flux de sortie sont envoyés à travers le réseau vers d'autres nœuds. Le composant de sérialisation d'un nœud prend en entrée un ou plusieurs flux OpenGL indépendants (chacun ayant son propre contexte graphique), et produit un flux OpenGL unique en sortie. L'ordre de traitement de flux en entrée est contrôlé par des primitives de synchronisation (barrières et sémaphores), dont les commandes correspondantes sont incluses dans le flux lui-même. Quand un flux est bloqué par une primitive (exemple une barrière), le sérialiseur effectue un changement de contexte graphique et traite un autre flux. Un algorithme de suivi d'état analogue à celui de WireGL permet de minimiser le coût de changement de contexte. Le composant de sérialisation d'un nœud peut être implanté de deux manières différentes, ce qui définit 2 types de nœuds.

- Nœuds serveurs : ce sont les nœuds qui ont 1 ou plusieurs arêtes entrantes. C'est le serveur Chromium qui effectue leur gestion (un processus par nœud). Un nœud serveur gère plusieurs connexions réseau, chaque connexion transmettant un flux OpenGL empaqueté sous forme de messages.
- Nœuds clients : ce sont les nœuds qui n'ont aucune arête entrante et doivent générer eux-mêmes leur propre flux de commandes OpenGL (qui sont donc nécessairement séquentiels). Leur flux provient d'une application OpenGL quelconque : la bibliothèque OpenGL de Chromium injecte dans la partie de transformation les commandes OpenGL de l'application.

La Figure 14 ci-dessous représente un nœud quelconque et ses deux composants (respectivement sérialisation et transformation).

Figure 14 : composants d'un nœud Chromium



4.2.2.3. Traitement par flux

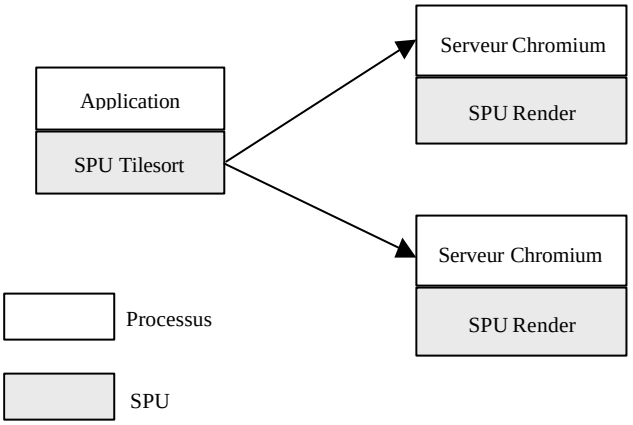
Comme nous l'avons décrit en 4.2.2.2, chaque nœud traite ses flux OpenGL au moyen de SPU. C'est la partie de transformation du nœud qui effectue les traitements. Une SPU fournit une interface OpenGL. Chaque nœud utilise ainsi une ou plusieurs SPU reliées en chaîne de traitement, qui est chargée par le composant de sérialisation à l'initialisation du nœud. Chaque SPU maintient une table de correspondance avec l'unité en aval. Une SPU donnée peut ainsi modifier certaines commandes OpenGL et transmettre à l'unité en aval les autres commandes non modifiées. Cela permet à une SPU particulière de modifier l'état de rendu et d'effectuer ce dernier avec un style différent (par exemple en mode fil de fer).

De plus, chaque SPU dérive d'une SPU parent (par héritage unique) ; une SPU donnée peut donc implanter sa propre version d'une commande OpenGL quelconque et utiliser les fonctions de la SPU parente (la SPU parent la plus fréquemment utilisée étant l'unité « passthrough », qui transmet en aval les commandes OGL sans modification).

4.2.2.4. Outils et SPU fournis

Chromium fournit ainsi 4 bibliothèques encapsulant les opérations de base de traitement des flux (empaquetage et dépaquetage de flux, connexion réseau point-à-point et suivi d'état OpenGL). La Figure 15 ci-dessous représente une configuration de rendu «sort-first», semblable à celle de WireGL, avec une application séquentielle. Le graphe de configuration comporte 3 nœuds. Un nœud client exécute une application OpenGL séquentielle. Il utilise une SPU «tilesort» (qui implante l'algorithme «sort-first»), et transmet ses commandes graphiques et données géométriques aux nœuds serveurs. Chacun des nœuds serveurs rend une partie différente de l'espace d'affichage global. La SPU «render» utilise le flux de commandes reçu pour appeler les fonctions OpenGL correspondantes de l'accélérateur graphique local au nœud serveur.

Figure 15 : configuration Chromium de rendu « sort-first »



Il est important de noter que les nœuds sont logiquement distincts mais peuvent être identiques physiquement (c'est-à-dire que les 2 nœuds serveurs peuvent se trouver sur la même machine, voire les 3 nœuds du graphe). Cependant, chaque nœud est géré par un processus distinct. La même configuration peut être utilisée avec une application parallèle. Dans ce cas, autant de nœuds clients sont utilisés qu'il y a de processus parallèles. Chaque nœud client utilise la SPU «tilesort» pour rendre les primitives de sa partie de la scène (en les transmettant aux serveurs adéquats).

4.2.3. Any-GL

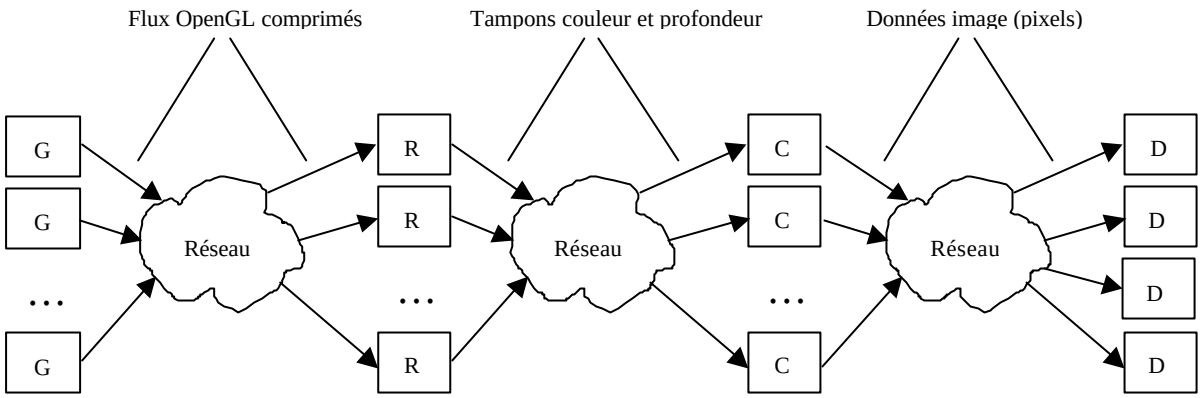
4.2.3.1. Présentation

Any-GL [YAN02] est un système basé sur WireGL (et de fait sur OpenGL). C'est un système logiciel hybride de rendu distribué. Il utilise en effet les méthodes de rendu parallèle sort-first et sort-last à deux endroits différents du pipeline graphique. Il utilise de plus un système de suivi d'état par horodatage et un système de compression de données (géométriques et images).

4.2.3.2. Architecture

Any-GL décrit la configuration de rendu par un graphe, qui comprend 4 types de nœuds logiques : les nœuds de distribution de données géométriques (G), rendu (R), composition d'image (C), et nœuds d'affichage d'image (D). Any-GL utilise un pipeline dont les 4 étages sont chacun constitués d'une des 4 catégories de nœuds décrites. La Figure 16 ci-dessous représente l'architecture d'un système Any-GL.

Figure 16 : architecture d'AnyGL



Nous décrivons brièvement les 4 catégories de nœuds.

- Nœuds G: ils transmettent les commandes OpenGL aux nœuds R. Les commandes spécifiant des primitives sont empaquetées dans 2 tampons (un pour les codes des commandes et un pour leurs arguments). Le même algorithme d'intersection de boîte englobante que WireGL permet de déterminer quels nœuds R sont concernés par le paquet (parallélisme de type sort-first).
- Nœuds R : ces nœuds exécutent directement les commandes de l'API OpenGL (ce qui permet d'utiliser un rendu matériel) suivant les commandes reçues à partir des nœuds G. Les commandes de changement d'état sont traitées par un système de suivi d'état. L'image rendue (couleurs et profondeur) est lue par la commande OGL « readPixels » puis transmise aux nœuds C.
- Nœuds C : les nœuds C utilisent une architecture de type sort-last. Chaque nœud C traite une partie de l'image finale et stocke un tampon z et couleur. Ils composent les images (venant des nœuds G) reçues dans le tampon couleur suivant la fonction de comparaison z du contexte graphique courant. Après composition et basculement de tampon, le nœud transmet son image aux nœuds D.
- Nœuds D: Un nœud D rassemble de manière logicielle les tampons couleurs des nœuds D et C et les affiche sur écran.

4.2.3.3. Système de suivi d'état

AGL effectue un suivi des nœuds G et R par horodatage logique. A chaque valeur d'état correspond une date logique (date de l'appel de la dernière commande modifiant l'état). Les contextes graphiques des nœuds et dates logiques sont organisés sous forme d'arbre pour comparaison rapide. Chaque nœud G maintient N+1 contextes graphiques virtuels (N nœuds R connectés au nœud G) : le contexte du nœud G (dit applicatif) et les contextes courant du nœud sur chacun des N nœuds R. Lors de modification d'état, la date et l'état du contexte applicatif sont mis à jour. Puis une différence est effectuée entre contexte (dit « actif ») du nœud de destination et contexte d'application. En cas de différence de dates logiques, la différence est traduite en commandes OGL d'état placées dans le tampon de commandes. Le tampon est alors transmis au nœud. Les contextes sont synchronisés en copiant le contexte applicatif dans le contexte actif (si leurs dates diffèrent).
Le suivi d'état sur les nœuds R est similaire à celui des nœuds G. Chaque nœud R maintient 1 contexte « applicatif » (reflétant celui du matériel) et 1 contexte virtuel pour chaque nœud G auquel il est relié. Le contexte correspondant au nœud G dont le nœud reçoit un paquet est dit actif. Un changement de contexte a lieu quand le nœud reçoit un paquet dont la source diffère de la source du paquet précédent. Le mécanisme de changement est similaire à celui de la différence de contextes des nœuds G, excepté qu'ici les commandes de l'API GL sont appelées en cas d'inégalité de dates pour chaque variable entre les deux contextes. Une synchronisation des deux contextes est ensuite faite (copie du contexte d'application dans le contexte actif).

4.2.3.4. Compression de données et partage de ressources graphiques

Any-GL implante des méthodes particulières pour réduire le débit de chacune des 3 étapes d'interconnexion et optimiser l'utilisation des ressources des différents nœuds.

- Codes des commandes : ils sont compressés avec l'algorithme LZW, ce qui permet un rapport de compression de l'ordre de 1/4.
- Compression des données géométriques : les valeurs des sommets (primitives) sont compressées de manière adaptative : 4 méthodes différentes de prédiction sont utilisées en fonction du type de primitive OGL (triangle, bande de triangles, etc.). Cette méthode tend à réduire la précision des données géométriques.
- Compression des données images : elles sont compressées par un algorithme temps réel sans perte (de type Huffman) pour un taux de compression entre 1/2 et 1/6.
- Partage global des textures et listes d'affichage : Any-GL utilise une méthode de partage global des textures et listes d'affichage pour empêcher l'explosion de la place mémoire consommée sur les nœuds R dans le cas d'applications parallèles (problème non résolu par WireGL). Un paramètre permet de définir une texture comme partagée globalement. Chaque nœud G peut séparément définir des textures globales. Quand un nœud R reçoit une texture partagée, et que celle-ci existe déjà sur le nœud (même ID), le nœud relie cette texture à l'emplacement mémoire de la texture globale déjà existante. Une méthode similaire est utilisée pour les listes d'affichage.

4.2.3.5. Performances

- Suivi d'état : selon des mesures effectuées par ses auteurs sur plusieurs applications, le système de suivi d'état d'Any-GL est aussi performant que celui de WireGL. Any-GL peut effectuer environ 500000 changements de contexte par seconde.
- Extensibilité des performances : les performances d'AnyGL ont été mesurées sur un simulateur en fonction du total de nœuds G et R (de 2 à 32). Les auteurs montrent que les performances (images rendues/s) sont relativement linéaires en fonction du nombre de nœuds, pour des applications de rendu surfacique de modèles de jusqu'à 1M de primitives.

Cependant, nous pensons que le réseau choisi pour les simulations (Ethernet 100 Mbit/s commuté) limite l'extensibilité du système. L'utilisation d'un réseau haut débit (type Gbit) paraît indispensable.

4.2.4. Comparaison des implantations parallèles d'OpenGL

Actuellement, seuls Chromium et WireGL ont pu être mis en œuvre sur des configurations matérielles, AnyGL n'existant qu'à l'état de simulateur. Chromium et WireGL dérivent d'une architecture similaire : abstraction de couche réseau, bibliothèque de suivi d'état, changement de contexte logiciel, utilisation de 2 types de nœuds (clients et serveurs), existence d'une extension parallèle de l'API OpenGL (barrières, sémaphores), etc. Cependant Chromium introduit la notion de traitement de flux par chaînes d'unités dédiées. Cela permet à Chromium d'être plus général que WireGL, qui n'implante que le rendu parallèle sort-first. Contrairement à WGL et CR, Any-GL comprend plus de types logiques de nœuds. Cela permet a priori une plus grande souplesse dans la configuration. Cependant, AGL utilise une architecture fixe, comme WGL, contrairement à Chromium qui peut implanter plusieurs architectures et plusieurs variantes (par exemple le rendu «binary-swap », qui est une méthode de rendu volumique de type sort-last) grâce à sa grande flexibilité de configuration. Les 3 architectures utilisent une méthode de partitionnement statique (pour le rendu « sort-first »). Une différence notable entre AGL et WGL/CR est l'utilisation, par le premier, de protocoles de compression (des données géométriques et des données image), ce qui permet potentiellement une meilleure extensibilité des performances. La méthode de partage global des textures (non implantée dans WGL/CR) permet de limiter l'explosion de la taille mémoire de texture pour les grappes de grande taille (> 32 nœuds), ce qui peut limiter potentiellement l'extensibilité d'un système. Enfin, AGL utilise une méthode logicielle de composition : les images sont extraites des accélérateurs des différents nœuds R (cette seule opération est souvent coûteuse en temps) et composées logiciellement. Cela induit une latence importante (50-100 ms d'après les auteurs), contrairement à WGL et CR qui ont été testés dans des configurations avec composition matérielle (respectivement avec L2 et le SGE, voir 3.4.2) qui n'induisent généralement qu'une latence de la durée d'une seule image.

Les implantations OpenGL permettent d'exécuter a priori n'importe quelle application OpenGL de base. Cependant, une application séquentielle ne peut exploiter les ressources que d'un certain nombre de nœuds de rendu. Au-delà, les performances sont limitées par la vitesse à laquelle l'application peut envoyer des commandes graphiques (le facteur limitant les performances est l'interface hôte-carte graphique) et l'efficacité du rendu décroît quand le nombre de nœuds augmente. Pour cela, il est nécessaire d'utiliser une application parallèle avec

les API fournies par les systèmes de rendu (cas de Chromium). La partie de pré-rendu (différente de la partie graphique, qui appelle des fonctions d'une API graphique) d'une application peut être parallélisée grâce à divers composants logiciels. Nous décrivons maintenant quels sont ces composants.

4.3. Composants applicatifs parallèles

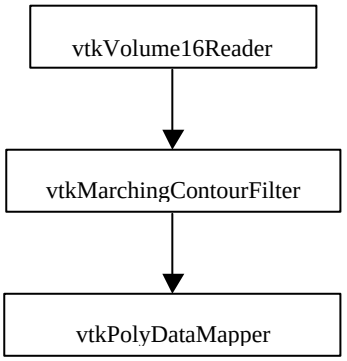
4.3.1. Parallel VTK

pVTK (« parallel VTK ») [AHR00] est une extension parallèle de VTK (« the Visualization Toolkit »). VTK [SCHR98] est une bibliothèque orientée objet de visualisation scientifique qui fournit une interface C++ (ainsi que pour d'autres langages interprétés comme Tcl ou Python) et implante de nombreux algorithmes de visualisation scientifique et de traitement de données (traitement de données 3D, traitement d'image 2D, rendu volumique, etc.).

4.3.1.1. Description de VTK

VTK est un système de type « data-flow ». Le processus de visualisation utilisé par VTK est le suivant : les données brutes, introduites en début de pipeline, sont transformées et traitées par celui-ci de manière à donner une représentation compréhensible par un observateur (c'est-à-dire une image rendue à partir des données). Le pipeline est constitué de plusieurs étages : chaque étage reçoit une forme intermédiaire des données à partir de l'étage amont, effectue dessus un traitement et les transmet à l'étage aval. Contrairement au pipeline graphique, qui est de type « feed-forward », le pipeline de visualisation de VTK est de type « demand-driven » : l'étage le plus aval (c'est-à-dire celui qui effectue le rendu graphique des données) requiert de l'étage immédiatement amont des données. Cette demande se retropropage le long du pipeline, jusqu'à l'étage initial (le plus amont), qui généralement effectue la lecture des données à partir de fichiers de stockage. Les données sont alors transmises à l'étage suivant et sont transmises vers l'aval pour subir le processus de traitement à travers tout le pipeline jusqu'au rendu final. La Figure 17 ci-dessous montre le pipeline VTK d'une application d'extraction d'isosurface.

Figure 17 : pipeline d'une application VTK de rendu surfacique



A chaque étape correspond un module de VTK. Le module vtkVolume16Reader effectue la lecture d'un volume de données à partir de fichiers binaires, vtkMarchingContourFilter traite les données (reçues de l'étape de lecture) avec un algorithme d'extraction de contour (surface 2D), vtkPolyDataMapper effectue le rendu de primitives graphiques triangulaires à partir des données du contour généré. Cependant l'exécution du pipeline de VTK est totalement séquentielle. Afin de visualiser dans de bonnes conditions de gros volumes de données, l'utilisation du parallélisme dans le pipeline de visualisation est nécessaire.

4.3.1.2. Description de Parallel VTK

Selon [AHR00], un système de visualisation parallèle doit, pour être efficace, avoir les propriétés suivantes : extensibilité, portabilité, fonctionnabilité, et complexité apparente réduite. Dans les systèmes de type « data-flow », on distingue 3 types de parallélisme : de tâche, pipeline, et de données. La plupart des systèmes existants

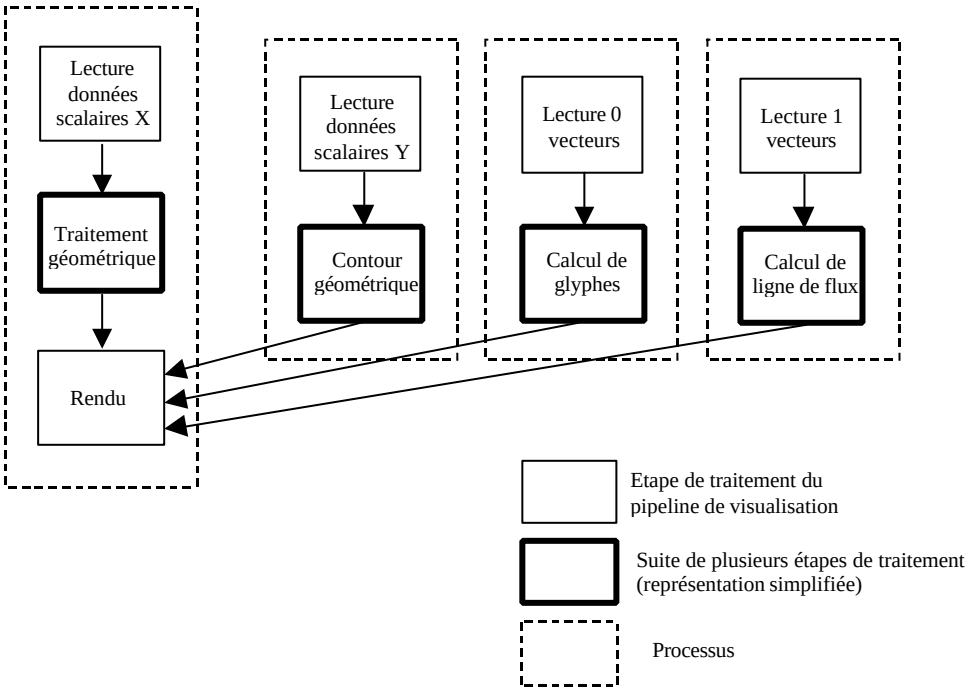
utilisent un mécanisme central pour contrôler les différents processus parallèles, mais cette solution est cependant peu extensible. pVTK utilise une autre solution et implante les 3 types de parallélisme sur multiprocesseurs à mémoire partagée et distribuée.
Comme nous l'avons dit, VTK utilise des modules instanciés reliés en pipeline. Ces modules s'exécutent automatiquement quand une requête leur est faite. Cela requiert : le partage des données entre modules, la persistance d'un programme, la possibilité d'exécuter des méthodes de module. L'ajout d'objets « processus système » à VTK permet d'abstraire le modèle de mémoire utilisé (partagée ou distribuée). Le partage des données est effectué par communications entre processus. Deux implantations distinctes des communications existent selon le modèle de mémoire : avec la bibliothèque de passage de message MPI pour le modèle à mémoire distribuée, et avec des mécanismes de synchronisation et de copie pour le modèle à mémoire partagée.
Les méthodes de chaque processus peuvent être invoquées par le processus système. Cela permet également de satisfaire le fonctionnement « par-demande » de VTK. Des objets « port » permettent de connecter des modules inter-processus.

4.3.1.3. Les 3 modèles de parallélisme de pVTK

PVTK utilise trois modèles différents de parallélisme : tâche, pipelines et données.

a) Parallélisme de tâche
Le parallélisme de tâche est utilisé quand des modules indépendants s'exécutent dans des processus différents. Ainsi, dans la Figure 18 ci-dessous, 4 processus lisent des données et effectuent dessus des traitements indépendants. Les résultats sont synchronisés par le module de rendu du premier processus : les calculs des processus sont effectués de manière asynchrone et la synchronisation est effectuée par un objet port au niveau du module de rendu.

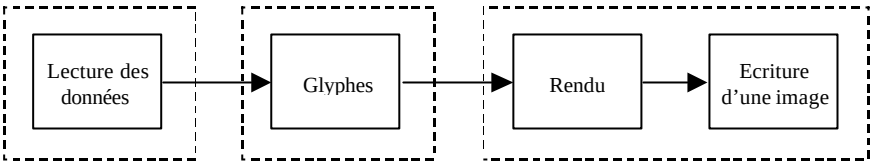
Figure 18 : parallélisme de tâche



b) Parallélisme de pipeline
Le parallélisme de pipeline permet de traiter un jeu de données avec des ressources de calcul indépendantes. Chaque étape est ainsi exécutée par une ressource indépendante. Le jeu de données doit pouvoir être découpé en

sous-ensembles indépendants, ou bien être défini en fonction du temps. Dans la Figure 19 ci-dessous, la lecture des données, le calcul de glyphes, et le rendu/écriture de l'image rendue sont effectuées en parallèle par trois processus différents. Ainsi, à une étape de temps donnée k, la partie k du jeu de données est rendue, la partie k+1 est traitée par un algorithme de glyphes et la partie k+2 est lue sur disque.

Figure 19 : parallélisme de pipeline

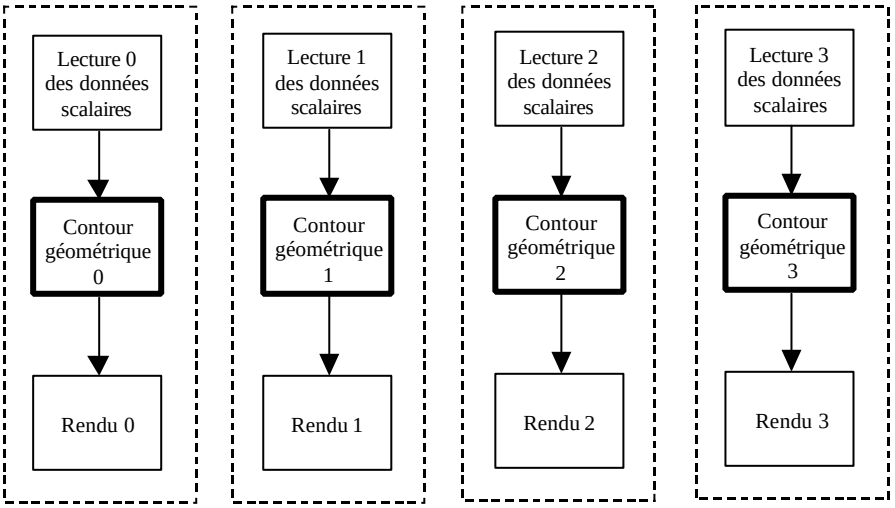


c) Parallélisme de données

Le parallélisme de données est utilisé pour traiter des jeux de données de taille importante. L'ensemble des données est découpé en N sous-ensembles indépendants, traités en parallèle par N processus. La Figure 20 ci-dessous représente un programme utilisant le parallélisme de données. Quatre processus effectuent le même traitement sur une sous-partie de l'ensemble complet de données. Ce traitement (décrit de manière simplifiée) comprend la lecture du sous-ensemble des données, l'application d'un algorithme de calcul d'un contour géométrique, et le rendu des données de contour produites. La partition des données en sous-ensembles est effectuée par le modèle de transmission continue de données de VTK (« streaming data model »). Ce modèle a les propriétés fondamentales suivantes :

- Séparabilité des données : la capacité à découper un jeu de données en sous-ensembles indépendants.
- Correspondance entre données d'entrée et de sortie : c'est la capacité de déterminer quelle partie des données en entrée est requise pour générer une partie demandée des données de sortie.
- Invariance de résultat : les données de sortie (résultats) ne doivent pas dépendre de la méthode de partition et de sa granularité.

Figure 20 : parallélisme de données



L'utilisation de ces différents modes de parallélisme permet d'améliorer fortement les performances du pipeline de visualisation (pré-rendu). [AHR00] présente les résultats d'un programme d'extraction d'isosurface utilisant le parallélisme de données (utilisant un modèle de traitement proche de la Figure 20).

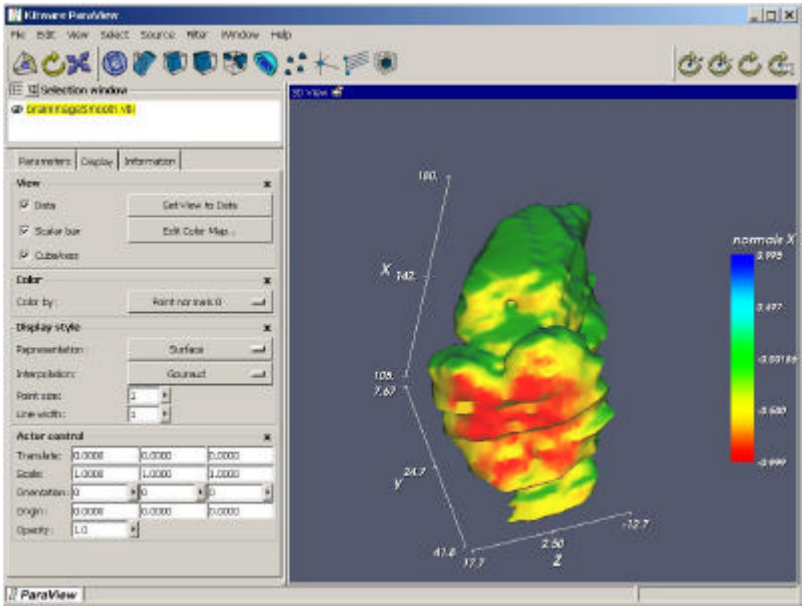
4.3.1.4. Extension du modèle de VTK

Les différentes méthodes de parallélisme que nous avons vues en 4.3.1.3 permettent de paralléliser la partie applicative d'une application de visualisation. Cependant, les jeux de données de très grande taille posent problème, compte tenu des ressources limitées d'une grappe de PC. [AHR01] présente une méthode de pré-rendu de jeux de données de très grande taille utilisant la transmission parallèle continue (« parallel data streaming »). Cette méthode permet de visualiser des données ne tenant pas en mémoire ou sur « swap ». Elle permet également de réduire l'empreinte mémoire de la visualisation, ce qui augmente l'efficacité du cache et réduit l'utilisation du stockage temporaire. Pour cela, l'application de visualisation doit découper le jeu de données en morceaux, qui doivent correctement être traités. Le jeu de données et les algorithmes le traitant doivent avoir les propriétés suivantes (mentionnées en 4.3.1.2) : séparabilité, correspondance entrée/sortie et invariance du résultat. Comme nous l'avons vu en 4.3.1.2, le traitement parallèle requiert les mêmes caractéristiques que le traitement continu (par exemple séparabilité et invariance de résultat) et requiert de plus l'exécution et transfert de données asynchrones. Une méthode de déclenchement de la mise à jour des différents processus permet le calcul en parallèle.

4.3.1.5. Paraview

Plus récemment, en 2002, VTK a été doté d'une interface visuelle : Paraview [PAR03], conçue dans ses grandes lignes par James P. Ahrens et développée par Kitware. Ce logiciel complète VTK en rendant accessible la plupart de ses modules en mode interactif (voir Figure 21), pour construire assez intuitivement des applications que l'on peut sauver et rejouer. En ce sens VTK se rapproche des environnements « data flow » classiques tels AVS ou OpenDX, en se dotant lui aussi d'un « éditeur visuel de programmes ».

Figure 21 : interface de programmation visuelle de VTK : Paraview



Paraview doit à notre sens surtout être employé pour ses vertus pédagogiques (initiation à VTK) et pour prototyper des portions d'applications. En effet, Paraview sélectionne automatiquement, pour un type de traitement donné, la classe VTK la plus adaptée, et optimise directement le pipeline, éventuellement parallèle (le faire «à la main » demande une certaine expertise VTK). En contrepartie, sitôt que l'on dépasse quelques fonctions de visualisation combinées, gérer une application complète sous Paraview devient épineux. Donc produire des branches précises d'application dans Paraview, les récupérer sous forme de script générés automatiquement, et les intégrer dans une application complexe gérée comme un développement classique (en C++, python…), nous semble une bonne approche.

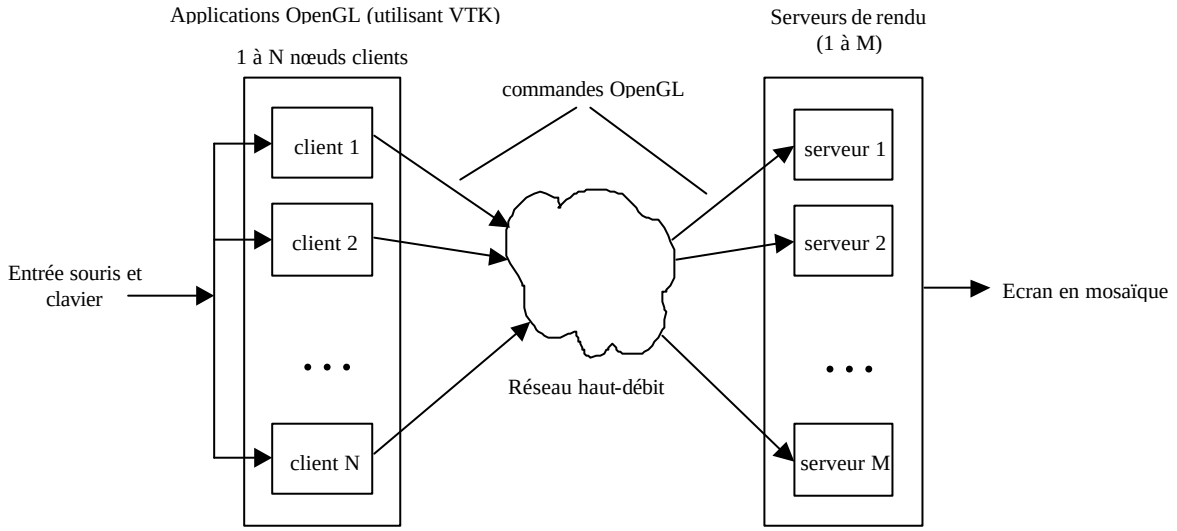
4.3.2. Utilisation de composants de rendu parallèle avec VTK

L'utilisation de composants applicatifs parallèles (comme pour pVTK) permet d'accélérer grandement l'étape applicative d'une application de visualisation. Cependant, à partir d'un certain niveau de parallélisation (nombre de nœuds de la grappe) dédiées à la partie applicative, si le rendu est séquentiel alors il devient l'étape limitante. La vitesse de l'application de visualisation est alors limitée par l'étape du rendu. Pour supprimer ce goulot, il est nécessaire de rendre le rendu parallèle. Cependant, comme l'interface entre l'application parallèle et le rendu parallèle reste séquentielle, il est nécessaire d'appliquer à l'application le modèle de l'API de rendu parallèle fournie par les systèmes de rendu parallèle tels que Chromium. [TOM02] présente une méthode d'application du modèle de parallélisme de Chromium (MPC) à la VTK. Pour exécuter une application parallèle de visualisation sur grappe utilisant Chromium et VTK, les auteurs effectuent deux modifications importantes :

- Modifications des méthodes de rendu de VTK : l'API OpenGL parallèle de CR est utilisée pour permettre aux applications de rendre leurs données en parallèle.
- Implantation d'une interface graphique parallèle ParaMouse : cela requiert d'ajouter à VTK un nouveau composant d'interaction.

La Figure 22 ci-dessous représente le système de visualisation sur grappe utilisant VTK et CR.

Figure 22 : application VTK et rendu Chromium



L'interface graphique parallèle ParaMouse récupère les événements utilisateur (souris et clavier) de manière séquentielle et les diffuse aux applications OpenGL (qui utilisent VTK pour les traitements autre que les appels de rendu OGL). Chaque application traite et rend une partie différente de la même scène. Les applications intercommuniquent avec MPI. Notons cependant que dans ce cas précis c'est VTK qui est utilisé et non pas pVTK. Comme chaque nœud rend une partie de la scène, VTK est utilisé pour traiter la sous-partie de la scène (et non pVTK). Dans ce cas précis, les 3 propriétés décrites en 4.3.1.3 qui permettent un traitement parallèle correct (séparabilité, correspondance et invariance) peuvent ne pas être satisfaites.

4.3.3. Autres logiciels parallèles de visualisation scientifique

Nous ne pouvons limiter à VTK l'évocation des logiciels ou plate-formes de visualisation scientifique parallèle. Nous citons ci-dessous très rapidement quelques-uns des plus connus et des plus répandus de ces logiciels.

- AVS : www.avs.com

Exemple typique d'atelier de développement avec modèle d'exécution par flots de données, programmable par éditeur visuel interactif, AVS, logiciel propriétaire et commercial, a ces dernières années perdu de son « leadership » sur le marché scientifique et technique de la visualisation. AVS propose des extensions parallèles pour le rendu « multi-pipe », modérément portables, et ne semble pas aborder activement la problématique de la visualisation (pré-rendu surtout) sur grappes de PC.

- OpenDX : www.opendx.org

A l'origine produit IBM, DX (Data Explorer) a été versé en « open source » depuis plusieurs années sous le nom d'OpenDX. Egalement « data flow » (exécution par flots de données) avec éditeur visuel de programmes, OpenDX propose un traitement très général du parallélisme de type SMP (mémoire unifiée). Une version « mémoire distribuée » d'OpenDX existe, plutôt expérimentale, et semble-t'il limitée dans son avenir par l'absence de développements actifs dans ce logiciel désormais ancien : OpenDX reste très utilisé et apprécié; le logiciel bénéficie d'une conception élégante et solide mais il ne bénéficie plus d'aucune évolution significative.

- COVISE : www.hlrs.de/organization/vis/covise

Développée par l'équipe « visualisation » du HLRS de Stuttgart, cette autre plate-forme « data flow » avec éditeur visuel de programmes est d'entrée de jeu conçue pour la visualisation et la simulation couplées et distribuées, dans une visée initialement « réseau étendu » et « couplage à gros grains » des modules.
A notre connaissance le sous-système d'affichage COVER de COVISE a fait l'objet d'adaptations sur grappes de PC, mais pas encore le noyau général. COVER possède des capacités « réalité virtuelle » avancées : stéréoscopie, affichage multi-écrans, interfaces avec menus 3D etc.

- EnSight : www.ceintl.com

Ce logiciel commercial et propriétaire est devenu un acteur important du marché de la visualisation scientifique haute performance, soutenu notamment par des contrats du DOE américain (pour usage par les laboratoires gouvernementaux). Après des développements sur le parallélisme SMP et l'accès parallèle aux résultats de calculs par décomposition de domaines, un autre contrat DOE récent, attribué en 2003, soutient l'adaptation d'EnSight sur grappes de PC (Red Hat, Beowulf). Ces développements seront d'abord disponibles dans la version « avancée » dite Gold du logiciel.

- AMIRA : www.tgs.com

Orienté objet et basé sur Open Inventor, ce logiciel, de conception plus récente que les précédents, est originaire du centre de recherche Konrad-Zuse (ZIB, Berlin). La société TGS le co-développe et le distribue. Le parallélisme ne faisait pas partie de la conception initiale d'AMIRA, dont un des points forts est la qualité et l'optimisation (notamment sur PC) des algorithmes graphiques (notamment en rendu volumique). Une première version « cluster » d'AMIRA est à notre connaissance imminente à la date de rédaction de ce rapport.

En conclusion de cette rubrique, nous pensons pouvoir dire que peu de logiciels de visualisation parallèle (« complets » c'est-à-dire aptes à traiter l'ensemble du processus pré-rendu+rendu) sont à la fois ouverts et mûrs pour le parallélisme sur grappe de PC. Ceux qui apparemment évoluent en ce moment le plus activement dans ce sens sont les logiciels les plus propriétaires : AMIRA, EnSight… VTK réalise sans doute à ce jour le meilleur compromis entre ouverture, gratuité, qualité de conception et complétude.

4.4. Autres composants de rendu parallèle

Les composants logiciels que nous avons décrits permettent soit d'accélérer le rendu (implantations parallèles d'OpenGL), soit de fournir des algorithmes de traitement pré-rendu de la visualisation (comme avec pVTK). Cependant, il existe des composants logiciels permettant de paralléliser la visualisation à un niveau différent de celui du parallélisme de type « sort-first » ou « sort-last », qui s'adresse au problème du rendu et de l'affichage.

4.4.1. Le système Net Juggler

Le système Net Juggler [ALL02] est un système logiciel de rendu parallèle qui permet de gérer une grappe de PC de manière unifiée. Il se base sur le système VR Juggler : Net Juggler (NJ) adapte VR Juggler, qui ne fonctionne que sur architecture SMP, aux architectures de grappes de PC.

4.4.1.1. Description de Net Juggler

Net Juggler est une plate-forme logicielle de réalité virtuelle indépendante des systèmes d'affichage et d'entrée/sortie. Elle donne accès aux API graphiques et peut être configurée durant exécution. Elle se compose d'un noyau et de plusieurs gestionnaires de sous-systèmes (entrées, rendu, affichage, configuration, environnement). Le noyau contrôle le système temps réel et gère les communications inter-gestionnaires. NJ parallélise les applications en les dupliquant sur chaque nœud avec des paramètres locaux (comme le point de vue). Comme un matériel d'entrée n'est relié qu'à un nœud donné, NJ diffuse les événements d'entrée à tous les nœuds avec des composants logiciels intermédiaires (« proxies ») client et serveur, qui abstraient les pilotes des matériels d'entrée et reçoivent les données émises par ces matériels. Les communications inter-nœuds se font par flux et ont lieu une seule fois par image, chaque nœud transmettant son tampon de messages vers tous les autres nœuds. NJ utilise un réseau Gbit ou un réseau Fast Ethernet associé à un réseau de synchronisation rapide. Les routines de communication utilisent MPI, et diverses implantations ont été faites en fonction du protocole réseau utilisé.

4.4.1.2. Synchronisation logicielle de l'affichage

Les cartes graphiques standard pour PC ne gèrent généralement pas le swaplocking et genlocking (voir 3.1.2 à propos du genlocking), indispensables à l'affichage sur plusieurs écrans, notamment en cas de rendu stéréoscopique. Le swaplocking est une technique qui force chaque nœud de la grappe à exécuter en même temps le basculement de tampon de sa carte graphique. NJ implante le swaplocking à l'aide d'une barrière de synchronisation forçant tous les nœuds à s'attendre avant le basculement de tampon (« buffer swap »). La bibliothèque SoftGenLock (SGL) implante le support de la stéréo active. Pour cela NJ utilise un tampon virtuel de 2 fois la taille du tampon d'affichage (une demi-zone par oeil), la demi-zone affichée est changée à chaque retraçage vertical. Les cartes graphiques PC sont généralement compatibles VGA et disposent de registres de statut et CRTC. Pour détecter le retraçage vertical, on sonde le registre de statut (attente active). Un temporisateur temps réel (utilisation du système RT-Linux) réveille le thread de SGL juste avant le retraçage, ce qui réduit le temps d'attente active à quelques dizaines de µs. Le signal de synchronisation stéréo peut être envoyé soit par le port parallèle, soit par le port VGA et puis extrait vers les lunettes stéréo. Lorsqu'un nœud détecte le retraçage vertical de sa carte, il exécute une barrière de synchronisation et y mesure le temps passé. Si celui-ci est trop long, le nœud ralentit sa vitesse de rafraîchissement et revient à vitesse normale lorsque le délai redescend à un certain seuil. On estime la durée de barrière nécessaire à un affichage de qualité inférieure à 100 µs. Un réseau de synchronisation dédié évite de surcharger le réseau de communication et de perturber la barrière de synchronisation par les autres communications.

Les auteurs ont ainsi mesuré les performances d'une grappe à base de 4 nœuds (bi P3 800 MHz avec GeForce 2 GTS 64 Mo), avec 2 réseaux différents (Fast Ethernet + réseau de synchronisation et Myrinet 2Gbit/s). Deux applications sont testées (elle n'utilisent qu'un processeur de chaque nœud). La vitesse d'affichage moyenne et la proportion de temps de synchronisation et communication sont mesurés. VRJ est exécuté sur une configuration à un seul nœud (qui rend à 1024 x 768) alors que NJ tourne sur 4 nœuds (la zone d'affichage est donc 4 fois plus grande). Les auteurs n'observent aucune baisse significative de performances avec NJ et SGL, avec les 2 configurations réseau. La proportion du temps de synchronisation est au plus de 4 %. Ce temps dépend de la configuration réseau et du nombre de nœuds mais pas de la complexité de la scène rendue.

4.4.2. Syzygy

Syzygy [SCHA03] est une bibliothèque de programmation d'applications de RV sur grappes de PC. Elle utilise une architecture distribuée avec un modèle de passage de message. Elle se compose de plusieurs couches logicielles. La couche de base est celle de la communication : elle contient une bibliothèque implantant une interface pour les fonctions spécifiques à chaque système d'exploitation (sockets, threads, etc.). Elle contient également l'infrastructure de transmission de message. Les couches intermédiaires de Syzygy se constituent d'unités fonctionnelles spécialisées. Tout d'abord un système d'exploitation distribué (Phleet) qui gère l'exécution de processus sur la grappe (dont le serveur de configuration de celle-ci). Puis un ensemble de protocoles et d'objets client/serveur effectuant la transmission et le rendu de données graphiques et audio. Enfin, une couche de

gestion réseau des E/S, recouvertes par une couche de pilotes logiciels d'E/S. La couche la plus haute de Syzygy est celle des environnements de développement d'applications.

Deux environnements sont disponibles : un environnement de type graphe de scène et un environnement de type maître/esclave. Le premier environnement utilise une base de données graphique distribuée et garantit la cohérence de celle-ci à travers tous les nœuds de la grappe. Il synchronise les différents nœuds implicitement et permet de lancer ou d'arrêter des clients de rendu (les clients de rendu ont une fonction équivalente aux serveurs de rendu de WGL et Chromium) à n'importe quel moment. Il permet donc une grande flexibilité au prix d'une efficacité réduite. Dans l'environnement maître/esclave, une copie de l'application, dite maître, collecte les données utilisateur, effectue des calculs et en distribue les résultats aux nœuds exécutant les instances esclaves. De même qu'avec l'environnement de graphe de scène, des copies de l'application peuvent être lancées et arrêtées à n'importe quel moment, leur resynchronisation se faisant automatiquement. Une boucle d'événements avec des fonctions de rappels est utilisée. Le transfert des données du maître vers les esclaves s'effectue à un moment précis de la boucle. En fin de boucle, toutes les instances exécutent une fonction de rendu, se synchronisent via le réseau et basculent leurs tampons de trame. Le transfert des données est automatique dans certains cas (matrice de navigation, données d'entrées, horodatage, etc.). Cela permet de construire des applications simples de type navigation architecturale sans code ou partage de données explicites. Des méthodes de partage de données explicites peuvent être ajoutées par le programmeur afin d'implanter des applications plus complexes. La cohérence des données doit cependant être gérée par l'application elle-même, contrairement à l'environnement de graphe de scène.

4.4.3. Comparaison entre les implantations parallèles d'OpenGL et les systèmes utilisant la duplication d'applications

Le parallélisme utilisé dans les systèmes logiciels que nous venons de décrire (Syzygy et NJ) consiste à dupliquer l'application sur chaque nœud de la grappe. Selon la classification décrite en 2.2.3, NJ utilise un parallélisme par duplication d'événements alors que Syzygy utilise un parallélisme du niveau graphe de scène. L'utilisation d'instances multiples a pour avantage de limiter le débit des communications inter-nœuds. Dans un tel cas, il n'y a pas besoin de transmettre les données de scène (primitives) entre nœuds, seulement les données des matériels d'interaction ou de point de vue. Le débit requis est donc bien moindre qu'avec un système de rendu parallèle classique comme sort-first ou sort-last, et ne requiert donc pas nécessairement un réseau à haut-débit (1 Gbit/s).

On peut penser que cette méthode est adaptée dans le cas d'une application de RV de type parcours architectural, où la scène est statique. Vu que la taille de la scène est limitée par la capacité mémoire d'un nœud unique, les scènes où les données sont dynamiques et de très grande taille sont proscrites (ce qui peut être le cas des données scientifiques). A priori, cette méthode ne permet donc que d'augmenter la résolution de l'affichage mais pas la taille de la scène. Cela est intéressant dans le cas d'applications de RV, mais est peu efficace. Dans ce cas précis, on parallélise en fait les opérations de tramage (en supposant un pipeline graphique du type OpenGL) mais les opérations de traitement géométrique sont effectuées de manière redondante. En effet, les polygones à tramer sont pour chaque nœud déterminés par les opérations de troncature. La partie applicative (c'est-à-dire l'envoi des commandes graphiques) est, dans ces systèmes, non parallèle.

Le genlocking est indispensable dans le cas de la RV, surtout lorsque l'affichage utilise la stéréo active. La méthode présentée par [ALL02] permet de s'affranchir de l'absence de gestion matérielle du genlocking sur la majorité des cartes grand public (les cartes qui le permettent ne sont pas nécessairement nettement plus performantes). Comme la plupart des cartes PC sont compatibles VGA, il est possible que la méthode utilisée par SGL fonctionne avec de manière correcte. Notons que SGL nécessite cependant l'utilisation d'un noyau Linux temps réel. Des commentaires similaires s'appliquent à la méthode de gestion de stéréo utilisée. Dans le cas de la visualisation scientifique, le genlocking peut être utile dans le cas des configurations d'affichage de type écran en mosaïque. En effet les effets de déchirement (tearing) dus à la non-synchronisation des différents écrans peut nuire à la perception d'une image continue et à la bonne analyse des données visualisées. Cependant, les cartes graphiques susceptibles d'être utilisées dans une grappe de visualisation sont souvent dotées de connecteurs de synchronisation pour le genlocking et permettent la gestion de la stéréoscopie via leurs pilotes logiciels. Cela peut éliminer le besoin d'une méthode de genlocking telle que celle de NJ, qui s'adresse a priori à des systèmes plus bas de gamme (à base de cartes graphiques grand public).

En conclusion, il semble que la méthode de parallélisation présentée s'applique plus à la RV qu'à la visualisation scientifique de jeux de données massifs. Cependant l'utilisation de SGL peut être intéressante en conjonction avec des cartes graphiques standard ne gérant pas le genlocking. Cela peut permettre de baisser le coût global d'une grappe, en supposant que son utilisation n'est pas contraignante.

4.5. Algorithmes utilisés dans les composants logiciels de rendu parallèle

Nous présentons maintenant divers algorithmes utilisés dans les systèmes logiciels de rendu parallèle. Comme nous l'avons vu en 2.2.2.2, les systèmes de rendu parallèle « sort-first » et « sort-middle » souffrent de deux inconvénients majeurs. Premièrement, le déséquilibre de charge, lorsque les primitives projetées sont concentrées dans un petit nombre de pavés de l'espace-écran. Ensuite le chevauchement des primitives, qui peut être important quand la subdivision de l'espace écran est fine. Le déséquilibre de charge diminue l'efficacité du rendu parallèle : plusieurs nœuds ne sont pas alimentés en primitives et seulement quelques nœuds effectuent le rendu des primitives. Dans le pire des cas, un seul nœud rend la totalité des primitives : le système aura une efficacité inférieure à celle du système séquentiel équivalent. Le déséquilibre de charge peut être réduit en augmentant la finesse de subdivision de l'espace-écran, cependant, cela a pour effet d'augmenter le chevauchement. Ce dernier augmente le débit de communication dû à la redistribution des primitives, et également le surcoût de travail dû au parallélisme (une même primitive peut être traitée par plusieurs serveurs). Il existe donc un compromis entre équilibrage de charge et réduction du chevauchement. Cependant, cela suppose d'utiliser une méthode de subdivision statique de l'espace-écran par une grille fixe, chaque pavé de la grille étant géré par un seul serveur de rendu. Des méthodes de subdivision dynamiques ont été développées pour contourner ce problème. Ces méthodes s'appliquent aux systèmes de rendu sur grappe de type « sort-first », car comme nous l'avons vu, la méthode de rendu « sort-middle » est difficilement implantable sur grappe.

4.5.1. Réduction du chevauchement

[SAM98] présente plusieurs algorithmes pour la réduction dynamique du chevauchement dans les systèmes de rendu « sort-first ». Selon les auteurs, un compromis doit être fait entre vitesse de tri des primitives (effectuée par les nœuds clients) et vitesse de rendu (effectué par les serveurs). Une méthode de tri exacte (une primitive n'est affectée qu'aux serveurs dont elle intersecte la zone de l'espace-écran). permet de minimiser le chevauchement, mais est coûteuse en temps de calcul pour le client. A l'opposé, un tri par diffusion de chaque primitive à tous les serveurs est rapide (client) mais force chaque serveur à traiter chaque primitive : le facteur de chevauchement est maximal. Dans les deux cas, il y a goulot d'étranglement, respectivement au niveau du client ou du serveur. On cherche à équilibrer ces deux étapes pour maximiser le débit en primitives du système. Quatre algorithmes sont présentés :

- Exact : projette la primitive dans l'espace écran, effectue un tramage de la projection pour déterminer les zones chevauchées.
- 2D-VE : projette la primitive dans l'espace-écran, calcule un volume englobant (VE) 2D de la projection et détermine les zones chevauchées par celui-ci.
- 3D-VE : calcule un VE 3D de la primitive, le projette sur l'écran, calcule un VE 2D du VE 3D projeté, détermine les zones chevauchées par le VE 2D.
- Diffusion : considère qu'une primitive chevauche toutes les régions de l'espace-écran.

Une méthode d'amortissement sur paquet de primitives est également utilisée pour les algorithmes 2D-VE et 3D-VE : le VE est calculé sur un paquet (ensemble) de primitives. Le surcoût de calcul du VE est donc amorti sur plusieurs primitives, au prix d'une précision encore réduite, le tri étant effectué sur un paquet de primitives et non sur une seule primitive. Cependant le nombre de primitives agrégées en un paquet (le facteur d'amortissement ou FA) est réglable. Les tests des différents algorithmes ont été effectués sur des applications réelles exécutées sur une grappe expérimentale. Plus le FA est grand, plus la vitesse de tri augmente, mais plus le temps de rendu est important. Les auteurs observent la valeur de FA donnant le meilleur débit en primitives rendues/s varie selon les applications (c'est-à-dire la scène rendue). Les auteurs utilisent un modèle d'estimation dynamique du FA idéal (le FA donnant les meilleurs performances de rendu pour une scène et un point de vue donnés). Ce modèle cherche à équilibrer les temps de traitement entre client et serveur pour maximiser le débit du système. En utilisant un modèle de coût par primitive simplifié, il prédit pour l'image n+1 la valeur du FA permettant d'équilibrer les temps de traitement. Les résultats montrent que l'amortissement dynamique offre un meilleur débit que l'amortissement statique.

[CHE98] présente une méthode d'évaluation du rendu par paquets de primitives triées (« bucket rendering »). Les auteurs présentent des modèles analytiques d'évaluation du chevauchement, qui, couplés avec des simulations, démontrent que l'impact du chevauchement est limité. Le modèle considère le pipeline graphique matériel à partir de l'étape de mise en place des triangles (« triangle setup ») jusqu'à l'étape du remplissage des triangles (ces deux étapes constituent l'étape de tramage). Il se base sur des hypothèses simplificatrices. Le modèle montre que l'impact est le plus important lorsque le temps de calcul par triangle est égal au temps de calcul des pixels, c'est-à-dire quand le pipeline est équilibré. Les auteurs concluent qu'un système de rendu matériel possède un « point de conception », représentant la valeur d'aire de triangle (k) pour laquelle le pipeline est équilibré. Cependant, ce modèle s'applique principalement aux accélérateurs utilisant le « bucket-rendering », c'est-à-dire les systèmes fortement couplés de type « sort-middle ». Cependant, nous pensons que certaines conclusions peuvent néanmoins être utiles à la conception de systèmes parallèles SF sur grappe (la méthode SF utilise également un tri des primitives ou « bucketization »). Ainsi, comme chaque nœud d'un tel système utilise un seul accélérateur de rendu, si on suppose que l'augmentation des performances d'un tel accélérateur diminue k, cela permet d'utiliser des tailles de données de plus en plus grandes (c'est-à-dire de plus en plus de primitives).

4.5.2. Equilibrage de charge

Plusieurs travaux ont été effectués pour résoudre les problèmes d'équilibrage de charge.

[SAM99] présente plusieurs algorithmes utilisés pour équilibrer la charge de travail dans les systèmes SF. Les auteurs utilisent une grappe de 8 nœuds de rendu, chaque nœud étant attaché à un projecteur, et 1 nœud client. Le système découpe l'espace écran en plusieurs zones virtuelles (ZV) non-chevauchantes et ne correspondent pas obligatoirement 1 à 1 avec les zones physiques des projecteurs. L'unité de travail du système est le rendu de tous les pixels d'une ZV, donc le rendu de toutes les primitives intersectant la ZV. Une ZV inclue dans une zone de projection (ZP) d'un serveur (local) peut être rendue sur un serveur distant, et ses pixels alors transférés au serveur local. Le système utilise un graphe de scène (GS), et permet de prédire les temps de rendu de ses différents nœuds : cela permet un équilibrage de charge et une affectation dynamique précis. De plus, la scène est répliquée sur tous les nœuds, seuls les pixels rendus à distance sont transférés entre serveurs. Lors du rendu d'une image : le client détermine les nœuds visibles du GS, décompose l'espace-écran en ZV, y trie les nœuds visibles et affecte chaque ZV à un serveur. Il envoie un message de rendu à chaque serveur, qui rendent alors leur partie du GS.

Les auteurs ont développé trois algorithmes différents d'affectation des ZV aux serveurs. Les objectifs de ces algorithmes sont :

- Minimisation des surcoûts de redistribution de pixels, de chevauchement (rendu redondant de primitives), et du traitement d'une ZV.
- Equilibrage du travail : les ZV doivent être construites et affectées aux serveurs de manière à équilibrer les coûts de rendu entre serveurs.
- Partitionnement rapide : une partition doit être assez simple pour être calculée en temps réél à chaque image, et le tri des nœuds de la scène parmi les ZV doit être fait à chaque image.

Les trois algorithmes sont :

- GRILLE : utilise une grille rectangulaire régulière. A chaque image, l'algorithme répartit les nœuds du graphe parmi les cases de la grille, et pour chaque case estime la charge de travail associée. Chaque case est affectée au serveur (local) dont elle chevauche la ZP. L'algorithme est itératif : il cherche à répartir la charge de travail en affectant à distance aux serveurs les moins chargés des cases locales aux serveurs les plus chargés.
- UNION : cherche à réduire l'impact du chevauchement, important lorsque la grille est fortement divisée, tout en minimisant les déséquilibres de charge. Son principe est d'unifier les cases affectées à un même serveur et intersectant la même ZP en une seule ZV. Au lieu de rendre les primitives de chaque case séparément, elles ne sont rendues qu'une fois pour toute la ZV, ce qui réduit fortement le chevauchement. Les cases affectées de manière distante sont également unifiées en ZV.
- « KD-split » : cet algorithme cherche à construire P ZV (P le nombre de serveurs), avec la même charge de travail. Il procède par subdivision récursive. L'espace écran est balayée horizontalement par une ligne verticale, et les coûts de rendu des parties gauche et droite sont mis à jour. Le balayage cesse lorsque ces deux coûts sont égaux. L'algorithme s'exécute récursivement P-1 fois, ce qui construit P ZV.

Les auteurs montrent, avec des tests d'applications de rendu surfacique, que l'algorithme à grille statique donne un déséquilibre de travail très important quand la scène n'occupe qu'un seul projecteur. Les performances de l'algorithme GRILLE varient suivant la subdivision (de 4x3 à 16x12 cases par projecteur) : lorsqu'elle est faible, le chevauchement est réduit, mais les temps de déséquilibre sont importants. A forte subdivision, le déséquilibre est réduit mais les temps de chevauchement sont importants. L'algorithme UNION réduit fortement le chevauchement à subdivision 16x12 par rapport à GRILLE. Cela est dû à l'affectation locale des cases proches. Cependant, le déséquilibre est plus important. L'algorithme ne peut en effet affecter à distance des ZV facilement que quand les nœuds qu'elles contiennent sont totalement dedans. Cela génère un déséquilibre lorsque les nœuds sont grands par rapport à la taille de case. L'algorithme KD-split a des temps de rendu uniformes, un chevauchement faible et peu de déséquilibre. Cependant les coûts de redistribution de pixels sont plus importants que pour les autres algorithmes. Les auteurs montrent ainsi que les performances des 3 algorithmes sont variables suivant la scène rendue et qu'aucun n'est réellement supérieur.

Les algorithmes d'équilibrage de charge permettent, selon [SAM99], d'avoir un rendu plus efficace qu'un algorithme de partitionnement statique, ce qui est particulièrement vrai lorsque la projection de la scène ne recouvre qu'une petite partie de l'espace-écran. Cependant, ce type de méthodes utilise une structure de type graphe de scène pour prédire les temps de rendu associés à chaque zone de l'espace-écran. Cela est rédhibitoire dans le cas de scènes de très grande taille, où une telle structure de données induit a priori un surcoût important (en place mémoire et en temps de traitement). Ensuite, ce système requiert une réplication de la scène sur chacun des nœuds de rendu, ce qui limite son extensibilité. Enfin, le calcul (dynamique) de la partition est effectué séquentiellement, ce qui limite la vitesse de calcul de la partition et ainsi la vitesse de rendu globale (en images/s). Nous pensons que les caractéristiques d'une telle méthode sont plus adaptées à la réalité virtuelle. Cependant, des études détaillées des problèmes de déséquilibre de charge sur des applications de visualisation scientifique seraient intéressantes afin d'évaluer si l'utilisation du partionnement statique est réellement néfaste aux performances des systèmes de rendu SF.
Dans la partie suivante, nous présentons plusieurs applications scientifiques du rendu distribué sur grappe.

5. Les applications scientifiques du rendu sur grappe

En visualisation scientifique, il existe une dualité importante entre rendu surfacique et rendu volumique. En effet, les données utilisées sont souvent décrites dans un espace à 3 dimensions. Un grand nombre de sources existent :

- Données médicales : IRM, TEP, etc.
- Données provenant de simulations numériques : dynamique des fluides, etc.
- Données provenant de mesures physiques : météorologiques, sismiques, astrophysiques, etc.

Les données 3D peuvent ainsi être visualisées à l'aide de deux grands types de rendu, surfacique et volumique, suivant les informations que l'on cherche à rendre. Les exigences en ressources de calcul de ces méthodes sur des jeux de données de très grande taille sont importantes, tout en étant différentes, ce qui nécessite d'utiliser le rendu parallèle pour permettre une visualisation interactive. L'augmentation récente des performances des grappes de calcul (taille mémoire disponible et vitesse de calcul et rendu) permet de développer de telles applications avec des jeux de données dont la taille est de plus en plus grande. Malgré cela, la taille des jeux de données (notamment ceux produits par les simulations numériques) est telle que la totalité ne tient pas dans la mémoire centrale de la grappe et/ou la mémoire du sous-système graphique, ce qui nécessite l'utilisation de méthodes « out-of-core » (OOC) dans lesquelles, à un temps donné, seule une sous-partie de la scène totale est présente dans la mémoire du système (cf 4.3.1.4). De plus chaque grand type de rendu nécessite généralement l'utilisation de méthodes de parallélisme différentes.
Nous décrivons tout d'abord les applications de visualisation scientifique sur grappe à base de rendu surfacique.

5.1. Rendu surfacique

5.1.1. Rappel

En informatique graphique, les objets sont généralement représentés par leur surface [FOL90]. En effet, ils sont visibles grâce à la lumière que leur surface émet. Visualiser les objets revient à observer l'interaction de la lumière avec leur surface. Une surface peut être décrite de plusieurs façons dont les suivantes :

- De manière discrète, par un maillage de sommets reliés par des polygones (souvent des triangles).
- Par des surfaces paramétrées, par exemple des surfaces de Bézier.
- Par des objets de base (cubes, sphères, tores, etc.).

En visualisation temps-réel et notamment en visualisation scientifique interactive, les objets surfaciques sont souvent représentés par maillages de triangles. Ce sont par exemple les isosurfaces, plans de coupe, surfaces de maillages, etc. Cela permet de représenter des objets géométriquement et topologiquement arbitraires au prix d'une très forte discrétisation si on veut un niveau de détail important. Le rendu surfacique sur grappe est compatible avec les méthodes de rendu parallèle « sort-first » et « sort-last ». Comme nous l'avons vu, la méthode « sort-middle » est assez peu adaptée à l'architecture faiblement couplée des grappes. Nous décrivons maintenant différentes applications de visualisation scientifique à base de rendu surfacique sur grappe.

5.1.2. Applications existantes

5.1.2.1. Rendu sort-last

[ZHA01] décrit un système de visualisation d'isosurface (extraites en temps réel à partir de données volumiques médicales) à base de grappe de PC. Ce système utilise un parallélisme au niveau de l'extraction de données jusqu'au rendu final. On s'intéresse ici à l'extensibilité de la phase initiale d'extraction de surface grâce à un algorithme «out-of-core » totalement parallèle. Une stratégie de partitionnement en fonction d'un spectre de travail permet de minimiser les E/S disques pendant le chargement des données. L'architecture du système est en pipeline. Des nœuds de calcul isosurfacique génèrent progressivement des flux de triangles (portions d'isosurfaces), transmis à des nœuds intermédiaires de rendu. Les images rendues sont composées en parallèle par un matériel spécialisé (Metabuffer).

L'algorithme d'extraction utilise calcul parallèle et accès aux données sur disque en parallèle. Il cherche à répartir la charge sur différents processeurs, minimiser le nombre d'E/S parallèles et minimiser les accès inter-processeurs. Pour réduire ce dernier facteur (le moins extensible), on partitionne donc les données de manière statique. La partition se fait suivant un histogramme de charge de travail. Puis un arbre d'E/S optimales pour chaque disque est construit. Le bloc (ensemble rectangulaire de cellules adjacentes) est utilisé comme unité de partionnement et d'accès aux données pour équilibrer place disque et nombre d'E/S. Le déroulement de l'algorithme est alors le suivant :

- Partitionnement des données en blocs, et classification de chaque bloc dans une matrice en fonction de sa valeur minimum de fonction et l'intervalle (discrétisé) de valeurs qu'il couvre.
- Distribution des blocs : pour chaque élément de la matrice, les blocs y appartenant sont affectés également à tous les processeurs, ils sont redistribués de manière à ce que chaque processeur ait le même nombre de blocs d'un élément de matrice.
- Construction d'un arbre d'intervalle externe : chaque bloc affecté à un processeur est rajouté à un fichier de blocs, l'intervalle associé est stocké dans un fichier d'intervalles, utilisé pour construire un index des blocs.
- Construction de l'isosurface : un parcours de l'arbre est effectué pour trouver les blocs dont l'intervalle est intersecté par l'isosurface. Puis celle-ci est extraite des blocs intersectés et transmise aux serveurs de rendu.

L'isosurface extraite est transmise sous forme compressée. Pour cela une méthode de codage d'indice d'arête est utilisée. Chaque « bitmap » d'une tranche de données est compressé par une méthode adaptative « run-length » ou par un codage arithmétique. Les valeurs de fonction des sommets des arêtes intersectées sont codées en utilisant leurs différences de second ordre. Chaque bitmap d'une couche de cellules est codé de manière similaire.

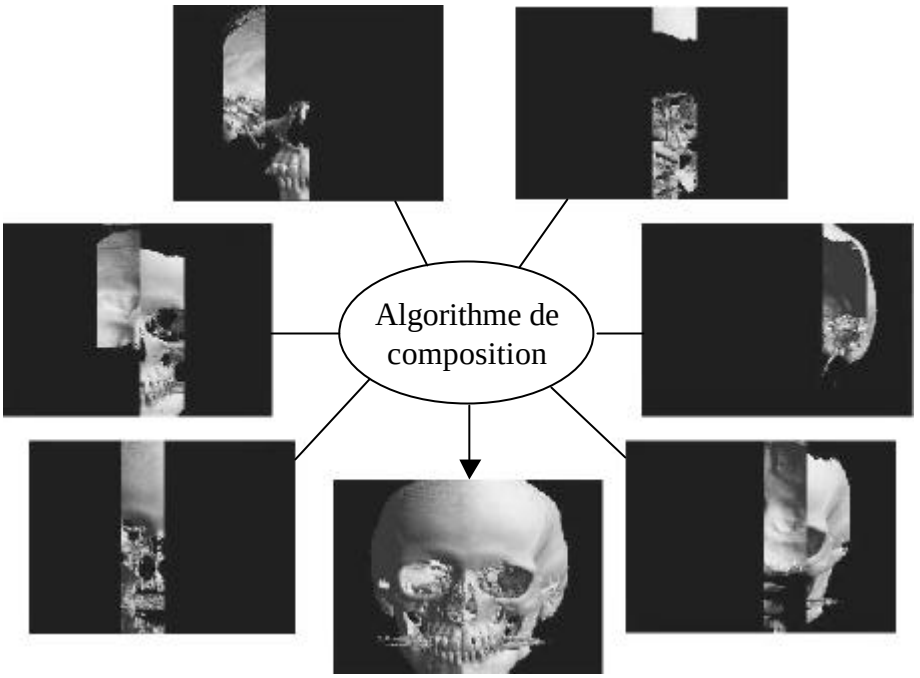
Cette méthode permet la compression et la reconstruction incrémentale des surfaces et utilise une place mémoire nécessaire à 2 tranches et 1 couche de données. La composition des images rendues se fait en matériel et donc n'est a priori pas critique pour les performances du système. Les résultats présentés par les auteurs montrent que la phase critique du pipeline du système est celle de l'extraction. Pour des données de 512 x 512 x 1252, on observe un temps de rendu jusqu'à 8 s sur une grappe de 32 nœuds (pour environ 50M de triangles extraits). Dans ce cas, l'étape limitante du système est donc celle de calcul (les serveurs de rendu sont sous-alimentés).

[WYL01] présente un système de rendu parallèle « sort-last » sur grappe de données surfaciques. Dans cette approche, la scène est répartie de manière statique avant le rendu, et à chaque image, chaque nœud rend son sous-ensemble de données. L'image finale est composée en parallèle (suivant un algorithme donné par l'utilisateur) à partir des sous-images extraites de chaque nœud. Les auteurs ont développé une bibliothèque OpenGL parallèle (libpglc) de rendu « sort-last », qui fournit des opérations de composition arbitraires (comparaison z, XOR, etc.) avec des formes de communication arbitraires (arbres, « binary-swap »). Elle utilise les routines de communication MPI de «Parallel Mesa » mais fonctionne dans n'importe quel environnement OpenGL. Elle permet à l'application appelante de créer son propre contexte graphique et d'utiliser des extensions GL spécifiques. Elle permet aussi l'utilisation d'algorithmes de composition spécifiques. Elle utilise les optimisations suivantes :

- Lecture du tampon image : on choisit les formats de pixels (couleur et z) donnant les meilleures performances sur les cartes graphiques utilisées (Nvidia GeForce 1).
- Débit réseau : chaque image (1024 x 768) est compressée avec une structure de données APE (Active Pixel Encoding) : celle-ci stocke l'indice et la longueur des bandes de pixels actifs (non nuls). Cela correspond à la méthode « SL-sparse » (voir 2.2.2.1). On observe une réduction de taille de 10 à 15 fois pour 64 nœuds.
- Algorithme de composition : il s'effectue directement sur les données entrantes compressées, sans décompression. Les canaux RVBA et z sont agrégés, cela amortit le coût des indices sur 2 canaux. L'algorithme n'effectue de comparaison z que sur les segments de pixels actifs se chevauchant dans les images source et destination ce qui permet un gain de 50 %.

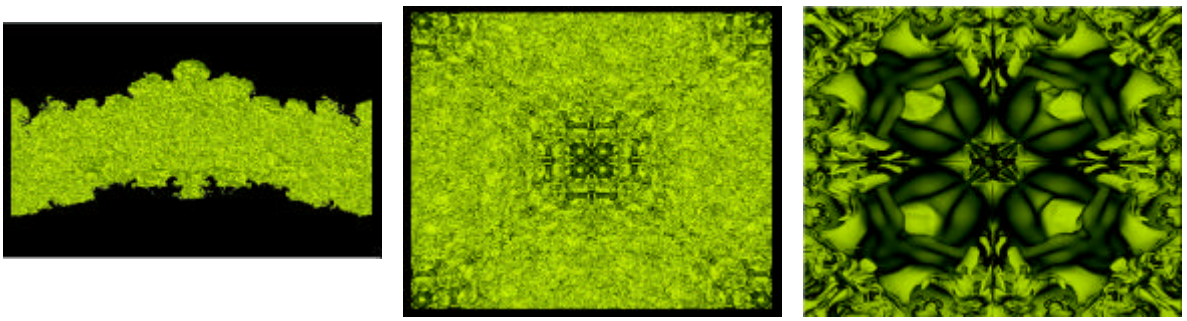
La Figure 23 ci-dessous (tirée de [WYL01]) montre le rendu d'une image finale d'un jeu de données surfaciques à partir des images intermédiaires, calculées par les différents nœuds de rendu. L'algorithme de composition transmet l'image finale (représentée tout en bas de la figure) à un nœud unique, qui la conserve en mémoire ou l'affiche avec son accélérateur graphique local.

Figure 23 : rendu parallèle sort-last et composition en profondeur



Les auteurs évaluent les performances d'une application de rendu parallèle surfacique. Les jeux de données utilisées proviennent de simulations numériques de turbulence. Chaque jeu (statique) est un pas de temps unique et est calculé lors d'une phase de pré-traitement par un algorithme d'extraction d'isosurface. Leur taille est comprise entre 7 Mtriangles et 469 Mtriangles. Ainsi, avec un jeu de données de 469 Mtriangles et 64 nœuds, l'application rend à une vitesse de 300 Mtriangles/s, avec une efficacité de 85% (la vitesse idéale de rendu étant 352 Mtriangles/s avec 64 nœuds). Les images ci-dessous (tirées de [WYL01]) montrent plusieurs vues d'une isosurface de 469 Mtriangles, la caméra se rapprochant graduellement du centre de l'isosurface au cours de la séquence d'images, montrant ainsi une vue de plus en plus détaillée.

Figure 24 : images d'une isosurface de 369 Mtriangles



5.1.2.2. Rendu sort-first

[COR02] décrit un système parallèle « sort-first » de rendu « out-of-core » de scènes de type architectural de grande taille (de l'ordre de 13M de polygones) sur grappes de PC avec affichage en mosaïque. Un algorithme de prétraitement construit une représentation hiérarchique de la scène sur disque. Durant l'exécution, chaque PC rend sa zone d'affichage par une méthode OOC qui gère rendu, calcul de visibilité et opérations d'entrées-sorties avec plusieurs threads. Un thread de rendu détermine les nœuds visibles et demande à des threads de gestion mémoire le contenu de ces nœuds (qui sont en mémoire du nœud sinon sur disque). Un autre thread prédit les

données requises à l'image suivante pour étaler les E/S disque dans le temps. L'algorithme de visibilité utilisé (PLP, « Prioritized-Layered Projection ») offre un compromis entre temps de calcul de visibilité et précision des résultats. Il utilise un algorithme modifié de troncature par volume de vue : il classe les nœuds de la scène suivant la priorité décroissante (déterminée lors d'une phase de prétraitement par des méthodes heuristiques), et cesse le parcours de la scène après qu'un certain nombre de nœuds ont été traversés. Le rendu est effectué par une méthode de type « sort-first » : chaque nœud rend la partie de la scène qui est déterminée comme visible dans sa zone d'affichage. Les auteurs présentent les résultats de grappes de 1 à 16 serveurs de rendu. Avec 16 serveurs et 70000 triangles alloués par nœud, l'application rend 12 Mpixels à 10,8 im/s.

5.1.2.3. Autres applications de rendu surfacique

[SHA02] présente un système de visualisation interactive et immersive de jeux de données moléculaires. Ce système est constitué de deux sous-parties : une partie calculant la partie potentiellement visible de la scène, et une partie effectuant le rendu proprement dit des données (atomes) visibles. Ces deux sous-parties sont exécutées respectivement sur une grappe de PC et sur une machine dédiée de type SMP (Onyx2), reliées par un réseau d'interconnexion.

Le système comprend 3 modules :

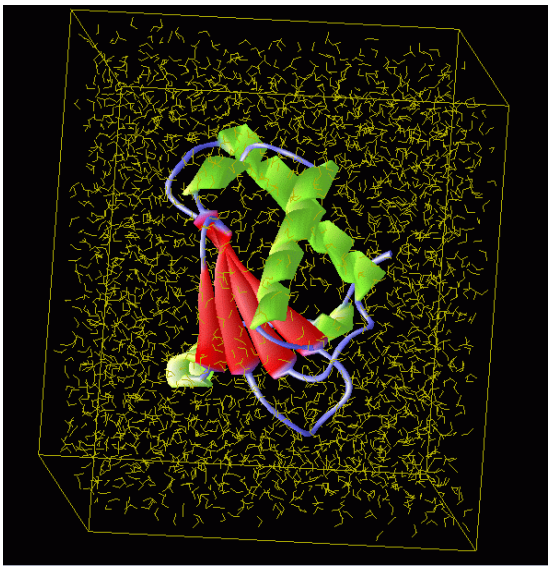
- Un module d'extraction de données parallèle (MED) : il stocke les données (atomes) dans un octree, ce qui permet de n'afficher que les atomes appartenant aux régions de l'octree intersectant le champ de vue. La version parallèle du module partitionne le volume de vue entre les nœuds de la grappe, chaque nœud gardant une copie de l'octree.
- Module d'exclusion probabiliste (MEP) : ce module réduit le nombre d'atomes à rendre, en calculant la probabilité qu'a un atome d'être visible, pour chaque région considérée. Le calcul de la probabilité est récursif et décroît avec la distance de la région considérée au point de vue. Un tirage aléatoire détermine la visibilité de chaque atome.
- Module de rendu et de visualisation : le MRV est interfacé avec le MED, à qui il transmet la position de l'utilisateur, et avec le MEP qui lui transmet les données à rendre. Une technique d'élimination par occlusion est utilisée avant le rendu pour réduire le nombre d'atomes à rendre. Celle-ci utilise un tampon z simulé pour déterminer si un atome est visible ou pas. Son efficacité augmente avec la taille de la scène (nombre d'atomes). Une technique de rendu par niveau de détail permet également une augmentation de la vitesse de rendu.

Le MED et le MEP sont exécutés par une grappe de PC, chacun de ces modules est parallèle. Le rendu (MRV) est effectué sur une machine dédié (Onyx2). Pour réduire les effets de la latence réseau, le rendu de l'étape N-1, l'extraction de données et l'élimination sélective à l'étape N sont effectués en parallèle (ce qui introduit cependant une latence de une image). Les calculs de pré-rendu de la grappe sont effectués de manière totalement logicielle. De plus, le rendu est séquentiel, car un seul accélérateur graphique (pipe) est utilisé. L'intérêt de ce système est de pouvoir réduire le temps de pré-calcul. Cependant, le pipeline de calcul paraît déséquilibré, car d'après les résultats des auteurs, le module qui a le plus grand temps de calcul est celui de rendu (quasiment un ordre de grandeur par rapport aux 2 autres modules, MED et MOP). Il est donc possible que l'utilisation d'une autre grappe à la place de la machine SMP utilisée permette de baisser fortement le temps de rendu. De plus, même si le système gère des jeux de données de grande taille (10^9 particules), le nombre de particules réellement rendues est relativement modeste (50000), ce qui ne nécessite pas un affichage avec beaucoup de pixels (environ 1M de pixels). Nous pensons que dans ce cas, l'utilisation de composants à faible coût pour le rendu donnerait probablement un meilleur rapport performances/prix au système.

[KLO02] présente une application de visualisation moléculaire, fonctionnant sur le système de rendu parallèle DeepView (voir 3.5.2). Cette application utilise le système de visualisation distribuée OpenDX-MPI, qui est une extension, développée par les auteurs, du logiciel de visualisation scientifique OpenDX. OpenDX-MPI permet de visualiser de manière distribuée des simulations parallèles distribuées. Les processus de simulation et de visualisation peuvent s'effectuer de manière concurrente. OpenDX-MPI distribue les opérations de visualisation à la totalité des nœuds de la grappe DeepView, ce qui permet de rendre à haute résolution (les auteurs ne spécifient pas la méthode de rendu parallèle utilisée). Les auteurs ont ainsi utilisé cette extension pour visualiser interactivement des simulations de dynamique moléculaire effectuées avec le logiciel libre GROMACS

(« Groningen Machine for Chemical Solutions »). La simulation et sa visualisation s'effectuent donc de manière concurrente sur la grappe. La Figure 25 ci-dessous montre la visualisation d'un pas de temps d'une simulation de dynamique moléculaire. Au centre de l'image, on peut voir la structure secondaire d'une protéine (représentée par des hélices et rubans). Les structures de la protéine sont rendues de manière surfacique, alors que les molécules entourant la protéines (solvant) sont représentées par un rendu de lignes.

Figure 25 : visualisation surfacique d'un pas de temps d'une simulation de dynamique moléculaire



Dans la sous-partie suivante, nous décrivons des applications du rendu volumique sur grappes.

5.2. Rendu volumique

Dans cette partie nous décrivons des applications de rendu volumique sur grappes de PC. Tout d'abord nous effectuons un rappel du principe du rendu volumique, et notamment la manière dont il peut être effectué à l'aide de cartes graphiques standard grâce à des méthodes de placage de texture et de composition par transparence. Ensuite, nous décrivons deux schémas de partition de données permettant de paralléliser le rendu volumique et montrons comment ces deux méthodes peuvent être implantées dans des architectures particulières. Puis nous présentons des applications du rendu volumique sur grappe de PC.

5.2.1. Rappels

En visualisation scientifique, les données à analyser sont souvent exprimées dans un espace à 3 dimensions. Ainsi la scène à rendre est souvent décrite par un volume 3D, discrétisé par une grille 3D, régulière ou irrégulière.

Comme nous l'avons vu en 5.1.1, de telles données peuvent être rendues de manière surfacique, en calculant une isosurface qui délimite les régions de l'espace dont les points ont une valeur associée supérieure ou égale à la valeur d'isosurface. Cependant, si cette méthode permet d'étudier certaines caractéristiques des données (une frontière délimitant plusieurs régions de l'espace, par exemple une onde de choc dans un fluide), elle ne permet pas d'apprécier pleinement leur caractère tridimensionnel, vu qu'une surface est (dans un espace 3D) bidimensionnelle. Le rendu volumique permet, comme le rendu surfacique, de générer une image 2D à partir d'un jeu de données 3D, mais en faisant contribuer la totalité des données (se trouvant dans le volume de vue) à l'image finale. Ainsi, le calcul de l'image finale ne nécessite pas la génération d'une représentation géométrique intermédiaire : c'est le rendu volumique direct. Selon [ELV92], la plupart des algorithmes de rendu volumique peuvent se décrire par un certain nombre d'étapes communes. Ce sont :

- L'acquisition des données.

- Le traitement des données afin de les rendre acceptables (intervalle de valeurs, bruit, etc.).
- Classification : cette étape permet d'attribuer certaines propriétés (couleurs, opacité, ou autres) aux différentes régions du volume en fonction de leurs intervalles de valeurs.
- Etablissement d'une correspondance entre les éléments du volume et des primitives (cellules, voxels, etc.).
- Calcul des teintes des primitives (shading), transformation et détermination de leur contribution à l'image finale.

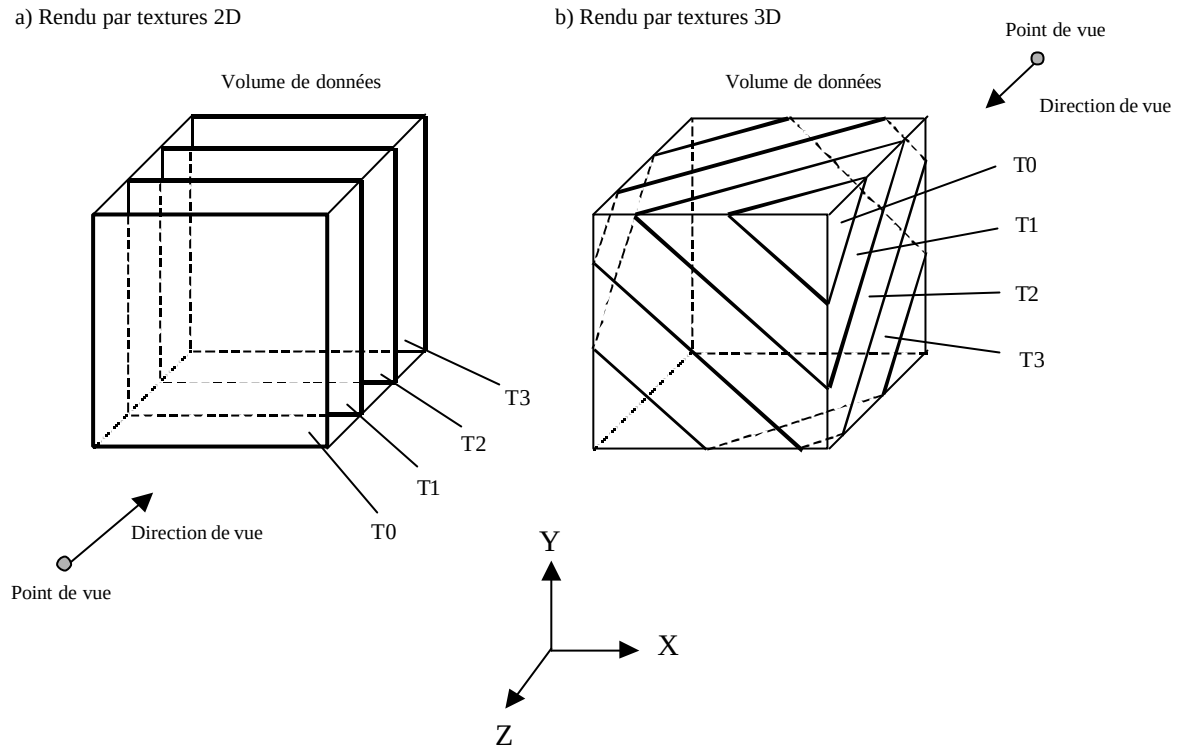
L'étape de correspondance est celle qui varie le plus selon les différents algorithmes. [SCHR98] décrit le rendu volumique comme n'importe quelle méthode qui utilise des données volumiques (3D) pour générer une image. Deux approches principales sont utilisées : le rendu par pixel et le rendu par objet. En rendu par pixel, l'image est calculée en itérant sur chacun de ses pixels. Pour chaque pixel du plan-image, un rayon (demi-droite) est tracé : c'est la méthode du lancer de rayon (ray-casting). Pour chaque rayon, les éléments du volume traversés sont déterminés : chaque élément de volume ajoute au rayon une contribution de couleur, déterminée en fonction des valeurs associées à l'élément de volume et d'une fonction particulière (dite fonction de transfert), cette dernière étant déterminée lors d'une étape préliminaire de classification. Comme les jeux de données 3D sont constitués de points (sans dimensions), un élément de volume est généralement défini comme un cube dont les 8 sommets sont 8 points voisins de la grille (voxel).

En rendu par objet, le volume de données est traité en fonction de l'organisation des voxels dans le jeu de données, et des paramètres de point de vue. Une méthode de rendu typique utilise la composition alpha : les voxels sont parcourus et composés itérativement de l'arrière du volume de vue vers l'avant. Ce type de méthode est particulièrement bien adapté au rendu matériel : ce sont les méthodes à base de composition matérielle de textures 2D ou 3D. Le principe de ces méthodes est d'échantillonner le volume par un ensemble de polygones texturés. Ces polygones sont projetés sur le plan-image et composés de manière à produire l'image finale. Les détails des 2 méthodes (textures 2D ou 3D) diffèrent :

- Textures 2D : décomposition du jeu de données par un ensemble de tranches espacées régulièrement et alignées sur les axes du repère global. 3 ensembles sont requis (1 par axe), et l'ensemble dont l'axe est le plus aligné avec la direction de vue est utilisé pour le rendu. Chaque tranche est une image (calculée par interpolation bilinéaire) chargée en mémoire de texture et rendue sur un quadrilatère plan.
- Textures 3D : décomposition du jeu de données par un ensemble de tranches régulièrement espacées et alignées sur l'axe de la direction de vue. Chaque tranche est une image 2D calculée par interpolation trilinéaire, ce qui permet d'utiliser des plans de texture alignés orthogonalement sur la direction de vue ou bien des morceaux de sphère concentriques.

L'algorithme de rendu volumique à base de texture procède itérativement : pour chaque tranche, la texture correspondante chargée en mémoire est rendue avec composition alpha.
La Figure 26 ci-dessous représente la subdivision d'un volume de type grille régulière par 4 tranches, plaquées sur des quadrilatères. Les tranches sont rendues dans l'ordre T3,T2,T1,T0, T3 étant la tranche la plus éloignée du point de vue et T0 la plus proche.

Figure 26 : rendu volumique par composition de textures



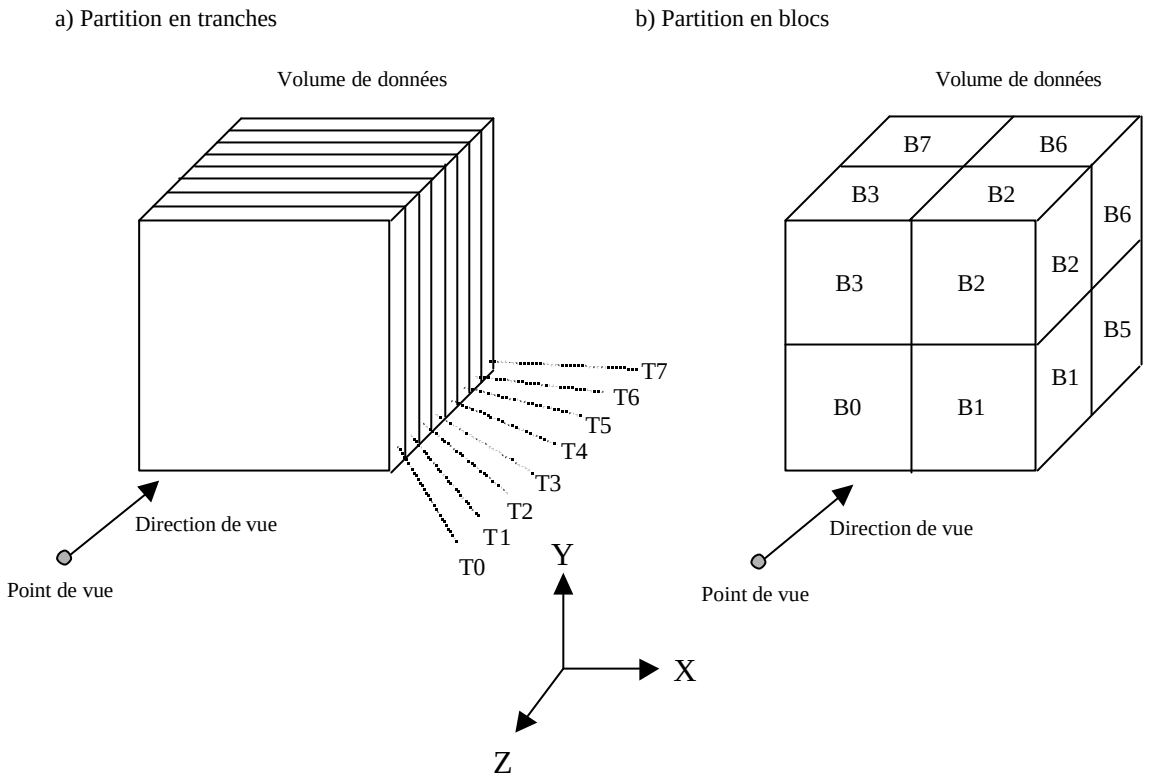
Le rendu par textures 3D considère le volume comme une image 3D alors que le rendu par textures 2D découpe le volume en une série de tranches (images 2D). De plus, le rendu par textures 3D produit moins d'artefacts, grâce à l'interpolation trilinéaire, alors que le rendu par textures 2D y est beaucoup plus sujet (chaque texture n'est interpolée que bilinéairement, c'est-à-dire dans les 2 dimensions du plan de l'image). Cependant, le rendu par texturage 3D requiert des fonctionnalités matérielles spécifiques alors que le rendu par texturage 2D peut être implanté avec n'importe quel accélérateur graphique standard qui supporte le placage de textures 2D (c'est-à-dire la totalité des accélérateurs actuels).
Dans ces méthodes, la classification des données s'effectue, préalablement au rendu, en attribuant à chaque pixel d'une tranche texturée des valeurs de couleurs et d'opacité suivant une fonction de transfert déterminée par l'utilisateur. Notons que le rendu par texturage matériel limite a priori les fonctions de transfert utilisables par rapport au rendu « par pixel » : en effet celles-ci doivent être implantées à l'aide des caractéristiques du matériel, ce qui limite leur variété. Cependant, les possibilités de programmation des cartes COTS récentes devraient en partie lever cette barrière et permettre d'implanter plus aisément une plus grande variété de fonctions (illumination de type Phong, gradient, etc.).
Nous décrivons dans la sous-partie suivante le principe du rendu volumique par objet parallèle.

5.2.2. Rendu volumique parallèle

Les premières applications parallèles du rendu volumique étaient logicielles et utilisaient surtout la méthode par pixel. Les méthodes de rendu par objet se prêtent néanmoins particulièrement bien à une parallélisation, notamment grâce à l'augmentation constante des performances des matériels de rendu. Le principe est le suivant : le volume de données est partitionné en sous-volumes, chaque sous-volume est rendu par une unité de traitement matérielle indépendante, et les images intermédiaires résultantes sont composées pour donner l'image finale. La méthode de rendu parallèle s'appliquant le mieux à ce type de rendu est ainsi la méthode « sort-last ». Chaque unité de rendu rend indépendamment sa partie de la scène et les images intermédiaires sont « triées » par un système de composition qui rend l'image finale.

La Figure 27 ci-dessous montre deux méthodes de partition différentes : la méthode par tranche et la méthode par bloc. Dans chacune des deux méthodes, le volume de données (d'une taille de n^3 points) est partitionné en N sous-volumes de $\frac{n^3}{N}$ points, N étant le nombre de nœuds de rendu de la grappe. Par contre, la méthode de répartition diffère : dans la méthode par tranches, le volume est découpé en tranches verticales de $\frac{n^3}{N}$ points alors que dans la méthode par blocs, le volume est découpé en blocs de $\frac{n^3}{N}$ points. Ici, N vaut 8, les blocs sont alors d'aspect cubique. Bien qu'une tranche et un bloc ont la même taille (en nombre de points), chacune de ces deux méthodes de partition a un coût de composition différent. Dans la méthode par tranches, chaque nœud compose l'image rendue à partir de sa tranche avec l'image rendue par le nœud précédent à partir de la tranche précédente. Par exemple, le nœud rendant la tranche T3 compose son image avec l'image de la tranche T4. Si on suppose qu'un voxel ne contribue qu'à un seul pixel, chaque nœud (excepté celui qui rend la tranche la plus éloignée du point de vue) effectue donc n^2 opérations de composition. De plus, le nœud qui rend la tranche i doit attendre que le nœud qui rend la tranche $i+1$ ait rendu son image $(0 \le i < N)$: la composition s'effectue en cascade. Cette méthode induit donc une latence totale (vu que la composition s'effectue en pipeline) de $(N-1)T$, avec T le temps de composition d'une image, proportionnel à n^2. En effet, comme les nœuds rendent en parallèle, ils ne peuvent rendre leur image partielle suivante sans que l'image finale ait été totalement composée. Dans la méthode par blocs, chaque nœud n'effectue au plus que $\left(\frac{n}{N}\right)^2$ opérations de composition, les nœuds qui rendent les blocs les plus éloignés du point de vue n'effectuent pas de composition et transmettent simplement leur image partielle rendue. Chaque bloc transmet son image partielle au bloc ayant la même altitude (Y) mais étant plus près de l'utilisateur. Les images partielles des blocs de même coordonnées Y sont donc composés en chaîne. Cependant la profondeur de cette chaîne est de $\sqrt[3]{N}$ et chaque chaîne compose en parallèle. La latence de composition est donc de l'ordre de la profondeur d'une chaîne, c'est-à-dire du nombre de blocs par coté du volume de données (soit $\sqrt[3]{N}$).

Figure 27 : méthodes de partitionnement d'un volume de données



Le Tableau 7 ci-dessous compare les performances des 2 méthodes de partitionnement. Pour chaque catégorie, on indique l'ordre de grandeur du nombre d'opérations en fonction du nombre de nœuds (N) et de la résolution du volume (n). On suppose ici que la racine cubique de N est un nombre entier (les blocs de la 2ème méthode de partition ont donc un aspect cubique)

Tableau 7 : coûts théoriques de méthodes de composition

Méthode de partition	Partition en tranches	Partition en blocs
Dimensions des sous-volumes en nombre de voxels (hauteur, largeur, profondeur)	$\left(n, n, \frac{n}{N}\right)$	$\left(\frac{n}{\sqrt[3]{N}}, \frac{n}{\sqrt[3]{N}}, \frac{n}{\sqrt[3]{N}}\right)$
Coût de composition par nœud	n^2	$\left(\frac{n}{\sqrt[3]{N}}\right)^2$
Surcoût de composition total	$(N-1)n^2$	$\left(\sqrt[3]{N}-1\right)\left(\frac{n}{\sqrt[3]{N}}\right)^2$
Débit réseau total	$(N-1)n^2$	$\left(\sqrt[3]{N}-1\right)\left(\frac{n}{\sqrt[3]{N}}\right)^2$
Latence de composition	$(N-1)n^2$	$\left(\sqrt[3]{N}-1\right)\left(\frac{n}{\sqrt[3]{N}}\right)^2$

Nous voyons que la méthode de partition en blocs a des performances meilleures que celle de la méthode par tranches. Elle est donc particulièrement adaptée au rendu volumique sur grappe, étant donné que les systèmes d'interconnexion de type COTS sont moins performants que ceux présents sur les machines de rendu spécialisées. Nous en donnons un exemple ci-dessous. Notons qu'il existe des méthodes de partition et composition plus sophistiquées (par exemple «binary-swap »). Enfin, la méthode de partition par blocs peut induire des artefacts de rendu gênants. En effet, en rendu volumique, il y a souvent des artefacts aux pixels rendus à partir des bords du volume. Ainsi, en découpant un volume en sous-volumes rendus individuellement, il y aura des artefacts aux bords des images individuelles (on suppose que la direction de vue est orthogonale au volume de vue). Ces artefacts se retrouveront dans l'image finale. La méthode par tranche n'introduit pas de tels artefacts car le volume est découpé de manière à ce que les artefacts restent sur les bords des images partielles, et donc de l'image finale. Pour annuler le coût de composition, nous pouvons partitionner le volume en $N = n$ tranches horizontales : le coût de composition serait alors nul, mais l'image finale serait probablement de très mauvaise qualité.

Dans la sous-partie suivante, nous décrivons deux applications de rendu volumique sur architectures de type SMP.

5.2.3. Applications du rendu volumique sur architectures SMP

[MCC99] présente une méthode de rendu volumique par objet accélérée matériellement. Cette méthode est utilisée pour visualiser des résultats de simulation numérique du faisceau d'un accélérateur de particules. Le rendu volumique permet ainsi de visualiser la densité du faisceau de particules dans l'espace. Cette application utilise une méthode de rendu parallèle « sort-last » avec une architecture de traitement en pipeline (sur Origin 2000). L'Origin 2000 possède plusieurs unités de traitement graphiques : le volume est ainsi découpé verticalement en sous-volumes, et chaque unité rend une sous-partie du volume. L'image résultant de chaque unité est passé à l'unité suivante qui la compose avec sa propre image. La taille du volume rendu croît linéairement avec le nombre d'unités. Cependant, cette méthode a un temps de latence également proportionnel au nombre d'unités n. Selon les auteurs, lorsque n est grand le temps de latence devient inacceptable. Une telle méthode limite donc l'extension des performances et limite la taille de la scène pouvant être traitée, celle-ci étant de plus statique (un volume de données correspond à un pas de temps de la simulation). Les auteurs rendent des scène de taille maximum de 512^3.

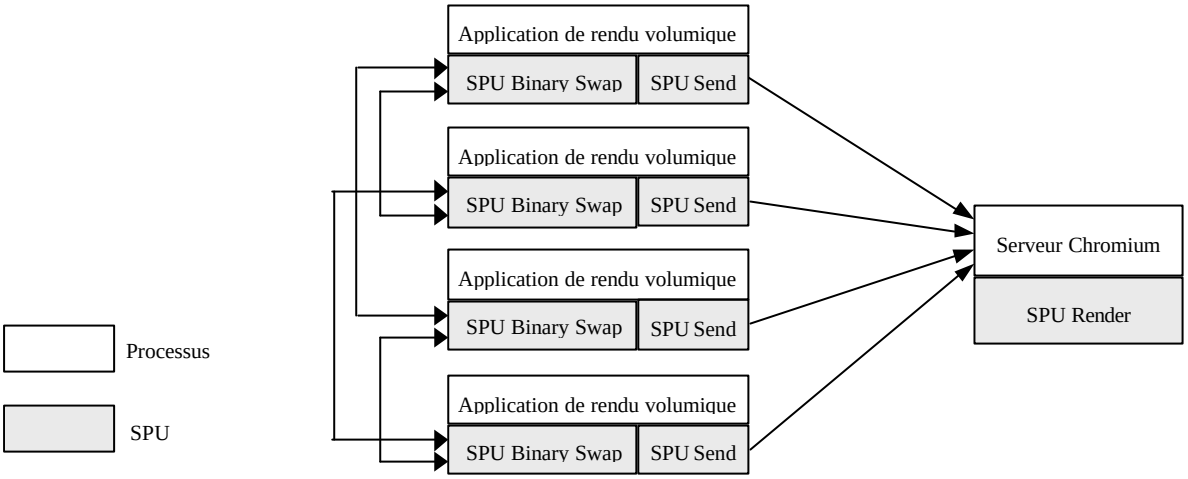
[KNI01] présente le système T-Rex, qui utilise une méthode de rendu parallèle proche de la précédente, mais permettant de traiter de jeux de données dynamiques, c'est à dire de rendre interactivement l'évolution en fonction du temps d'un jeu de données dynamiques. Selon les auteurs, l'objectif de T-Rex est de rendre des jeux de données volumiques dynamiques, c'est à dire en fonction du temps. Pour cela une vitesse d'affichage interactive est nécessaire (>= 5 Hz) et les données doivent pouvoir être chargées dynamiquement. Le système utilise une architecture en pipeline, mêlant calculs matériels et logiciels. Il est implanté sur une machine SMP Origin 2000 avec 128 processeurs et 16 accélérateurs de rendu indépendants. Les 4 étages du pipeline sont : lecture des données, rendu, composition et interface utilisateur (IU). Chaque étage est un processus multi-thread, s'exécutant simultanément avec les autres, et comprend 2 parties fonctionnelles (gestionnaire de communication et exécution réelle). Pour un volume divisé en N sous-volumes, N threads de lecture et N de rendu sont créés. Le nombre de threads de composition est fonction de la taille image finale et du nombre de sous-volumes. Un thread IU affiche l'image composée et envoie les événements utilisateurs aux étages précédents. Le rendu des sous-volumes se fait avec une méthode à base de textures 3D, gérée matériellement par l'architecture SMP utilisée. Le volume est partitionné en tranches verticales et composé de l'arrière vers l'avant. Le débit de chargement de données est de 4 Go/s, ce qui permet de rendre un volume de 1024^3 à près de 5 Hz.

Les applications de rendu volumique avec accélération matérielle ont jusqu'à récemment été principalement développées sur machines spécialisées, telles les deux applications que nous avons décrites ci-dessus. La bonne intégration des différents sous-systèmes de telles machines permet en effet d'atteindre les performances requises pour un affichage de données dynamiques de grande taille. Cependant, la progression des composants individuels des grappes de PC rend possible le développement de tels systèmes sur grappe. Nous décrivons maintenant plusieurs applications du rendu volumique sur grappe et les sous-systèmes qui y sont associés.

5.2.4. Rendu volumique parallèle sur grappes.

[HUM02] présente une application de rendu volumique sur grappe utilisant Chromium (voir 4.2.2). Elle utilise une méthode de composition de textures 3D. La Figure 28 ci-dessous représente l'architecture logicielle de l'application.

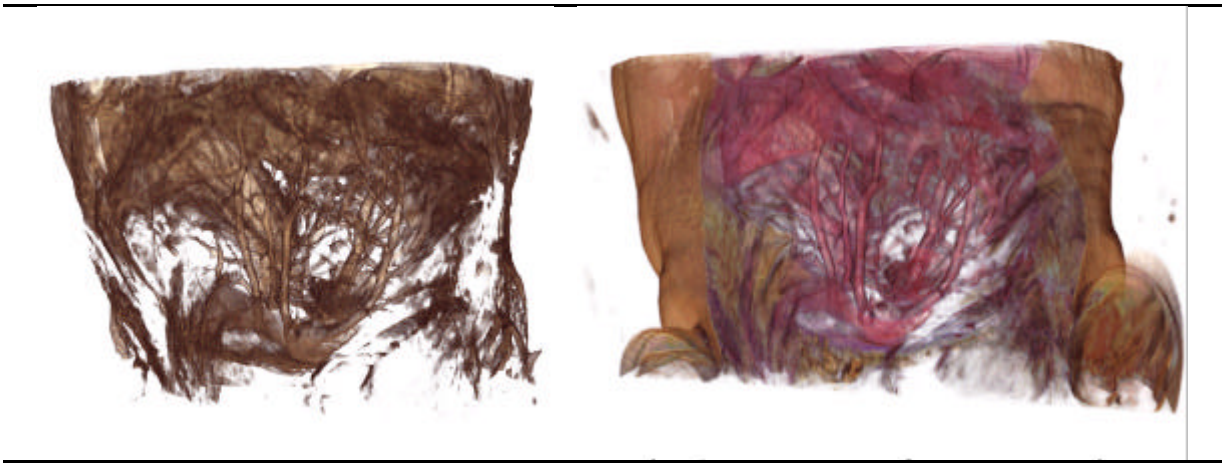
Figure 28 : application Chromium de rendu volumique



L'application utilise la méthode de composition « binary-swap » (implantée dans la SPU « Binary Swap »). Le volume est réparti également parmi les nœuds de la grappe, chaque nœud exécutant un processus de rendu volumique. Chaque nœud rend son sous-volume à l'aide de la SPU « Binary Swap » qui compose les images partielles avec la technique « binary-swap ». Son principe est le suivant : les nœuds sont groupés en paires : pour chaque paire, chacun des deux nœuds envoie la moitié de son image à son partenaire et en reçoit l'autre moitié. Pour chacun des 2 nœuds, l'image reçue est composée avec son image locale. La région composée est alors découpée en 2, un appariement différent est choisi et le processus se répète récursivement. On montre qu'avec N nœuds, après $\log(N)$ étapes de composition, chaque nœud aura composé $\frac{1}{N}$ de l'image finale. L'image finale est alors assemblée à partir des $\frac{1}{N}$ parties disjointes. Le choix de l'ordre des séquences d'appariemment est déterminant pour que la composition (qui utilise la transparence) donne une image correcte. Les auteurs présentent les résultats de cette application rendant un jeu de données biologiques (IRM de souris). Le volume a une taille de 1024 x 256 x 256, ce qui nécessite 8 nœuds (la carte graphique d'un nœud peut stocker un volume de 256 x 256 x 128). Il est rendu à la résolution de 1024 x 256, chaque voxel ne contribuant qu'à un seul pixel. L'utilisation de cartes programmables (Nvidia GeForce3) permet d'implanter plusieurs fonctions de transfert : une fonction 2D, une fonction d'isosurface, une fonction 2D avec éclairage, etc. Avec 16 nœuds, le système rend entre 0,64 Gvoxels/s et 1,59 Gvoxels/s, selon la fonction de transfert utilisée.

La Figure 29 ci-dessous (provenant de [HUM02]) montre 2 images de rendu volumique d'un jeu de données de RMN de souris (d'une taille de 256 x 256 x 1024). L'image de gauche est rendue avec une fonction de transfert de type isosurface éclairée, tandis que l'image de droite est rendue avec une fonction de transfert 2D éclairée. Notons que la fonction de transfert 2D permet de mettre en évidence différents types de tissus : on peut par exemple observer dans le centre du volume des structures tubulaires (semblant être des vaisseaux sanguins), représentées avec une couleur rosâtre, alors que les tissus plus externes ont une couleur brune.

Figure 29 : rendu volumique de RMN de souris : isosurface éclairée et fonction de transfert 2D éclairée



[MCP01] présente un système de rendu volumique parallèle sur grappe de PC. Selon les auteurs, une analyse détaillée des performances requises pour chaque composant du système est nécessaire pour obtenir un rendu à vitesse interactive. Le but d'un tel système est de rendre un volume d'un Gvoxel à 5 Hz. Ce système utilise une organisation et un algorithme proches de [KNI01], en partie des mêmes auteurs, c'est-à-dire une méthode de rendu parallèle « sort-last » avec composition :

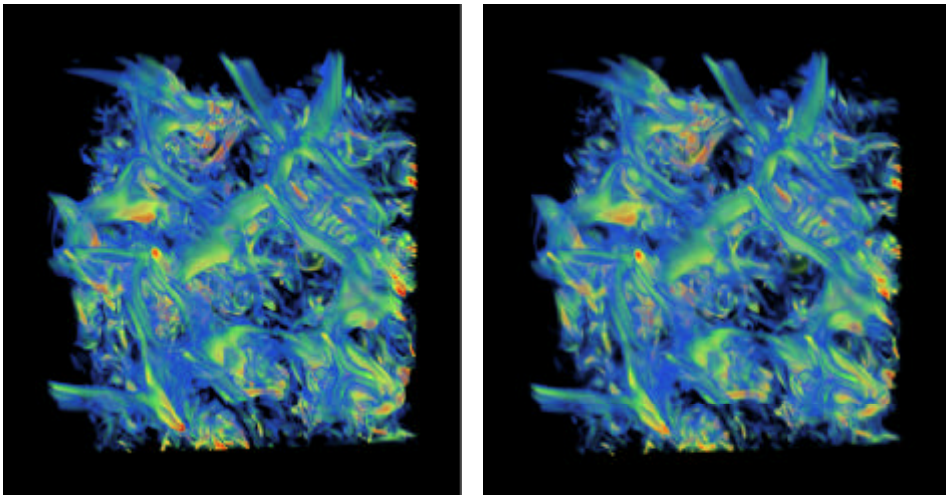
- Découpage du volume en N sous-volumes.
- Rendu de chaque sous-volume par un nœud de la grappe (PC avec carte graphique standard).
- Lecture des images rendues et composition logicielle par des nœuds interconnectés.
- Affichage dans interface utilisateur.

L'architecture du système est donc en pipeline avec trois étages distincts. interface utilisateur (IU), rendu et composition. Les auteurs montrent que l'étape limitante du pipeline est celle du transfert des images rendues des nœuds de rendu vers les nœuds de composition. Des mesures montrent qu'un lien du réseau d'interconnexion (Ethernet Gbit) permet un débit de 55 Mo/s, ce qui correspond à un débit de 11 images/s (images de 1024^2 avec 32 bits/pixel). Cependant, une image peut être transportée plusieurs fois, ainsi une vitesse de 5 images/s ne permet que 2 transports par image, ce qui peut être insuffisant. La taille de la scène pouvant être rendue dépend de la taille de la mémoire des accélérateurs graphiques, de leur vitesse de remplissage (en pixels/s) et du nombre de nœuds de la grappe. Selon les auteurs, pour des cartes ayant 128 Mo de mémoire vidéo et une vitesse de rendu de 240 Mtexels/s, il est possible de rendre un sous-volume de 512 x 512 x 256 par nœud (taille mémoire de 64 Mo avec table de couleurs), ce qui requiert 32 nœuds pour environ 1 Gvoxel (2 Gvoxels théoriquement). Cependant, les jeux de données variant en fonction du temps requièrent des vitesses de chargement élevées pour être rendus interactivement : chaque nœud (pour un sous-volume de 64 Mo) requiert une vitesse de chargement en mémoire graphique (via le bus système AGP) de 320 Mo/s, ce qui est juste car le bus est utilisé pour relire les images partielles rendues. De plus, le débit de lecture de chaque pas de temps des données doit être aussi élevé : cela requiert un débit d'E/S agrégées de 320 x 32 = 10 Go/s, ce qui est rédhibitoire vu que les systèmes d'E/S pour PC sont difficilement extensibles.

Cependant, il existe des méthodes pour contourner ce problème. [LUM02] présente un système de rendu volumique sur grappe qui utilise une méthode de compression de texture avec perte pour visualiser interactivement des jeux de données volumiques variant en fonction du temps. Il utilise une méthode de rendu par texture 2D avec un découpage en tranches verticales « back-to-front », chaque nœud gérant une tranche. Une méthode de compression de texture (durant une étape de prétraitement) permet de stocker dans la mémoire graphique de chaque nœud un volume de taille plus importante. La compression est effectuée dans le domaine temporel : N pas de temps (N étant ajustable) sont compressés en 1 seul volume. Le codage d'une séquence de N images se fait par DCT (« Discrete Cosine Transform »), algorithme avec perte. Pour chaque point du volume, une fenêtre de N valeurs temporellement consécutives donne une seule valeur après codage. Cette valeur scalaire est

stockée comme indice dans une texture palettisée. Chaque entrée de palette est reliée à la valeur moyenne du scalaire que représente l'indice pour chaque pas de temps. Cette séquence est décodée vers ses valeurs variant temporellement par changement de palette au cours du temps. Le découpage « back-to-front » utilisé ne requiert pas la réplication des voxels aux frontières, mais a cependant un coût de composition élevé. La composition se fait logiciellement avec la méthode « binary-swap », les images rendues étant compressées en temps réel avec une méthode RLE pour minimiser le débit réseau. Les auteurs ont également implanté une méthode « out-of-core » quand le volume de données ne tient pas en mémoire centrale agrégée. L'étape limitante est alors celle de lecture des données sur disque, ce que montrent les mesures des temps des étages du pipeline effectuées par les auteurs. Cependant la méthode de rendu « out-of-core » permet de rendre un volume de données dynamiques de 512^3 à environ 3 im/s sur 8 nœuds (contre 4,9 im/s dans le cas «in-core »). La Figure 30 ci-dessous montre la visualisation par rendu volumique d'un pas de temps d'une simulation de convection solaire turbulente d'une taille de 512^3 et comprenant 200 pas de temps (voir [LUM02]). Chacune des 2 images est rendue à la résolution de 512 x 512. L'image de gauche a été rendue sans compression tandis que l'image de droite a été rendue avec un facteur de compression de 8.

Figure 30 : rendu volumique d'une simulation de convection solaire turbulente



5.3. Discussion

Les différentes applications que nous avons décrites en 5.1.2 et 5.2.4 sont pour la plupart à un stade de recherche et développement. La totalité des applications de rendu surfacique que nous avons présentées utilisent des jeux de données statiques. [SHA02] décrit un système utilisant une grappe de PC uniquement pour la partie de détermination des données visibles, le rendu étant effectué par une machine SMP. [WYL01] décrit un système rendant des isosurfaces de très grande taille avec un rendu parallèle « sort-last ». Cette méthode permet un très bon équilibrage de charge. Cependant, elle limite la résolution du rendu. Ainsi, pour les jeux de données que les auteurs utilisent (de l'ordre de 350 Mtriangles), un grand nombre de polygones contribue à chaque pixel d'une image. Il n'est donc pas possible de visualiser la totalité des données avec un grand niveau de détail, ce que permet un rendu parallèle « sort-first » avec un affichage sur écran en mosaïque. Cependant un système parallèle « sort-first » est plus sensible au déséquilibre de charge. Enfin, un système de rendu surfacique avec parallélisme « sort-last » est sujet aux problèmes de précision du tampon de profondeur des cartes COTS. En effet, sur la plupart des cartes, le tampon de profondeur a une précision de 24 bits. Cela peut être insuffisant à partir d'une certaine taille de données, et entraîner des problèmes de chevauchement et de visibilité incorrecte.

Dans les applications de rendu volumique que nous avons décrites, [LUM02] est la seule qui supporte le rendu volumique de jeux de données dynamiques, grâce à l'utilisation conjointe de méthodes « out-of-core » et de

compression de textures. Les limites intrinsèques de l'architecture des PC rendent plus complexe la mise en place d'un tel système, par rapport aux architectures spécialisées où tous les composants parallèles sont fortement intégrés (systèmes d'E/S, sous-système graphique, etc.), voir [KNI01]. Le développement de méthodes logicielles permettant de bien exploiter la capacité agrégée d'une grappe est probablement nécessaire, vu l'augmentation constante de la taille de jeux de données générés par les simulations numériques, pour pouvoir rendre les données dynamiques. A ce titre, l'utilisation de méthodes « out-of-core » semble incontournable. Nous nous interrogeons sur la validité des méthodes de compression : bien que [LUM02] prétend le contraire, il est possible qu'une méthode de compression de textures nuise trop à la qualité du rendu. L'apparition de cartes graphiques COTS programmables peut potentiellement ouvrir toute une gamme d'applications de rendu volumique aux grappes.

Cependant, nous pensons que la généralisation de bibliothèques de haut niveau est nécessaire pour permettre une utilisation simple de ces caractéristiques. En effet, la programmation d'une fonction de transfert (au moyen d'un langage d'assemblage souvent spécifique à un modèle particulier de carte) nécessite des connaissances dépassant celles des utilisateurs. La généralisation de bibliothèques de langages graphiques évolués pourrait permettre la généralisation de telles caractéristiques et ainsi faciliter le développement d'applications qui permettraient à d'offrir à l'utilisateur une grande souplesse de maniement tout en restant accessibles. En conclusion, nous pensons que les applications du rendu volumique sur grappe sont potentiellement vastes : imagerie médicale, visualisation de simulations numériques, etc., mais que les applications existantes semblent encore à un stade de développement plutôt que d'exploitation industrielle. Nous recensons dans l'annexe C les caractéristiques des différentes applications que nous avons décrites, pour les rendus surfacique et volumique.

6. Conclusion sur le rendu parallèle sur grappes de PC

Les composants COTS ne cessent de progresser. L'augmentation des performances des GPU des cartes graphiques, des microprocesseurs et de la mémoire des PC ne semblent pas devoir faiblir à très court terme. [HOU02] liste ainsi trois configurations de grappes de rendu parallèle utilisant les composants les plus récents qui soient (à la date d'écriture du document). Au moment où nous rédigeons ce document, de nouveaux modèles de cartes graphiques sont apparus. Citons la Radeon 9800 d'ATI et la GeForce FX 5900 de Nvidia, qui offrent des performances encore supérieures aux cartes décrites dans [HOU02]. La croissance exponentielle des performances des cartes graphiques (débit de triangles rendus/s et de pixels rendus/s) renforce à notre avis la viabilité des solutions de rendu sur grappe : depuis cinq ans, cette croissance est de l'ordre de la loi de Moore au cube, soit un doublement tous les six mois. Il reste cependant de la marge de progression avant qu'un PC unique soit capable d'effectuer la visualisation d'une scène dynamique de très grande taille. En effet, comme les besoins en visualisation sont virtuellement illimités, il est probable qu'un seul PC ne soit capable d'y répondre pendant encore plusieurs années. L'utilisation d'une grappe nous semble obligatoire à moyen terme. L'existence de fonctions programmables sur les cartes COTS pourrait devenir intéressante, particulièrement en ce qui concerne le rendu volumique. En effet, cela permettra de programmer et de faire exécuter des algorithmes sophistiqués (fonctions de transfert complexes, illumination, etc.) ce qui demandait auparavant un matériel spécialisé, tel la carte de rendu volumique VolumePro, et avec une flexibilité a priori moindre. Sur ce point précis, le développement de fonctionnalités de texturage 3D rendrait les cartes standard d'autant plus intéressantes pour le rendu volumique.

Comme nous l'avons vu en 3.2.2.3, un système d'affichage approprié est nécessaire pour visualiser correctement des jeux de données de très grande taille. Comme la finalité d'un système de visualisation scientifique est de présenter un rendu visuel à des observateurs humains, les caractéristiques du système visuel humain contraignent les performances utiles des systèmes d'affichage (nombre de pixels visibles par degré du champ visuel à une certaine distance d'observation). [DEE98] présente un modèle basique donnant le nombre de pixels nécessaire pour saturer le système visuel d'un observateur humain idéal. L'auteur estime que le système visuel humain peut percevoir environ 15 Mpoints, dans un champ visuel égal au tiers de celui d'une sphère. Nous pouvons donc penser qu'un système d'affichage pouvant afficher 15 Mpixels est suffisant. Cependant, le nombre de pixels d'un écran réellement perceptibles dépend de la distance à laquelle se trouve l'observateur. La résolution apparente (séparation angulaire entre 2 points) dépend donc de la distance d'observation. Or, en visualisation scientifique haute résolution, plusieurs observateurs peuvent regarder en même temps l'écran, et à des distances différentes (pour observer des détails fins, ou avoir une vue plus globale de la scène rendue). Nous pensons donc que la résolution nécessaire d'un système d'affichage devrait dépasser les 15 Mpixels, pour correspondre à la résolution de la scène visualisée (qui pourrait éventuellement comprendre des centaines de millions de primitives). Pour l'instant, seuls les systèmes d'affichage par mosaïque permettent d'atteindre des résolutions de l'ordre de 10 Mpixels avec des surfaces d'affichage de grande taille (plusieurs m^2). Il est également probable (voir [HER00a]) que les systèmes d'affichage à surface continue (par exemple des écrans souples à base de technologie OLED) ne soient disponibles qu'à moyen terme, du fait des contraintes technologiques actuelles. L'utilisation d'afficheurs à pavage (notamment ceux à base de projecteurs) semble se développer pour les années à venir.

Il semble donc que le domaine du rendu sur grappe soit encore dans une phase d'émergence. Les performances des PC ne sont devenues acceptables que depuis quelques années (environ 5 ans). Pour l'instant les systèmes existants sont plutôt développés dans des laboratoires de recherche (voir [WYL01], [MCP01]) . Les systèmes commerciaux disponibles paraissent encore peu nombreux, comparativement aux architectures SMP qui sont disponibles à l'échelle « industrielle » depuis relativement longtemps. De même les composants logiciels permettant d'effectuer du rendu parallèle sur grappe sont encore à l'état de développement. Par exemple, Chromium (voir [CHR03]) est un projet « open-source » en cours de développement, bien qu'il semble utilisé par plusieurs laboratoires pour piloter des grappes de visualisation. Les applications scientifiques du rendu sur grappe semblent donc encore en émergence, et il est possible que de nombreux travaux restent à faire dans ce domaine avant d'atteindre les performances d'une application de rendu volumique telle que [KNI01].

7. Perspectives

Pour les travaux futurs, nous nous intéresserons particulièrement à l'optimisation d'applications (optimisation logicielle) pour les systèmes de rendu (et plus généralement de visualisation) distribué. Rappelons que les données typiques des calculs du CEA/DIF ont les caractéristiques générales suivantes :

- Représentation de phénomènes instationnaires, avec couplages de modèles physiques non linéaires.
- Grande dynamique (fortes variations numériques) dans l'espace et le temps.

Maillages très détaillés et grand nombre de pas de temps de sauvegarde des résultats en découlent. De très gros volumes de données sont donc à traiter après stockage.
Ces caractéristiques posent des contraintes fortes sur l'architecture des systèmes de visualisation que l'on peut utiliser. Nous présentons les grands axes d'étude que nous projetons de suivre afin de pouvoir répondre à ces contraintes.
Premièrement, nous nous plaçons dans un contexte de compatibilité avec OpenGL. En effet, les composants logiciels de visualisation parallèle existants (VTK, et autres comme Ensight, OpenDX, etc.) utilisent tous OpenGL pour le rendu, qui est un standard de fait. Plus spécifiquement, les composants de rendu parallèle existants se basent également sur OpenGL (tel Chromium). L'utilisation d'un autre modèle de rendu, bien que possible, n'est pas envisagée à l'heure actuelle.
Deuxièmement, nous choisissons les approches de parallélisme utilisant un réseau d'interconnexion rapide, telles les approches décrites par Molnar et al. Il existe des approches de rendu parallèle, que nous avons décrites en 4.4, utilisant un mode de parallélisme de type réplication des données avec un réseau d'interconnexion à faible débit. Cependant, nous considérons que ces approches ne sont actuellement pas compatibles avec les problèmes que nous voulons traiter. En effet, la taille importante des données de ces problèmes nécessite l'utilisation d'une capacité de traitement agrégée très importante. Cela n'est pas possible lorsque la base de données est répliquée sur chacun des nœuds. Afin d'interconnecter les nœuds stockant la base des données, et effectuant les calculs de pré-rendu, avec les nœuds chargés du rendu, l'utilisation d'un réseau rapide est donc nécessaire.
Troisièmement, nous allons adopter une approche par analyse dimensionnante. Les différentes applications que nous avons décrites en 5.1 et 5.2 sont des solutions « ad hoc » : chacune de ces applications utilise une méthode spécifique au problème qu'elle résout (type de données). Les applications existantes sont donc trop spécifiques aux problèmes qu'elles traitent. Nous choisissons donc d'utiliser l'analyse dimensionnante pour, en fonction des différents ensembles de valeurs paramètres de la chaîne de visualisation (taille des données, mode de parallélisme, algorithmes de rendu et de pré-rendu), déterminer de manière plus générale des catégories d'applications auxquelles ces paramètres peuvent s'appliquer.
Quatrièmement, les performances logicielles sur les systèmes distribués offrent encore une grande marge de progression. Voici les pistes que nous pensons explorer pour améliorer ces performances.

- Utilisation de méthodes « out-of-core » : les volumes de données que l'on veut traiter ne tiennent pas en mémoire physique. A ce titre, l'utilisation de méthodes de rendu (voire pré-rendu) « out-of-core » parait nécessaires. [COR02] et [LUM02] présentent un aperçu prometteur des capacités des méthodes « out-of-core ». Il reste à voir s'il est possible d'étendre ces méthodes à de plus gros volumes de données, sur grappes.
- Traitement parallèle à toutes les étapes : l'utilisation d'un traitement parallèle à toutes les étapes de la visualisation est nécessaire pour atteindre des performances optimales. Ainsi, [ZHA01] utilise une étape de pré-rendu parallèle, mais pas une étape de rendu parallèle, ce qui limite potentiellement les performances globales. L'utilisation du parallélisme à toutes les étapes du pipeline de visualisation est nécessaire pour maximiser son débit.
- Utilisation de modes hybrides de parallélisme : les méthodes de rendu parallèle SF et SL sont respectivement limitées par la taille de la scène et la résolution. Une méthode de rendu parallèle hybride pourrait avoir leurs avantages tout en réduisant leurs inconvénients respectifs. Les travaux de ce type déjà développés ont l'inconvénient d'utiliser une étape de pré-rendu séquentielle (en plus de la réplication de la base de données), ce qui est fortement limitant dans les cas que l'on veut traiter. Un mode hybride se basant sur des composants matériels supplémentaires (« compositeurs », voir 3.4 et 3.5) ou utilisant une étape de pré-rendu totalement parallèle pourrait résoudre ce problème.

L'utilisation d'un traitement parallèle à tous les niveaux du pipeline de visualisation (entrées/sorties parallèles, pré-rendu et rendu) nous semble donc nécessaire pour maximiser les performances d'une grappe de visualisation. L'étude de la répartition des traitements parallèles à tous les niveaux de la chaîne, et d'abord au niveau du rendu,

nous paraît donc la clé de l'amélioration des performances d'un système de visualisation distribuée. Nous avons vu des composants logiciels et mêmes des infrastructures offrant les mécanismes parallèles nécessaires pour le rendu ou le pré-rendu (p.ex. Chromium: cf 4.2 ; Parallel VTK : cf 4.3.1). Appliquer ces éléments à la visualisation de gros volumes de données variant dans le temps exigera un effort dans la méthodologie de dimensionnement et d'intégration matérielle et logicielle, ainsi que des améliorations dans les algorithmes eux-mêmes, suivant la nature des données et des techniques de visualisation.

8. Annexe A : Rappels sur les principes et algorithmes du rendu graphique

Dans cette annexe, nous rappelons les concepts fondamentaux de l'informatique graphique en relation avec le rendu 3D interactif. Une des références classiques en la matière est [FOL90].
Tout d'abord nous définissons le processus du rendu et les deux grands types de méthodes qu'il utilise. Ensuite, nous présentons la manière dont les couleurs des surfaces des objets rendus sont déterminées. Enfin, nous décrivons le pipeline graphique utilisé en rendu 3D interactif. Nous détaillons alors les deux phases consécutives qui le composent.

8.1. Définition du rendu

Nous définissons le rendu graphique (ou rendu) comme le processus consistant à calculer une image (bidimensionnelle) à partir de la description d'une scène généralement tridimensionnelle, contenant un ensemble d'objets (ou primitives). On répartit généralement les différentes méthodes de rendu en deux grandes catégories, suivant l'espace dans lequel elles opèrent.

Premièrement, les méthodes de rendu dans l'espace image (ou rendu par pixel), itèrent sur tous les points (pixels) constituant l'image à rendre et déterminent pour chaque pixel sa couleur, généralement définie par un triplet de couleurs (R,V,B).

```
Pour chaque pixel p(i, j)
      Determiner la couleur de p(i, j)
```

Deuxièmement, les méthodes de rendu par objet parcourent la liste des objets graphiques (ou primitives) constituant la scène, et pour chaque objet, calculent l'ensemble de pixels par lequel il contribue à l'image.

```
Pour chaque objet o(i)
      Determiner l'ensemble de pixels de o(i)
```

A l'heure actuelle, les deux méthodes les plus utilisées pour ces deux grandes catégories respectives, sont respectivement le lancer de rayons (« ray-casting ») et les méthodes projectives.
Le principe du lancer de rayon est le suivant : à chaque pixel p(i, j) de l'image est associé un rayon r(i, j). Une grille 2D g, placée dans l'espace de la scène, est associée à l'image finale, à chaque nœud de la grille g(i, j) correspond le pixel p(i, j). Le rayon r(i, j) est une demi-droite dont l'origine est le point g(i, j). L'algorithme du lancer de rayon est alors le suivant :

```
Pour chaque rayon r(i, j)
      Determiner c, la contribution a p(i, j) de tous les objets de S
      traverses par r(i, j)
      Affecter c a p(i, j)
```

Le lancer de rayon est généralement implanté de manière logicielle, bien qu'il existe des architectures matérielles y étant dédiées. On l'utilise en synthèse d'image réaliste (variantes du lancer de rayon récursif, « path tracing », etc.), en visualisation scientifique (rendu volumique à base de lancer de rayon), etc.

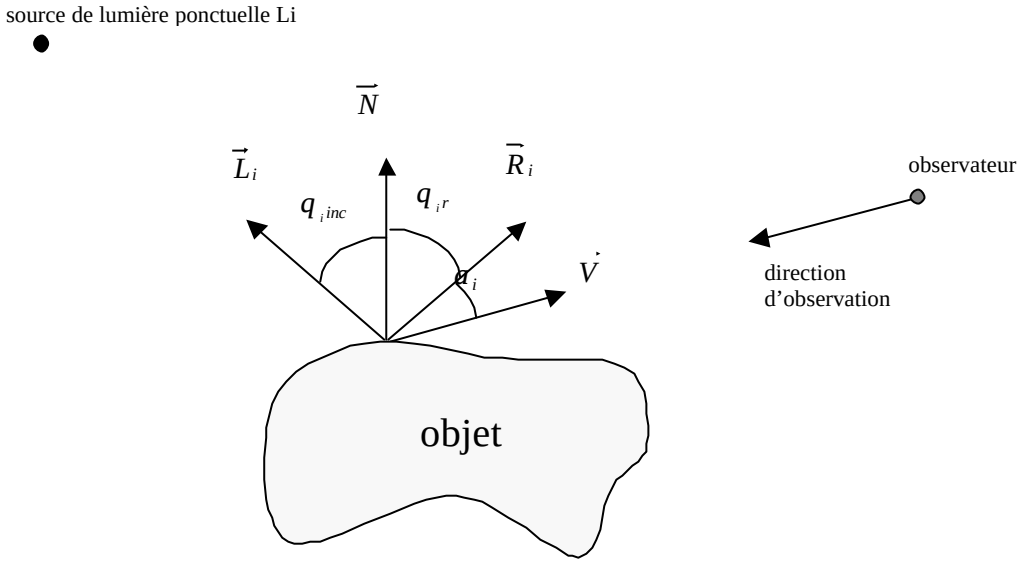
8.2. Eclairage

8.2.1. Modèle d'illumination de Phong

En informatique graphique, on représente généralement la couleur de la surface d'un objet à l'aide d'un modèle colorimétrique. Ainsi, à chaque primitive, on associe un ensemble de coefficients (généralement trois) compris entre 0 et 1 et définissant la manière dont la surface réfléchit la lumière. Le modèle le plus utilisé est le modèle RVB additif. Il se base sur la perception visuelle humaine, qui se fait à l'aide de trois types de cellules rétiniennes

sensibles à la couleur, chacun spécialisé dans une partie du spectre visible. Ainsi le modèle RVB considère que n'importe quelle couleur pouvant être synthétisée par un appareil électronique d'affichage (moniteur d'ordinateur) peut se définir à l'aide de 3 coefficients R,V, et B.
La couleur d'un pixel échantillonnant la surface projetée d'un objet se calcule en déterminant l'intensité de la lumière réfléchie au point de la surface de l'objet. Pour cela on utilise un modèle d'illumination. Le modèle le plus couramment utilisé est celui de Phong. Il considère que la lumière réfléchie par une surface se décompose en deux termes : la réflexion diffuse, et la réflexion spéculaire. La réflexion diffuse (ou Lambertienne) considère que la quantité de lumière réfléchie en un point est la même quelle que soit la direction de réflexion. La quantité de lumière réfléchie en un point d'une surface dans une direction donnée dépend seulement de l'angle de la source lumineuse avec la normale à la surface en ce point. La réflexion spéculaire dépend quant à elle de l'angle que fait l'observateur avec la direction de réflexion. La Figure 31 ci-dessous représente la surface d'un objet et la réflexion en un point de cette surface.

Figure 31 : illumination locale par une source lumineuse ponctuelle



- $\vec{L}_i$ vecteur unitaire donnant la direction vers la i^{ème} source lumineuse à partir de p.
- $\vec{N}$ normale à la surface en p.
- $\vec{R}_i$ vecteur unitaire donnant la i^{ème} direction de réflexion à partir de p.
- $\vec{V}$ vecteur unitaire donnant la direction de l'observateur à partir de p.
- $q_{i\,inc}$ angle d'incidence de la lumière Li au point p.
- $q_{i\,r}$ angle de réflexion de la lumière en p.
- a_i angle entre $\vec{V}$ et $\vec{R}_i$.

L'équation décrivant le modèle d'illumination de Phong est alors, pour plusieurs sources de lumière :

$$I_l = I_{l\,a} k_{l\,a} + \sum_i I_{l\,i} k_{l\,d} \left(\vec{N} \cdot \vec{L_i}\right) + I_{l\,i} k_{l\,s} \left(\vec{R_i} \cdot \vec{V}\right)^n$$

soit

$$I_l = I_{l\,a} k_{l\,a} + \sum_i I_{l\,i} k_{l\,d} \cos q_{i\,inc} + I_{l\,i} k_{l\,s} \cos^n a_i$$

avec :

- I_l : intensité de la lumière réfléchie à la longueur d'onde l .
- I_{la} : intensité de la lumière ambiante.
- k_{la} : coefficient de réflexion ambiante de l'objet à la longueur d'onde l .
- I_{li} : intensité de la source lumineuse Li à la longueur d'onde l .
- k_{ld} : coefficient de réflexion diffuse de l'objet à la longueur d'onde l .
- k_{ls} : coefficient de réflexion spéculaire de l'objet à la longueur d'onde l .
- n : exposant spéculaire.

Bien que l'intensité de la lumière réfléchie soit définie pour une longueur d'onde l particulière, en pratique, elle n'est calculée que pour 3 pseudo-longueurs d'onde, ce qui permet de calculer un triplet (R,V,B) d'intensité, soit (I_R, I_V, I_B).
Tout d'abord, le modèle (empirique) de Phong permet de modéliser des réflecteurs spéculaires dit imparfaits, dont la surface n'est pas uniformément lisse, par opposition à un miroir poli, dit réflecteur parfait. Il considère que l'intensité de la lumière réfléchie de manière spéculaire par un objet décroît suivant une fonction (cosinus) de l'angle entre l'observateur et la direction de réflexion idéale. L'exposant spéculaire n permet de moduler la rapidité de décroissance : plus n est grand, plus l'angle de réflexion autour de la direction de réflexion idéale est petit et plus la tâche spéculaire (tâche brillante que l'on observe à la surface de l'objet) est fine.

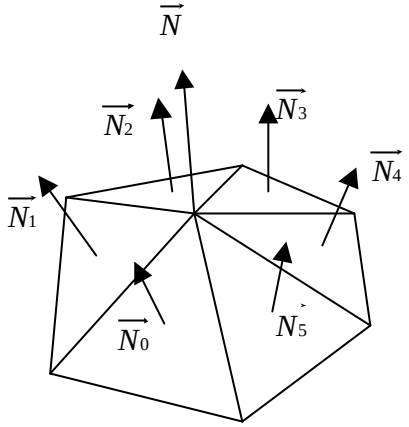
Deuxièmement, l'intensité de la lumière réfléchie en tout point d'un objet ne dépend que des sources lumineuses et des propriétés intrinsèques de l'objet (coefficients diffus et spéculaires). Elle ne dépend pas des autres objets constituant la scène. Les réflexions inter-objets ne sont donc pas prises en compte. Un tel modèle d'illumination est donc désigné comme étant local. Des modèles d'illumination prenant en compte les réflexions inter-objets existent (notamment dans les méthodes de la radiosité ou du « path-tracing »), ce sont des modèles dits globaux.

Notons qu'en visualisation scientifique, on utilise généralement pour le rendu surfacique un modèle local de type Phong. En effet, on cherche à privilégier la simplicité du rendu et sa bonne lisibilité par l'observateur humain par rapport à un réalisme qui peut nuire à la compréhension de l'image finale et qui n'est pas forcément significatif. De plus, un tel modèle s'implante très facilement en logiciel ou en matériel.
Notons qu'en rendu temps réel, on utilise généralement des modèles d'illumination locaux. Ainsi, les calculs d'éclairage des primitives d'une scène sont indépendants et peuvent facilement s'exécuter en parallèle sur des unités de rendu différentes.

8.2.2. Représentation des surfaces courbes

En informatique graphique, les objets surfaciques sont généralement représentés par des maillages de triangles. Chaque triangle est défini par trois sommets et un vecteur normal, constant en tout point de sa surface. Afin de représenter les surfaces courbes (c'est-à-dire des surfaces dont la normale en tout point de la surface varie de manière continue), on utilise le lissage interpolé (« interpolated smooth shading »). Pour cela, on calcule pour tout triangle une normale au sommet (« vertex normal ») à chacun de ses trois sommets. Pour chacun des trois sommets, ce vecteur est calculé en faisant la somme des normales des triangles partageant le sommet. Le vecteur ainsi calculé est alors normalisé.

Figure 32 : calcul des normales aux sommets



Pour chaque sommet, le vecteur normal est utilisé lors des calculs d'éclairage pour déterminer la couleur associée à tout sommet.

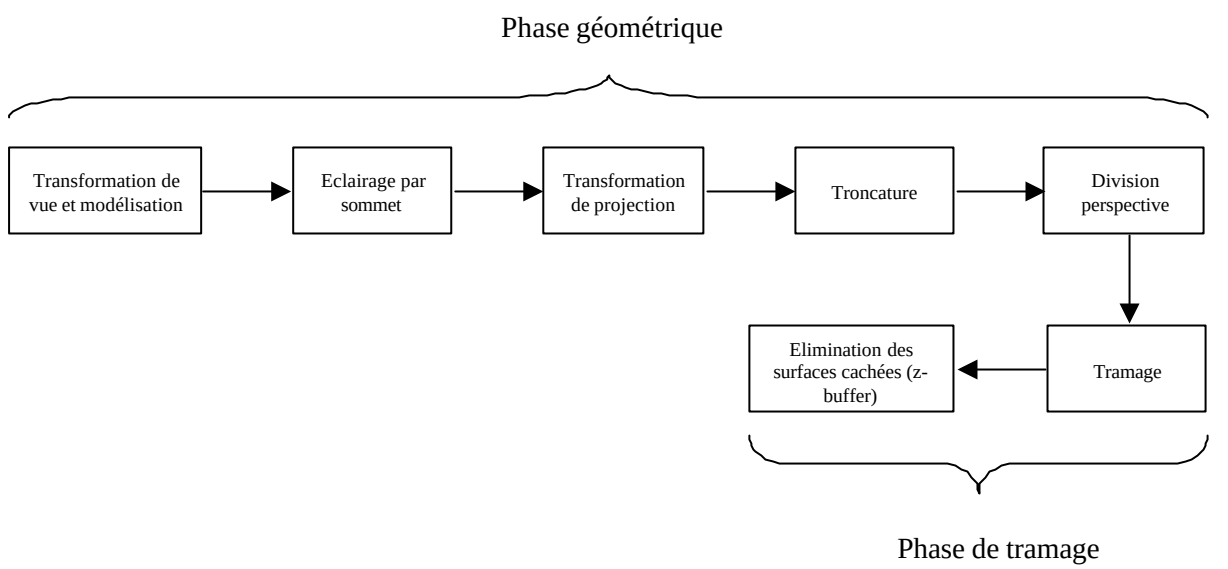
Dans la sous-partie suivante, nous décrivons le pipeline graphique utilisé en rendu temps réel et qui est à base de méthodes projectives. Nous présentons tout d'abord son principe et les deux phases consécutives qui le composent. Puis nous décrivons de manière détaillée ces deux grandes phases : les transformations géométriques et les opérations de tramage.

8.3. Pipeline graphique

8.3.1. Présentation

Les méthodes projectives sont utilisées en rendu temps réel, et notamment en rendu surfacique pour la visualisation scientifique. Elles sont maintenant implantées dans tous les accélérateurs de rendu grand public pour PC. De plus, toutes les bibliothèques graphiques utilisées en rendu temps réel implantent ce pipeline, avec quelques variations : par exemple OpenGL. Pour cette raison, nous choisissons donc de nous focaliser sur son fonctionnement.
Le rendu par méthode projective se déroule en deux grandes phases consécutives. Tout d'abord une phase géométrique durant laquelle les primitives de la scène (représentées dans un espace global) sont transformées dans un espace normalisé et projetées sur un plan 2D. Puis une phase de tramage, durant laquelle les primitives projetées sont discrétisées en pixels qui contribuent à l'image rendue finale.
Ces deux phases consécutives constituent ce que l'on appelle le pipeline graphique. Nous décrivons ci-dessous ce pipeline de manière détaillée.
La Figure 33 ci-dessous représente les différents processus du pipeline graphique dont l'enchaînement effectue le rendu d'une image.

Figure 33 : pipeline graphique



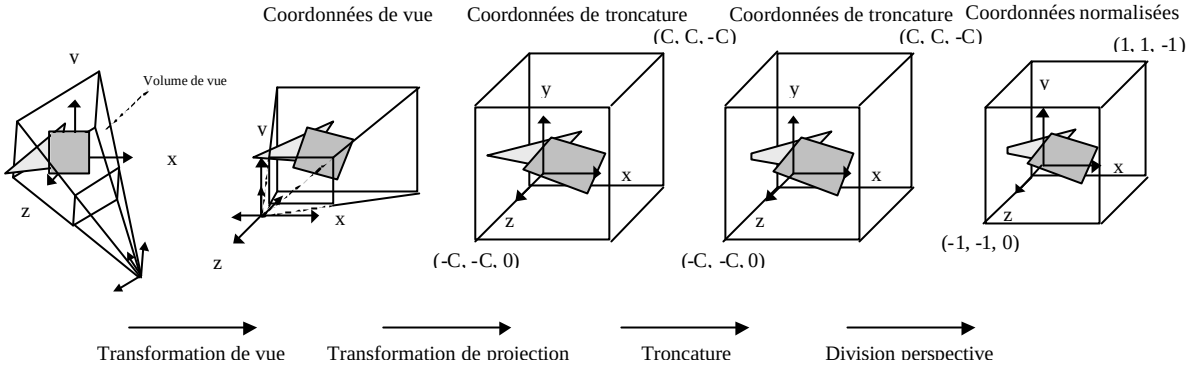
8.3.2. Transformations géométriques

Chaque point est représenté par un vecteur (x,y,z,w). La dernière coordonnée est notamment utilisée pour la projection perspective (division) et est généralement fixée initialement à 1 pour tous les points.
Les transformations géométriques (translations, rotations, homothéties) sont effectuées en multipliant successivement chaque point (représenté par un vecteur 4D) par une matrice carrée 4x4. Un volume de vue permet de sélectionner les primitives contribuant à l'image finale : il est défini par son orientation dans l'espace global, 6 plans le délimitant et sa direction de projection. Nous décrivons ci-dessous les phases successives des transformations géométriques.

- Transformation de vue et de modélisation :
  Initialement, les primitives graphiques sont représentées dans l'espace 3D global. La transformation de modélisation permet de placer les différentes primitives dans cet espace global. La matrice de vue, associée à la caméra, définit la position et l'orientation de celle-ci dans l'espace global. Chaque point est alors transformé par la matrice de vue. Cette opération transforme les sommets de l'espace global dans l'espace de l'observateur (caméra) : c'est la transformation de vue.
- Eclairage par sommet
  Après la transformation de vue, pour chaque sommet une valeur d'éclairage est calculée. L'intensité de la lumière est calculée pour chaque sommet en utilisant un modèle d'illumination locale de type Phong (voir ci-dessus).
- Transformation de projection
  Le volume de vue de la caméra est transformé par une matrice de projection. Cette matrice dépend du type de projection utilisé, soit une projection de type perspective soit orthographique. Après cette transformation, le volume de vue est un volume parallélépipédique.
- Troncature
  Les primitives se trouvant hors du volume transformé sont éliminées du pipeline. Les primitives intersectant les plans délimitant le volume sont tronquées par ceux-ci. Les valeurs d'éclairage doivent être calculées à chaque nouveau sommet créé.

- Division perspective
  Les coordonnées de chaque sommet sont divisées par la quatrième coordonnée (w). Chaque sommet a alors les coordonnées (x/w, y/w, z/w, 1). Cela revient à effectuer une normalisation du volume de vue transformé.

Figure 34 : transformations géométriques

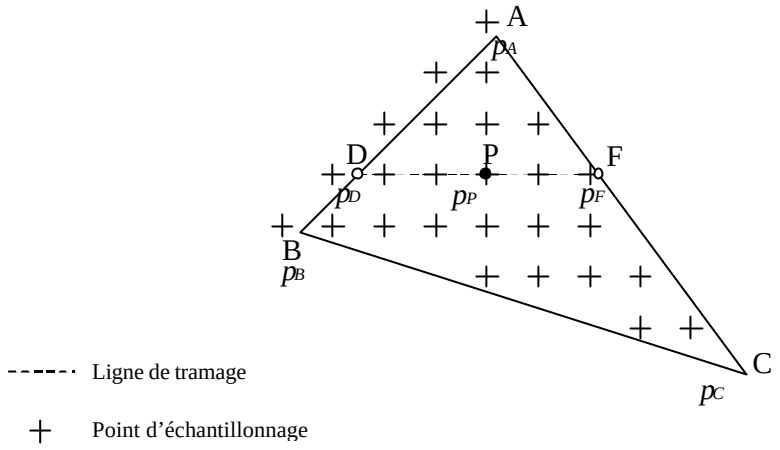


8.3.3. Tramage

Chaque primitive transformée (primitive d'espace-écran) est discrétisée en un ensemble de fragments, un fragment étant un pixel avec des informations supplémentaires (transparence, valeur de profondeur, coordonnées de texture, etc.). Notons qu'avant le tramage, les coordonnées (x,y) de la primitive sont transformées par homothétie pour que l'aspect de la zone de tramage (les deux plans Z=cte du cube de vue après division perspective sont des carrés) soit le même que celui de la fenêtre graphique dans lesquelles les primitives sont dessinées (c'est la transformation de fenêtre). Chaque fragment a des coordonnées (i, j) dans l'image finale et un point (x,y,z) sur la surface de la primitive qu'il échantillonne lui est associé. La valeur de profondeur z est généralement convertie en entier positif. Des plus des valeurs de couleur et de texture calculées par interpolation des valeurs des sommets de la primitive peuvent être associées au fragment. Notons qu'une texture est une image associée à une primitive et qui plaquée sur la surface de celle-ci permet d'apporter des informations visuelles supplémentaires (c'est le « texture-mapping »).

Il existe deux méthodes principales pour calculer la couleur d'un fragment : le lissage de Gouraud ou le lissage de Phong. Le lissage de Gouraud (« Gouraud smooth shading ») consiste à calculer avant le tramage les couleurs associées à chaque sommet de la primitive en appliquant le modèle d'illumination ci-dessus. La couleur d'un fragment f(i, j) est alors déterminée par interpolation linéaire des couleurs aux sommets.
Le lissage de Phong (« Phong smooth shading ») consiste à calculer le vecteur normal en chaque fragment f(i,j) par interpolation linéaire des normales aux sommets de la primitive. Le modèle d'illumination est alors appliqué à chaque fragment en utilisant la normale interpolée. La Figure 35 ci-dessous représente la manière dont un triangle est discrétisé en un ensemble de pixels.

Figure 35 : tramage d'un triangle



Chaque fragment échantillonne la projection dans l'espace image d'une primitive : ainsi à un fragment P(i, j), on fait correspondre un point p(x, y, z) dans l'espace normalisé (coordonnées normalisées). Comme les primitives sont en général représentées par leurs sommets, à chaque sommet correspond un ensemble de propriétés (position, couleur, normale, transparence, profondeur, etc…). Le tramage d'une primitive dans l'espace image s'effectue généralement dans un ordre précis, par exemple de haut vers le bas et de gauche vers la droite. Pour chaque pixel, on peut calculer la propriété p(i, j) par interpolation linéaire. Dans la Figure 35, les deux points D et F sont deux points de la primitive, se projetant aux extrémités d'une ligne de tramage et intersectant les arêtes projetées de la primitive. Au fragment P(i, j), échantillonnant la primitive au point P, on peut calculer la propriété p_P par interpolation linéaire des propriétés des deux points aux intersections de la ligne de tramage avec les arêtes (projetées) de la primitive. Leurs propriétés sont elles-mêmes calculées par interpolation linéaire des propriétés aux sommets le long des arêtes de la primitive (cas d'un triangle) ;

$$p_p = lerp(D, F, P, p_D, p_F)$$
$$p_D = lerp(A, B, D, p_A, p_B)$$
$$p_F = lerp(A, C, F, p_A, p_C)$$

avec :

$$lerp(A, B, P, p_A, p_B) = p_A + (p_B - p_A)\frac{P - A}{B - A}$$

L'ensemble des propriétés associées aux sommets et utilisées lors du tramage contient la couleur (pour le lissage interpolé de Gouraud), le vecteur normal (pour le lissage de Phong), les coordonnées normalisées (dont la coordonnée de profondeur), la transparence, les coordonnées de texture, etc.

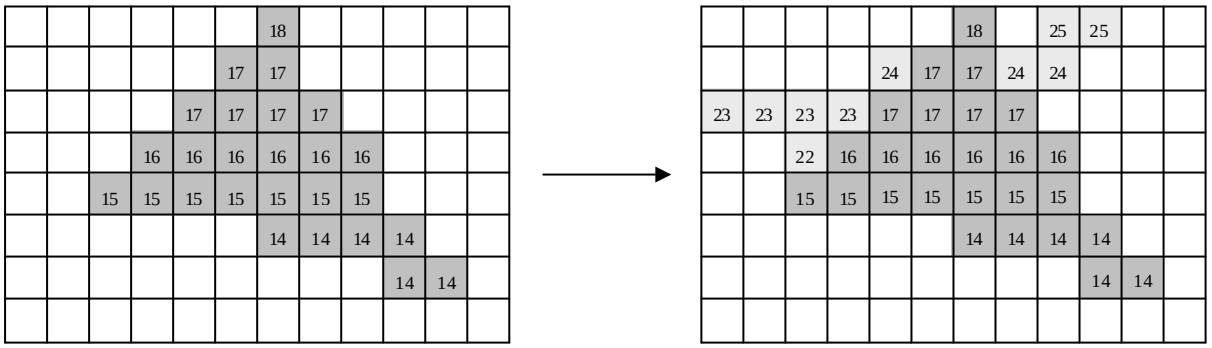
8.3.4. Elimination des surfaces cachées

Lors du calcul de chaque fragment, un calcul est effectué pour déterminer sa contribution à l'image finale. Le test le plus important est celui du z-buffer. Chaque pixel de l'image finale a une valeur de profondeur (z) associée, un entier compris entre 0 et Zmax, Zmax étant la valeur maximale du zbuffer (entier positif). Les valeurs de profondeur sont stockées dans un tableau de mêmes dimensions que l'image finale : le z-buffer, ou tampon de profondeur, noté Z. Initialement, toutes les cases du z-buffer sont, dans notre cas, initialisée à Zmax. Pour chaque fragment, une comparaison entre Z(i, j) et z(i,j) est effectué. Si le test est positif, alors le pixel p(i, j) de l'image finale prend la valeur de couleur de f(i, j) et z(i, j) est affecté à Z(i,j). Sinon, le fragment est rejeté et ne contribue pas à l'image finale. Le test effectué peut être variable suivant l'application, mais le plus souvent c'est un test d'infériorité :

```
Si z(i, j) < Z(i, j)
      Test positif
Sinon
      Test negatif
```

La Figure 36 ci-dessous représente l'état d'une partie d'un tampon de profondeur après le tramage successif de deux triangles. Chaque case représente une case mémoire du tampon, stockant la valeur de profondeur associée à un pixel(i, j). Les cases blanches ont une valeur de profondeur à Zmax (ce qui n'est pas représenté directement sur la figure). Notons que les valeurs de profondeur indiquées sont fictives. Nous observons que le premier triangle occulte en partie le second : il est plus proche du point d'observation, dans l'espace global. L'algorithme du z-buffer permet donc de mettre en œuvre de manière simple l'élimination des surfaces cachées ; il est implanté dans quasiment toutes les cartes graphiques actuelles.

Figure 36 : z-buffer après tramage successif de deux triangles



Z-buffer après tramage du triangle 1

Z-buffer après tramage du triangle 2

Des opérations autres que celles du z-buffer peuvent être appliquées : transparence (par canal alpha), filtrage, etc. Nous ne les détaillons pas ici.

Il devient possible sur les cartes graphiques actuelles de pouvoir spécifier par programmation quels sont les traitements effectués pour chacune des deux grandes étapes (transformations géométriques et tramage). En effet, les cartes actuelles ont des processeurs programmables et permettent à l'utilisateur de définir ses propres traitements, à l'aide d'un langage d'assemblage simple. Cela permet, par exemple d'implanter un modèle d'illumination autre que celui de Phong lors des opérations géométriques (par exemple un modèle pour représenter les surfaces de type métallique) ou bien d'effectuer des traitements algorithmiques supplémentaires lors du tramage (utilisation de textures algorithmiques, etc.). Il reste à voir quel sera l'impact de ces nouvelles caractéristiques sur la visualisation scientifique interactive.

9. Annexe B : Configurations matérielles de rendu parallèle

Nous présentons dans cette partie les caractéristiques principales des configurations matérielles que nous avons décrites en 3.5. Le Tableau 8 ci-dessous donne, pour chaque configuration, les caractéristiques des systèmes de rendu, de composition, d'interconnexion et autres lorsqu'elles sont connues.

Tableau 8 : configurations matérielles de rendu parallèle

NC : Non Connu
NS : Non Significatif

Référence	[KLO02]	[LOM01]	[ORA02]	[HPC01]
Nom de la configuration	DeepView	Sepia2	DVG	Sv6
Type d'architecture	Grappe de PC	Grappe de PC	Grappe de PC reliés en chaîne de composition	Grappe de stations de travail, 1 nœud de contrôle, N nœuds de rendu
Domaine d'application	CAO, visualisation médicale, visualisation moléculaire	Visualisation en dynamique des fluides, imagerie médicale	Simulations civiles et militaires, planification urbaine, architecture, prospection gaz/pétrole	Présentation et prototypage virtuels (aéronautique et automobile), robotique, productique, etc.
Type de données/visualisation	Données surfaciques, données volumiques	Données volumiques	Données surfaciques	Données surfaciques
Parallélisme rendu	Variable	Sort-last	• Subdivision espace-image par sous-pixel • Multiplexage temporel • Répartition suivant espace-écran avec équilibrage de charge • Répartition de la scène avec composition Z (sort-last)	• Équivalent sort-first • Subdivision espace-image par sous-pixel • Mode intermédiaire entre les 2 premiers
Type rendu	Surfacique, volumique	Volumique	Surfacique	Surfacique
Configuration globale	8 nœuds	8 nœuds de rendu, 1 nœud d'affichage	4 ou 8 nœuds de rendu reliées en chaîne (nombre arbitraire possible)	N nœuds de rendu par canal d'affichage, M canaux (3 en pratique)
Configuration nœud de rendu	PC (modèle non connu)	Station de travail Compaq SP 750	Nœud DVG (PC recarossé)	Station de travail HP j6700
processeur	Bi-processeur, à 866 MHz	Intel Pentium 3 Xeon à 800 MHz	NC	Bi-PA-8700 à 750 MHz
mémoire	1 Go RAM	RDRAM (quantité NC)	NC	1 à 16 Go SDRAM 120 MHz
rendu	Accélérateur milieu de gamme (caractéristiques NC)	Accélérateur rendu volumique VolumePro 500	Carte graphique COTS	Carte graphique fx10 Pro
interconnexion	Myrinet et Ethernet Gbit	Carte Sepia 2 (interconnexion ServerNet 2)	• Bus dédié au trafic de pixels (composition • Ethernet (contrôle)	• FastEthernet (contrôle) • Ethernet Gbit (envoi des commandes)
Configuration nœud d'affichage	NS	Idem que nœud de rendu excepté l'accélérateur VolumePro	NS	NS
Configuration rendu	Accélérateur milieu de gamme	Accélérateur de rendu volumique VolumePro 500	ATI Radeon 9700/FireGL-X1	HP fx10 Pro
caractéristiques	NC	• ASIC vg500 (rendu) • 256 Mo SDRAM • rendu lancer de rayon shear-warp • gradient et illumination Phong	• Processeur FGL9700 • 128 ou 256 Mo RAM • stéréo	• 128 Mo RAM • stéréo, lignes anticrénelées • FSAA x4
performances	NC	256^3 voxels à 24,8 à 27,5 im/s pour image de 512^2 pixels	NC	NC
interface	NC	Bus PCI 64 bit à 66 MHz	Bus AGPx8 ou AGP Pro50	Bus PCI-X
Configuration affichage	Mono-écran	Mono-écran	Variable	Variable, jusqu'à canaux (illimité en théorie)
écran	• T221 IBM LCD • diagonale 22,2 pouces • 4 entrées DVI synchrones	NC	Variable	Variable
résolution	3800 x 2400	1024 x 1024	• Standard jusqu'à 1280 x 1024 • Stéréo en 1280 x 1024 à 120 Hz • De 1600 x 1200 (60-85 Hz) à 2048 x 1536 (60 Hz)	• 1600 x 1200 max par canal
Matériel de composition	SGE (Scaleable Graphics Engine)	Carte programmable Sepia 2	Carte DVG programmable	Compositeur DVI propriétaire
Interface	• 16 entrées DVI max • 8 sorties DVI max	• Interface PCI vers données locales (120-150 Mo/s en lecture) • 2 liens ServerNet-2 à 180 Mo/s vers nœuds amont et aval	• Interface hôte PCI • Interface DVI vers les cartes graphiques • Bus pixel dédié vers nœuds amont/aval	• De 4 à 16 entrées DVI
Méthode de composition	NC	• Programmable (circuit PFGA) • Suivant profondeur ou transparence	Suivant valeurs de profondeur	• Juxtaposition • Par calque (interdécalés d'une distance sous-pixel)
Configuration réseau				
Inter-nœud	Myrinet	Fast Ethernet partagé (distribution des informations de point de vue)	NS	Ethernet Gbit
Réseau de composition	Ethernet Gbit	Commutateurs ServerNet 2	Bus dédié de trafic pixels vers nœuds amont et aval	Interface DVI
Environnement logiciel				
OS	Linux Red Hat 7.1	Windows 2000	Linux, Windows	NC
rendu	OpenGL	API Volume Pro		OpenGL (support natif)
Rendu parallèle	Chromium, OpenDX-MPI	NC		Toute application OpenGL
Performances globales	NC		4 et 8 nœuds (avec cartes Radeon 9700)	-
		24 à 28 im/s (1024 x 256 x 256 et 512^3 voxels)	Vitesse de remplissage (texture 32^2) (Mpixels/s) : 4700/4700	Jusqu'à 369M éch/image avec 3 canaux (16 nœuds/canal)

			Traitement géométrique : 4 et 8 nœuds (Mpolygones/s)	249,6/374,4	-

10. Annexe C : applications de visualisation scientifique sur grappe, tableaux de synthèse

Dans cette annexe nous donnons les caractéristiques et performances des applications de visualisation scientifique que nous avons décrites en 5.1 et 5.2.
Pour chacun des deux grands types d'applications (rendu surfacique et volumique), nous donnons les caractéristiques générales des systèmes, matériels et logiciels, sur lesquels elles s'exécutent, ainsi que leurs performances, lorsqu'elles sont connues.

10.1. Rendu surfacique

Tableau 9 : applications de rendu surfacique sur grappe

NC : Non Connu
NS : Non Significatif

Référence	[ZHA01]	[COR02]	[WYL01]
Domaine d'application	Visualisation de données médicales	Visualisation architecturale interactive	Visualisation de simulation numériques
Type de données	Volumiques	Surfaciques	Surfaciques
Taille	De 512 x 512 x 512 voxels à 1600 x 1000 x 5186 voxels	13 Mtriangles	De 7 à 469 Mtriangles
Taille des données extraites	De 6,4 à 487 Mtriangles	NS	NS
Rendu			
Couplage avec calcul	Fort	Fort	-
Parallélisme	Sort-last	Sort-first	Sort-last (SL-sparse)
composition	Matérielle (Metabuffer), Z	NS	Logicielle (« binary-tree/swap »), Z
Type rendu	Surfacique	Surfacique	Surfacique
Calcul	Extraction parallèle d'isosurface « out-of-core »	Détermination en parallèle des surfaces visibles	_
Configuration globale	1 à 32 nœuds d'extraction/rendu, client et serveur sur les mêmes nœuds	1 nœud client, de 1 à 16 nœuds de rendu	1 à 64 nœuds de rendu/composition, 1 nœud d'affichage
Configuration nœud client	Intel Pentium 3 800 MHz, 256 Mo RAM, carte graphique Nvidia GeForce2	Intel Pentium 3 700 MHz	NS
Configuration nœuds de rendu	Idem nœud client	AMD Athlon 900 MHz, 512 Mo RAM, carte graphique nVidia GeForce2, disque IDE	Intel Pentium 3 800 MHz, 512 Mo RAM, GeForce 256
Configuration affichage			
écran	NC	Mosaïque, 1 écran par nœud	Mono-écran
Résolution (par écran/totale)	NC	1024 x 768 / 4096 x 3072	1024 x 768 / NS
Configuration réseau	Fast Ethernet	Ethernet Gbit commuté	ServerNet 2
Environnement logiciel			
Rendu	NC	NC	Libpglc
composition	NC	NS	Libpglc
réseau	NC	MPIPro 1.6.3	MPIPro
Performances agrégées théoriques	NC	NC	352 Mtris/s
Performances agrégées réelles	6 Mtris/s avec 32 nœuds et 487 Mtris	12 Mtris/s	90% avec 32 nœuds et 235 Mtris 85% avec 64 nœuds et 469 Mtris
Vitesse d'affichage max	NC	27 im/s (16 nœuds)	5 im/s (avec 7Mtris)

10.2. Rendu volumique

Nous décrivons ici les grandes caractéristiques des deux applications décrites en 5.2.4. Nous comparons avec une application de rendu volumique sur architecture dédiée ([KNI01], première colonne du tableau qui suit). Notons que dans cette dernière, la composition de l'image i est simultanée avec le rendu de l'image i+1, alors que dans les applications sur grappe ici présentées, le rendu de l'image i+1 ne débute que lorsque la composition de l'image i est finie. En effet les nœuds de rendu effectuent séquentiellement rendu et composition alors que dans le cas de [KNI01], composition et rendu s'effectuent en parallèle (pipeline).

Tableau 10 : applications de rendu volumique sur grappe

NC : Non Connu
NS : Non Significatif

Référence	[KNI01]	[HUM02]	[LUM02]
Domaine d'application	Dynamique des fluides, imagerie médicale	Visualisation de données biologiques	Dynamique des fluides
Données			
Type	Dynamique	Statique	Dynamique
Dimensions spatiales (nbre de voxels)	1024^3	1024 x 256 x 256	512^3
Taille temporelle (nombre de pas de temps)	NC	NS	350 et 200
Rendu	« Out-of-core »	« In-core »	« In-core » et « out-of-core »
Couplage avec calcul	Non	Non	Non
Parallélisme	Sort-last	Sort-last	Sort-last
composition	« Back-to-front », simultanée avec rendu de l'image suivante	Binary-swap	Binary-swap
Type du rendu	Textures 3D alignées sur la direction de vue	Textures 3D alignées sur la direction de vue	Textures 2D alignées sur repère global (3 jeux de textures)
Point de vue mobile	Oui	Non	Oui
Fonction de transfert	Codage par table de texture (LUT)	Programmable par utilisateur en matériel	Codage dans table de texture
Configuration globale	Origin 2000 SGI	Grappe : 16 nœuds de rendu/composition	Grappe : 8 nœuds de rendu/composition, 1 nœud d'affichage
Configuration parallèle	128 processeurs, 16 accélérateurs graphiques IR-2	Intel Pentium 3 Xeon 800 MHz, 256 Mo RAM, carte graphique GeForce 3	AMD Athlon 1,3 GHz, 1 Go RAM, carte graphique GeForce 3 64 Mo, 2 disques IDE avec RAID 0
Configuration affichage	Mono-écran	NC	Mono-écran
Configuration réseau	Réseau d'interconnexion spécialisé	Myrinet	Commutateur Ethernet Gbit
Environnement logiciel			
rendu	NC	Chromium (OpenGL parallèle), rendu par textures 2D	NC (par textures 2D), compression DCT et textures indicées (8 bit)
composition	Composition « back to front »	Méthode « binary-swap »	Méthode « binary-swap » avec compression RLE
réseau	NC	Chromium (couche d'abstraction réseau)	MPICH
Performances			
Nombre de cartes graphiques	16 pipes	16	8

Performances agrégées (Gvoxels/s)	4	0,64 à 1,59	0,65 (in-core) 0,39 (out-of-core)
Performances par accélérateur graphique (Gvoxels/s)	0,25	0,04 à 0,1	0,08 (in-core) 0,04 (out-of-core)
Vitesse de rendu (im/s)	4	5 à 13	4,9 (in core) 2,9 (out-of-core)

11. Bibliographie

[AHR00] J. Ahrens, C. Law, M. Papka, A Parallel Approach for Efficiently Visualizing Extremely Large Time-Varying Datasets, Los Alamos National Laboratory – Technical Report #LAUR-00-1620, 2000

[AHR01] J. Ahrens, K. Brislaw, K. Martin, B. Geveci, C.C. Law, M. Papka, Large-Scale Data Visualization Using Parallel Data Streaming, IEEE Computer Graphics and Applications, vol. 21, n° 4, pp. 34-41, Juillet/Août 2001

[ALA03] Alacritech, Alacritech 1000x1 Copper Gigabit Adapter, http://www.alacritech.com/html/1000x1_copper_adapter.html, 2003

[ALL02] J. Allard, V. Gouranton, L. Lecointre, E. Melin, B. Raffin , Net Juggler and SoftGenLock : Running VR Juggler with Active Stereo and Multiple Displays on a Commodity Component Cluster, Proceedings of the IEEE Virtual Reality 2002, pp. 273-274, 2002

[BLA00] W. Blanke, C. Bajaj, X. Zhang, D. Fussell, A Cluster Based Emulator for Multidisplay, Multiresolution Parallel Image Compositing, TICAM Technical Report, University of Texas, Austin, Février 2000

[BOD95] N.J. Boden, D. Cohen, R.E. Felderman, A.E. Kulawik, C.L. Seitz, J.N. Seizovic, W.K. Su, Myrinet : A Gigabit-per-Second Local-Area Network, IEEE Micro, vol. 15, n° 1, pp. 29-36, Février 1995

[CHE98] M. Chen, G. Stoll, H. Igehy, K. Proudfoot, P. Hanrahan , Simple Models of the Impact of Overlap in Bucket Rendering, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, pp. 105-112, 1998

[CHR03] The Chromium Project, http://sourceforge.net/projects/chromium/, 2003

[COR02] W.T. Corrêa, J.T. Klosowski, C.T. Silva, Out-of-Core Sort-First Parallel Rendering for Cluster-Based Tiled Displays, Parallel Computing, vol. 29, pp. 325-338, 2003

[COS02] J-E. Coste, P. Guerville, J-P. Nominé, Visualisation Haute Performance et Haute Résolution à base de Clusters de PC sous Linux, CEA/DAM Ile-de-France, Rapport CEA -R-6001, Mai 2002

[DDW99] Digital Display Working Group, Digital Visual Interface, Revision 1.0, http://www.ddwg.org/downloads.html, Avril 1999

[DEF98] M.F. Deering, The Limits of Human Vision, Sun Microsystems, 2nd International Immersive Projection Technology Workshop, Ames, Iowa, USA, 1998

[DIS03] Dolphin Interconnect Solutions Inc., PCI-64/66 - PCI-SCI Adapter Card for System Area Networks, http://www.dolphinics.com/products/hardware/pci64.html, 2003

[ELV92] T. Todd Elvins, A Survey of Algorithms for Volume Visualization, Computer Graphics, vol. 26, n° 3, pp. 194-201, Août 1992

[EXT02] Extremetech, ATI's Radeon 9700 Takes Performance Lead, http://www.extremetech.com/article2/0,3973,388843,00.asp, 2002

[FOL90] J.D. Foley, A.V. Dam. S.K. Feiner, J.F. Hughes, Computer Graphics : Principles and Practice, Addison-Wes1ey, 1990

[HEI02] A. Heirich, M. Shand, E. Oertli, G. Lupton, P. Ezolt, Alpha/Depth Acquisition through DVI, Vis2002, Workshop on Commodity-Based Visualization Clusters, 2002

[HER00a] M. Hereld, I.R. Judson, R.L. Stevens, Introduction to Building Projection-Based Tiled Display Systems, IEEE Computer Graphics & Applications, vol. 20, n° 4, pp. 22-28, Juillet/Août 2000

[HER00b] M. Hereld, I.R. Judson, R.L. Stevens, Developing Tiled Projection Display Systems, Proceedings of Fourth Immersive Projection Technology Workshop, 2000

[HOC] P. Hochschild, R. Swetz, Scaleable Graphics Engine, High-Performance SP Graphics, IBM T.J. Watson Research Center http://www.emsl/pnl.gov/capabs/mscf/visualization/sge/sge.pdf

[HOU02] M. Houston, G. Humphreys, R. Frank, P. Hanrahan, Learning From the Stanford/DOE Visualization Cluster, VIS2002, 2002

[HPC01] Hewlett-Packard Company, Introduction to the hp visualization center sv6, 2001

[HUG03] R. Hughes-Jones, S. Dallison, G. Fairey, Performance Measurements on Gigabit Ethernet NICs and Server Quality Motherboards

[HUM00] G. Humphreys, I. Buck, M. Eldridge, P. Hanrahan, Distributed Rendering for Scalable Displays, Proceedings of IEEE Supercomputing 2000, Octobre 2000.

[HUM01] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett et P. Hanrahan, WireGL : A Scalable

Graphics System for Clusters, Proceedings of SIGGRAPH 2001, pp. 129-140, Août 2001

[HUM02] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J.T. Klosowski, Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002), vol. 21, n° 3, pp. 693-702, Juillet 2002

[INT02] Intel, Intel 845GE/845PE Chipset Datasheet, http://www.intel.com/design/chipsets/datashts/251924.htm?iid=ipp_845pechpst+info_ds&, Octobre 2002

[KLO02] J.T. Klosowski, P.D. Kirchner. J. Valuyeva, G. Abram, C.J. Morris, R.H. Wolfe, T. JackMann, Deep View: High-Resolution Reality, IEEE Computer Graphics and Applications, vol. 22, n° 3, pp. 12-15, Mai/Juin 2002

[KNI01] J.Kniss, P. McCormick, A. McPherson, J. Ahrens, J. Painter, A. Keahey, C. Hansen , Interactive Texture-Based Volume Rendering for Large Data Sets, , IEEE Computer Graphics and Applications, vol. 21, n° 4, pp. 52-61, Juillet/Août 2001

[LOM01] S. Lombeyda, L. Moll, M. Shand, D. Breen, A. Heirich, Scalable Interactive Volume Rendering Using Off-the-Shelf Components, IEEE Parallel and Large-Data Visualization and Graphics Symposium, pp. 115-121, 2001

[LUM02] E.B. Lum, K.L. Ma, J. Clyne, A Hardware-Assisted Scalable Solution for Interactive Volume Rendering of Time-Varying Data, IEEE Transactions on Visualisation and Computer Graphics, vol. 8, n° 3, pp. 286-301, Juillet/Septembre 2002

[MCP01] A. McPherson, Los Alamos Cluster Visualization, 13 Août, SIGGRAPH 2001

[MCC99] P.S. McCormick, R.D. Ryne, Visualizing High-Resolution Accelerator Physics, IEEE Computer Graphics and Applications, vol. 19, n° 5, pp. 11-13, Septembre/Octobre 1999

[MOLL99] L. Moll, A. Heirich, M. Shand, Sepia : Scalable 3D Compositing Using PCI Pamette, Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 146-157, 1999

[MOLN94] S. Molnar, M. Cox, D. Ellswoth, H. Fuchs, A Sorting Classification of Parallel Rendering, IEEE Computer Graphics and Applications, vol. 14, n° 4, pp. 23-32, Juillet 1994

[MON97] J.S. Montrym, D.R. Baum, D.L. Dignam, C.J. Migdal, Infinite Reality : A Real-Time Graphics System, Proceedings of SIGGRAPH 1997, pp. 293-302, 1997

[MYR03] Myricom, M3F-PCI64B & M3F-PCI64C, Universal, 64/32-bit, 66/33MHz, Myrinet-2000-Fiber/PCI interfaces, http://www.myrinet.com/myrinet/PCI64/m3f-pci64b.html, 2003

[NOM01] J-P. Nominé, Visualisation de données scientifiques, une introduction, http://dept-info.labri.u-bordeaux.fr/Enseignement/IMM/Imm/CoursVisualisationDonnees/tsld001.htm, Novembre 2000

[NVI02] Nvidia Corporation, Technical Brief, AGP 8X, Evolving the Graphics Interface, http://www.nvidia.com/docs/lo/2188/SUPP/AGP8X_92502v1.pdf, Novembre 2002

[ORA02] ORAD Hi-Tec Systems Ltd, DVG – the first COTS-based 3D scalable engine, http://www.orad.co.il/pdf2/DVG_0006.PDF, Octobre 2002

[PAR03] ParaView, Parallel Visualization Application, http://www.paraview.org/HTML/Index.html, 2003

[PER01] K.A. Perrine, D.R. Jones, Parallel Graphics and Interactivity with the Scaleable Graphics Engine, IEEE Proceedings of Supercomputing 2001, Novembre 2001

[QUA03] Quadrics, Quadrics Products, QsNet, www.quadrics.com, 2003

[RAF03] B. Raffin, Grappe de PC : de l'installation à la réalité virtuelle, séminaire CEA, Janvier 2003

[SAM98] R. Samanta, T. Funkhouser, Dynamic Algorithms for Sorting Primitives Among Screen-Space Tiles in a Parallel Rendering System, Technical Report, Department of Computer Science, Princeton University, 1998

[SAM99] R. Samanta, J. Zheng, T. Funkhouser, K. Li ; J.P. Singh, Load Balancing for Multi-Projector Rendering Systems, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, pp. 107-116, Août 1999

[SCHA03] B. Schaeffer, C. Goudeseune, Syzygy : Native PC Cluster VR, Proceedings of the IEEE Virtual Reality 2003, pp. 15-22, 2003

[SCHR98] W. Schroeder, K. Martin, B. Lorensen, The Visualization Toolkit, An Object-Oriented Approach to 3D Graphics, Prentice Hall PTR, 1998

[SHA02] A. Sharma, X. Liu, P. Miller, A. Nakano, R.K. Kalia, P. Vashishta, W. Zhao, T.J. Campbell, A. Haas, Immersive and Interactive Exploration of Billion-Atom Systems, Proceedings of IEEE VR 2002, p. 217-223

[STO01] G. Stoll, M. Eldridge, D. Patterson, A. Webb, S. Berman, R. Levy, C. Caywood, M. Taveira, S. Hunt,

P. Hanrahan, Lightning-2 : A High-Performance Display Subsystem for PC Clusters, Proceedings of SIGGRAPH 2001, pp. 141-148, 2001

[TOM02] S. Tomov, R. Bennett, M. McGuigan, A. Peskin, G. Smith, J Spiletic, Developing Interactive Parallel Visualization For Commodity-Based Clusters, Vis2002, Workshop on Commodity-Based Visualization Clusters, 2002

[UND00] I. Underwood, A review of microdisplay technologies, Digest of SID@EID, 2000

[WYL01] B. Wylie, C. Pavlakos, C. Lewis, K. Moreland, Scalable Rendering on PC Clusters, IEEE Computer Graphics and Applications, vol. 21, n° 4, pp. 62-70, Juillet/Août 2001

[YAN02] J. Yang, J. Shi, Z. Jin, H. Zhang, Design and Implementation of A Large-scale Hybrid Distributed Graphics System, Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization, 2002

[YOD97] L. Yoder, The Digital Display Technology of the Future, INFOCOMM ’97, 5-7 juin 1997 www.dlp.com/dlp_technology/images/dynamic/white_papers/144_yoder.pdf

[ZHA01] X. Zhang, C. Bajaj, W. Blanke, Scalable Isosurface Visualization of Massive Datasets on COTS Clusters, Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics, pp. 51-58, Octobre 2001