Visualisation interactive de simulations à grand nombre d'atomes en physique des matériaux

Simon Latapie

ÉCOLE CENTRALE DES ARTS ET MANUFACTURES « ÉCOLE CENTRALE PARIS »

THÈSE

présentée par

Simon Latapie

pour l'obtention du

GRADE DE DOCTEUR

Spécialité : Ingénierie Scientifique et Visualisation

Laboratoire d'accueil : MAS

SUJET : Visualisation interactive de simulations à grand nombre d'atomes en Physique des Matériaux

soutenue le : 7 avril 2009

devant un jury composé de :

Florian De Vuyst	**Président du Jury**
Sabine Coquillart	**Rapporteur**
Frédéric Mérienne	**Rapporteur**
Patrick Callet	**Directeur de thèse**
Jean-Philippe Nominé	**Examinateur**
Bruno Raffin	**Examinateur**

2009ECAP0015

[9 mai 2009 at 3:15]

[9 mai 2009 at 3:15]

Dédiée à Tiphaine, qui a réussi à me supporter pendant toute ma thèse, et qui, bien plus étonnant, a décidé de me supporter pendant encore un petit moment.

RÉSUMÉ

Afin de répondre au besoin croissant de puissance de calcul qui intéresse les applications scientifiques, des architectures de calculateurs de plus en plus parallèles se répandent, de type cluster, MPP, ou autre, avec de plus en plus d'unités de calcul. Les codes de calcul, en s'adaptant à ce parallélisme, produisent de fait des résultats de plus en plus volumineux. Dans le domaine plus spécifique de la dynamique moléculaire appliquée à la physique des matériaux, le nombre de particules d'un système étudié est aujourd'hui de l'ordre de quelques millions à quelques centaines de millions, ce qui pose d'importants problèmes d'exploitation des données produites. Les solutions de visualisation utilisées auparavant ne sont plus capables de traiter une telle quantité d'information de façon interactive.

Le but de cette thèse est de concevoir et d'implémenter une solution de manipulation de données composées d'un très grand nombre de particules, formant un ensemble "dense" (par exemple, une simulation en dynamique moléculaire de quelques dizaines de millions d'atomes). Cette solution devra être adaptative pour pouvoir fonctionner non seulement sur un poste de bureau de type "desktop", mais aussi dans des environnements collaboratifs et immersifs simples, par exemple de type mur d'image, avec éventuellement des éléments de réalité virtuelle (écrans stéréo, périphériques 3D). De plus, elle devra augmenter l'interactivité, c'est-à-dire rendre le système plus apte à réagir avec l'utilisateur, afin d'effectuer des phases d'exploration des données plus efficaces.

Nous proposons une solution répondant à ces besoins de visualisation particulaire dense, en apportant des améliorations :

- à l'interaction, par un système permettant une exploration simple et efficace d'une simulation scientifique. A cette fin, nous avons conçu, développé, et testé le "FlowMenu3D", une extension tridimensionnelle du "FlowMenu", un menu octogonal utilisé en environnement 2D.
- à l'interactivité, par une architecture optimisée pour le rendu de systèmes particulaires denses. Cette dernière, intégrée au framework de visualisation VTK, est conçue sur le principe d'un rendu parallèle hybride sort-first/sort-last, dans lequel le système sort-last Ice-T est couplé à des extensions de partitionnement spatial des données, d'occlusion statistique par une méthode de Monte-Carlo, mais aussi de programmation GPU à base de shaders pour améliorer la performance et la qualité du rendu des sphères représentant les données.

ABSTRACT

The need for more computing power for scientific applications leads supercomputer manufacturers to rely on more and more parallel architectures, such as cluster or MPP systems, increasing the number of processing units. The correlative adjustment of computing software - simulation codes - to massive parallelism leads to larger and larger datasets. More specifically, in molecular dynamics simulations applied to material physics, particle sets are typically composed of millions to hundreds of millions of particles, which raises important post-processing issues. Most current visualization solutions do not scale up to interactively handle such an amount of information.

The goal of this thesis is to design and implement a post-processing solution which is able to handle "dense" sets with a very large number of particles (for example, a molecular dynamics simulation result with several millions of atoms). This solution will have to be adaptive enough to run either on a desktop environment, or on a simple collaborative and immersive configuration, such as a tiled display, possibly with some virtual reality devices (such as stereo displays and 3D interaction peripherals). Moreover, this solution will have to maximize the interactivity, i.e. to increase the reactivity of the system against the user commands, to improve data exploration stages.

We propose a solution optimized for dense particle systems visualization, improving both interaction and interactivity :

- regarding interaction, a new simple and efficient way of exploring a scientific simulation is proposed. We developed and tested "FlowMenu3D", a 3D extension of the "FlowMenu" concept, an octagonal menu previously designed for 2D environments.
- regarding interactivity, we propose an optimized architecture for dense particle systems rendering. The system relies on VTK visualization framework and a parallel hydrid sort-first/sort-last renderer. The existing Ice-T component is augmented with spatial data partitioning as well as statistical (Monte Carlo) occlusion mechanisms. GPU shaders eventually improve the performance and quality of rendering, using spheres as basic primitives.

Je sers la Science
et c'est ma joie.

Léonard est un génie, Bob de Groot et Turk,
Dargaud, 1977

REMERCIEMENTS

Je tiens à sincèrement remercier, dans un ordre purement aléatoire :

Jean Philippe Nominé, pour son soutien presque quotidien pendant plus de trois ans, ses conseils, ses coups de pied au derrière, sa base d'articles inépuisable, et son goût prononcé pour la correction des fautes d'orthographe,

Patrick Callet, pour son soutien, sa patience, et ses conseils éclairés sur le rendu de couleur,

Patrick Allal, pour son accueil au bâtiment L1 et sa disponibilité,

Christian Saguez, pour son accueil au laboratoire MAS,

Cédric Enaux, Gilles Kluth, Gaël Poette, et Manuel Juliachs, pour leur soutien réellement quotidien (du moins les jours ouvrés), et pour nos discussions si passionnantes et enflammées sur n'importe quel sujet un temps soit peu polémique,

Thierry Carrard, pour ses conseils plus qu'avisés sur les arcanes du rendu parallèle,

Laurent Soulard, pour ses conseils en matière de Physique des Matériaux,

Erwine, qui fut la seule à ne jamais me poser de questions sur ma thèse pendant cette dernière,

Tiphaine, sans qui cette aventure n'aurait peut-être pas commencé, et sûrement pas fini.

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION

Cette étude trouve son origine dans un besoin simple et concret d'un laboratoire en Physique des Matériaux du CEA/DIF : post-traiter les données générées par de nouveaux algorithmes de simulations de matériaux en Dynamique Moléculaire.

En changeant de matériel de calcul scientifique, le nombre de particules par simulation a tellement augmenté que les outils de visualisation qu'ils utilisaient auparavant n'étaient plus suffisants. Or cette augmentation est un facteur clé dans les recherches qui sont effectuées en Physique des Matériaux : l'une des problématiques est justement de modéliser assez de particules, donc un milieu discret, pour se raccorder aux résultats macroscopiques, donc en milieu continu, que peuvent fournir d'autres échelles de modélisation de la Physique.

Les phénomènes que les chercheurs espèrent reproduire étant macroscopiques, donc globaux, il leur fallait donc une solution de visualisation permettant des phases "d'exploration", constituées d'abord de vues d'ensemble des jeux de données, qui amènent alors à la sélection de zones d'intérêt qui seront étudiées plus en détail.

Le laboratoire voulait non seulement pouvoir visualiser les cas de petite et moyenne taille sur leur poste de post-traitement, mais aussi pouvoir étudier de plus grands jeux de données sur un mur d'image, en utilisant au maximum la surface d'affichage pour l'exploration (donc en limitant au maximum l'apparition des outils de contrôle du système).

Lors de l'étude préliminaire qui nous a permis de définir les besoins précis de ce domaine en matière de visualisation scientifique, nous nous sommes aperçus qu'il n'existait pas, à proprement parler, de solution adaptée à cette exploration de systèmes particulaires. Il manquait en particulier :

EN INTERACTION : un système simple et assez efficace pour permettre une exploration aisée d'un jeu de données, destiné à des chercheurs non experts en informatique, dans un environnement matériel, considéré comme immersif, composé d'une grande surface d'affichage, mais avec peu d'outils de capture et de saisie.

EN INTERACTIVITÉ : il n'existait a priori aucune solution basée sur un matériel générique, qui soit réellement optimisé pour la visualisation de systèmes particulaires denses.

Nous avons donc proposé au laboratoire d'étudier ces deux problématiques.

En interaction, nous nous sommes concentrés sur la conception d'un système de contrôle d'application original, capable d'être utilisé dans un environnement immersif, mais sans être dépendant d'un matériel très spécifique à la réalité virtuelle.

En interactivité, nous avons développé une architecture spécifique aux besoins de visualisation de systèmes particulaires denses, en utilisant des configurations informatiques de type poste de bureau avancé (génération de carte graphique récente), ou cluster (grappe) d'ordinateurs de rendu graphique.

La première partie de ce mémoire présente rapidement les domaines concernés par l'étude, ainsi que les concepts indispensables à la bonne compréhension du document, et un tour d'horizon des différentes solutions logicielles déjà existantes.

La seconde partie décrit l'étude effectuée sur l'interaction : les choix qui ont amené à étudier le système *FlowMenu*, et les efforts qui ont été fournis pour adapter ce dernier à un environnement immersif.

La dernière partie propose une architecture avec un mode de parallèlisme hybride dit *sort-first/sort-last*, ainsi que des optimisations pour le rendu de sphères colorées, qui offrent une solution adaptée à la visualisation de systèmes particulaires denses.

Première partie

VISUALISATION SCIENTIFIQUE : GÉNÉRALITÉS, APPLICATION À LA PHYSIQUE DES MATÉRIAUX

Cette partie présente rapidement les différents domaines abordés par cette étude. Elle tente de fournir assez d'informations pour fixer précisément le contexte et justifier les choix qui ont amené à travailler sur les sujets qui seront abordés plus en détails dans la seconde et la troisième partie.

Sont présentés, successivement :

- une courte introduction à la visualisation scientifique,
- ses applications possibles à la physique des matériaux
- le concept d'interaction, et en particulier de contrôle d'application,
- le concept d'interactivité, et les systèmes et solutions qui lui sont liés.

1

LA VISUALISATION SCIENTIFIQUE : QU'EST-CE QUE C'EST ?

Ce chapitre rappelle brièvement les concepts fondamentaux liés à la visualisation scientifique, et présente le contexte de cette étude.

1.1 UNE DÉFINITION

La visualisation scientifique consiste à mettre en images des résultats de calcul ou de mesure afin d'aider à la compréhension de phénomènes physiques et naturels.

Cette technique permet de mettre à profit les capacités de synthèse et d'interprétation de la perception visuelle humaine dans la compréhension des données : elle donne du sens aux nombres.

1.2 VISUALISER : QUOI, ET QUAND ?

La visualisation scientifique est souvent utilisée à des fins de communication pour un domaine donné. On produit de "jolies images", afin de promouvoir des résultats de simulation, ou expliquer des phénomènes à des néophytes.

Pourtant, la visualisation scientifique est utile à bien d'autres moments de la simulation :

AVANT : La visualisation est utile à la préparation de calculs : elle aide à la conception des maillages, à l'évaluation de leur qualité, à la vérification des conditions initiales. Elle intervient aussi dans des phases éventuelles plus "amont" de modélisation géométrique et CAO qui peuvent précéder un maillage en vue de calcul numérique.

PENDANT : Elle permet d'effectuer rapidement des vérifications sur l'évolution globale de la simulation. Elle peut même, dans le cas de simulations où le temps de calcul est faible ou peut être réduit, être couplée au système de calcul : on a alors un système de pilotage visuel interactif, approche dite de *steering*[Esno5] en anglais. Une forme plus faible de couplage peut consister en un simple suivi visuel (*monitoring*), sans rétro-action de l'utilisateur sur le calcul et ses paramètres.

APRÈS : Elle permet un dépouillement efficace des résultats de la simulation, en filtrant les données et en les représentant graphiquement.

1.3 LE PROCESSUS DE VISUALISATION

Le processus de visualisation scientifique a été décomposé par Pavlakos[PH04] de la façon suivante (cf. figure 1) :

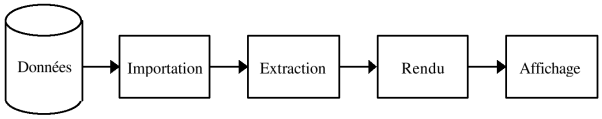


FIG. 1: processus de visualisation

IMPORTATION : les données scientifiques que l'on souhaite visualiser sont extraites statiquement (données "mortes" stockées pendant une simulation numérique, par exemple), ou interactivement (extraction des informations en cours de simulation).

EXTRACTION : il s'agit là d'extraction de parties, de sous-ensembles, de caractéristiques ou de régions d'intérêt. Ces extraits des données sont plus ou moins transformés et préparés sous forme de primitives géométriques. Suivant les différents mode de visualisation (surfacique, volumique, atomistique, etc.) et les différents types de données (scalaires, vecteurs 2D, 3D etc), les primitives peuvent être de nature diverse (points, lignes, triangles, tétraèdres, etc). Pour la visualisation atomistique, il peut s'agir directement de sphères ou de triangles composant des sphères.

RENDU : l'ensemble de la scène, composée des primitives géométriques, est alors projeté dans l'espace d'affichage. Lorsqu'il s'agit d'un écran (comme c'est souvent le cas), on appelle ce processus la "rasterisation" (cf. figure 2 : l'image est représentée comme un plan subdivisé en cellules régulières, appelées pixels, et une fois la scène projetée sur ce plan, seules sont conservées les valeurs des couleurs des pixels.

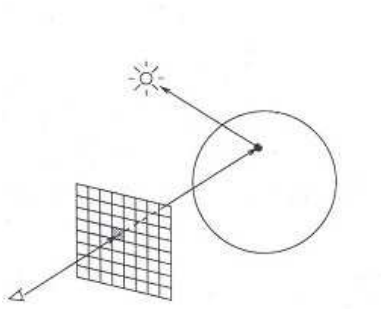


FIG. 2: Rendu de scène : de la géométrie aux pixels

AFFICHAGE : une fois les données filtrées par les différents processus précédents, elle sont envoyées au matériel d'affichage (un écran de taille variable, ou plusieurs écrans via plusieurs projecteurs, par exemple).

1.4 LES DIFFÉRENTS TYPES DE DONNÉES

On classe généralement les données selon :
- leur nature, ou dimension physique (scalaire, vecteur, tenseur, par exemple)
- la dimension de leur support (1D, 2D, 3D, ou plus)

Ces deux caractéristiques déterminent pour l'essentiel les types de représentations qui peuvent être utilisés pour représenter efficacement les données. Par exemple, des données scalaires ne seront pas affichées de la même manière suivant que le support est 1D ou 3D.

1.4.1 *Notion de maillage*

Si les données représentent l'ensemble des valeurs d'un milieu continu, il devient difficile de les calculer par l'intermédiaire d'un ordinateur.

Pour cela, il faut "discrétiser" le domaine en un ensemble de sous-éléments, dans lesquels sont effectués les calculs. Cette décomposition est appelée un maillage.

Il existe de nombreux types de maillages : pour les domaines 2D et 3D, on utilise en général :

UN MAILLAGE STRUCTURÉ, qu'il soit uniforme, régulier (cartésien par exemple), ou irrégulier (curvilinéaire par exemple),

UN MAILLAGE NON STRUCTURÉ, où les sommets et les connectivités ne suivent aucune règle a priori.

UN MAILLAGE SPÉCIFIQUE, qu'il soit adaptatif (comme l'AMR (-*Adaptive Mesh Refinement*- qui recouvre une méthode générale de raffinement dynamique par blocs ou cellules dans les zones de forte variation du phénomène calculé), multiblocs, qu'il mélange les dimensions, non conforme, etc.

Le choix d'un maillage dépend principalement du domaine physique de la simulation, et des méthodes numériques utilisées pour calculer les solutions. Cela fixe en général la nature des données, et leur niveau de structure. Ce dernier influe sur la complexité de stockage.

Par exemple, un maillage non structuré, où il faut donc stocker l'ensemble des sommets et des connectivités, consommera beaucoup plus de mémoire qu'un maillage uniforme, où la structure est implicite (seul le pas d'espace doit être stocké). La figure 3 présente rapidement les différents types de maillage. De plus, les algorithmes de visualisation seront plus complexes sur des maillages non structurés.

La variation temporelle peut ajouter une source de complexité. Souvent, les maillages les plus complexes sont aussi ceux qui varient le plus au cours du temps (comme l'AMR).

Les maillages sont des artefacts de calcul dont on s'abstrait parfois pour visualiser globalement les variables sur les domaines d'intérêt. Ils sont visualisés en tant que tel, pour la mise au point du calcul, son initialisation, ou simplement utile à la compréhension du phénomène physique ou numérique.

Les maillages sont difficiles à visualiser : la forme des mailles est révélée par les arêtes et les faces, ce qui conduit vite à des densités d'information inextricables en 2D ou difficiles à percevoir en profondeur en 3D. Une visualisation efficace de maillages suppose souvent des extractions spécifiques (voisinages, peaux, coupes), de la transparence, voire de l'affichage stéréoscopique.

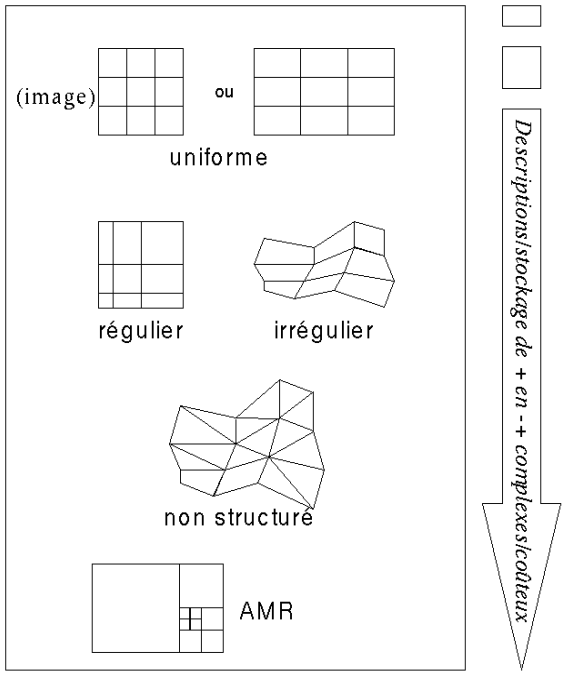


FIG. 3: Exemples de maillages triés par complexité de stockage

Nous allons maintenant rapidement décrire les techniques les plus courantes pour la représentation de données en visualisation scientifique.

1.4.2 *La représentation des scalaires*

Pour un support 1D

Pour de telles données, on utilise des représentations par histogrammes, courbes interpolées ou approximées, camemberts, entre autres. C'est ce qu'un bon "grapheur" est en général capable de représenter.

Pour un support 2D

Pour ce type de support, on retrouve généralement deux grands types de représentations (cf. figure 4) :

LA COLORATION (*colormapping* en anglais). Pour chaque valeur, on associe une certaine couleur, qui sera reportée sur la représentation du domaine. Pour plus de détails sur l'utilisation de la couleur dans la visualisation scientifique, voir la section 10.6 de ce document.

LES ISOLIGNES, appelées lignes de niveau dans certains domaines. Le principe est de représenter uniquement les points du domaine

où la variable étudiée porte une certaine valeur. Pour des phénomènes ayant une continuité spatiale, l'ensemble de ces points forment des lignes.

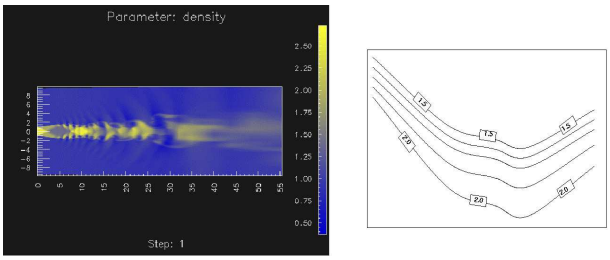


FIG. 4: *Colormapping* et isolignes

Pour un support 3D

Il existe deux grandes philosophies pour la représentation de scalaires en 3D :

LA VISUALISATION SURFACIQUE : elle consiste à transférer les techniques 2D dans un environnement 3D. Le *colormapping* est utilisé sur des plans de coupe, alors que les isolignes deviennent des isosurfaces (cf. figure 5). Il est également possible de coupler les deux techniques, pour visualiser simultanément plusieurs variables.

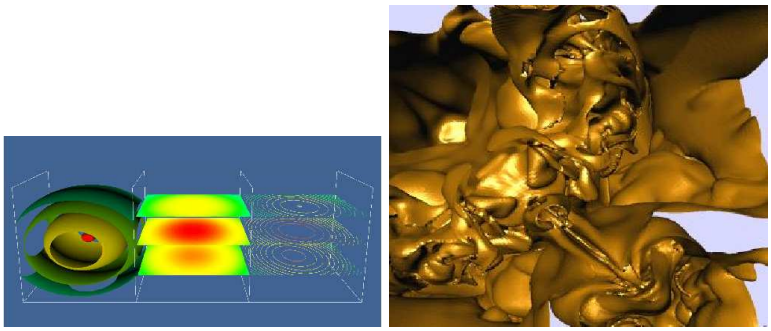


FIG. 5: Techniques de visualisation surfacique : plans de coupes, isosurfaces

LA VISUALISATION VOLUMIQUE : le principe d'une telle technique est de représenter les variables par une propriété optique (couleur ou opacité), et d'effectuer un rendu le plus réaliste possible de la propagation de la lumière dans le domaine (cf. figure 6).

1.4.3 *La représentation des vecteurs*

La représentation "naïve" des vecteurs consiste à tracer des "glyphes" à l'intérieur du domaine que l'on souhaite visualiser, glyphes qui font alors apparaître les informations vectorielles, à savoir les directions

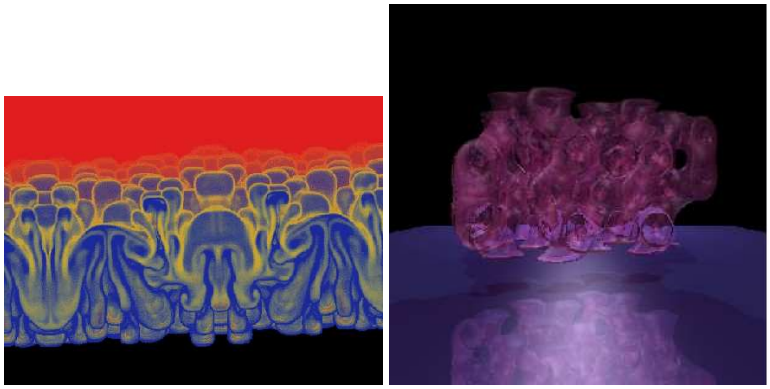


FIG. 6: Techniques de visualisation volumiques, tirées du projet TRex : http ://www.ccs.lanl.gov/ccs1/projects/Viz/pr-trex.html

et intensités. On utilise en général des flèches, cônes, ou tubes pour visualiser des vecteurs.

Dans un rendu 3D à base de facettes, une flèche se représente à l'aide d'un cylindre et d'un cône.

Or cette représentation souffre de deux problèmes principaux : en plus de surcharger rapidement la scène jusqu'à la rendre souvent illisible, elle peut être très coûteuse en temps lors du rendu (à base de facettes), puisqu'au final la scène sera composée d'un nombre de facettes égal au nombre de glyphes multiplié par le nombre de facettes que compose un glyphe.

On cherche donc souvent à ne plus représenter les vecteurs directement, mais seulement des informations indirectes qui donnent des indications sur le champ de vecteurs. Comme techniques usuelles, on peut citer :

LES LIGNES DE COURANT : on représente certaines tangentes aux champs de vecteurs (cf. figure 7).

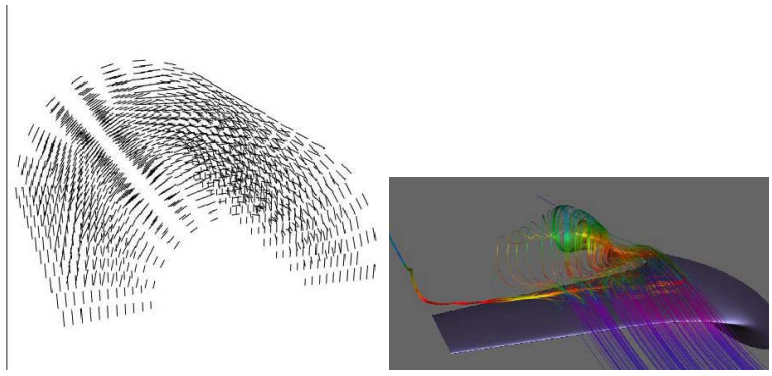


FIG. 7: Techniques de visualisation de vecteurs : glyphs, lignes de courant (ISL : *illuminated stream lines*[Avi])

LE LÂCHER DE PARTICULES : l'animation met en évidence les trajectoires des particules suivant le champ.

L'ADVECTION DE TEXTURES : aussi appelée LIC (Line Integral Convolution). Cette technique consiste à "transporter" une texture composée de bruit blanc suivant le champ vectoriel. On parle de "techniques de représentation denses" car elles remplissent une trame d'image malgré la structure discrétisée du champ de vecteurs.

L'EXTRACTION DE CARACTÉRISTIQUES : on tente de faire apparaître des informations très précises, comme les points critiques, les lignes de séparation, les vortices (cf. figure 8). Cette technique est très dépendante du domaine d'application, et demande souvent des pré-calculs lourds.

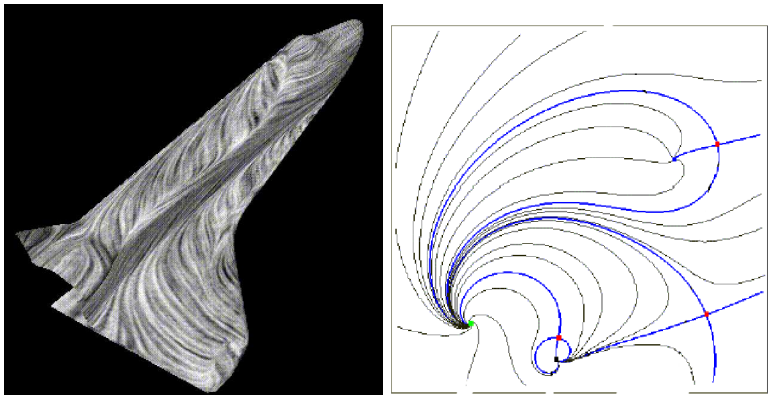


FIG. 8: Techniques de visualisation de vecteurs : advection de textures, et extraction de topologie

1.4.4 *La représentation des tenseurs*

Les techniques de représentations graphiques de tenseurs se limitent en général à ceux d'ordre deux et symétriques. Leur caractère diagonalisable permet de se ramener à trois champs de vecteurs. On peut alors utiliser :

DES ELLIPSOÏDES, les trois vecteurs donnant les directions et caractérisques (cf. figure 9).

DES "HYPER-LIGNES" DE COURANT, où l'on trace les lignes suivant les vecteurs de valeurs propres principales, de section elliptique axés sur les deux autres valeurs propres.

Ces dernières années, motivées par les applications médicales, beaucoup de travaux ont approfondi la visualisation de "tenseurs de diffusion" sur des données issues d'IRM, utile à l'étude de structures dans le cerveau notamment. Des mélanges de techniques précédemment citées pour les vecteurs et les tenseurs sont souvent utilisées pour révéler des structures fibreuses, des anisotropies, le tout dans des données mesurées bruitées, ce qui pose en outre des problèmes de robustesse des méthodes[ZB04][ZLK04].

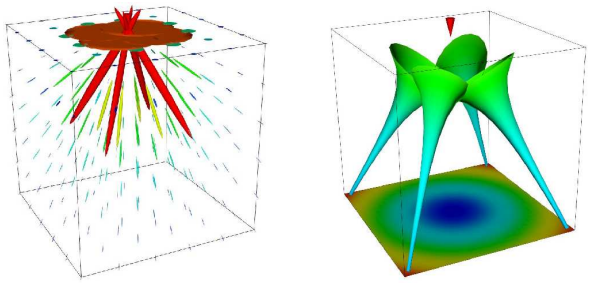


FIG. 9: Techniques de visualisation de tenseurs : ellipsoïdes et "hyperlignes" de courant

1.4.5 *Les données dispersées*

Il ne s'agit plus de données continues par rapport à l'espace : chaque information est précisément localisée. Il n'est donc plus nécessaire de faire appel à la notion de maillage. On peut citer comme exemples la simulation d'un système de particules en dynamique moléculaire, ou des relevés météorologiques sur le terrain, pour ce qui concerne des données immergées dans l'espace 3D "naturel".

Il peut aussi s'agir de toute série de données multidimensionnelles (nuages de points généraux). La visualisation se ramène alors à des projections 2D ou 3D, ou peut utiliser des techniques spécifiques comme les coordonnées parallèles[Ins98].

La technique de visualisation la plus utilisée est le "nuage de points". Pour chaque donnée discrète, on affiche un "glyphe" qui représente la ou les valeurs que l'on souhaite interpréter. Dans le cas qui va nous intéresser, on représente les données par des sphères.

Nous en reparlerons section 10.1.

1.5 LA VISUALISATION "HAUTES PERFORMANCES"

Une définition possible de la visualisation HP ("Hautes Performances") est de réserver le terme à la représentation de données qui nécessitent des ressources largement supérieures à celles normalement disponibles (station de bureau, avec un écran de taille classique). Cette notion implique bien entendu de conserver une interactivité correcte en dépit du nombre de points, de cellules maillées, ou de pixels affichés.

1.5.1 *HP par les données*

La taille des données peut être un facteur problématique. En effet, pour une vitesse donnée de chargement, plus la quantité d'informations est importante, plus le temps passé au transfert augmente.

Tous les ordinateurs fonctionnent à l'heure actuelle avec plusieurs types de mémoire pour cette raison : afin d'accélérer l'envoi des informations à calculer au processeur, on hiérarchise la structure mémorielle de la machine afin que la mémoire la plus rapide se trouve la plus proche de l'unité de calcul.

Le transfert entre la RAM (Random Access Memory), le processeur, et la carte graphique est le plus rapide que l'on puisse trouver dans un

ordinateur. C'est pourquoi il est intéressant que les données que l'on souhaite manipuler puissent tenir en mémoire RAM.

Si ce n'est pas le cas, on se retrouve avec un problème dit *out-of-core*, c'est-à-dire que toutes les données utilisées pour la visualisation ne tiennent pas dans le coeur "rapide" de la machine (processeur, RAM, et carte graphique).

La taille des données est généralement considérée comme un des facteurs les plus contraignants de la visualisation, avant la taille de l'affichage, et en particulier les étapes d'accès aux données dans un contexte de plus en plus multifichiers répartis sur machine parallèle.

1.5.2 *HP par l'affichage*

La taille de l'écran, comptée en nombre de pixels, est aussi un facteur qui peut relever des HP.

La visualisation scientifique utilise l'affichage graphique pour représenter des résultats scientifiques de simulations ou d'expériences. Il est donc nécessaire que cet affichage soit net et avec une définition capable d'apporter une résolution égale ou supérieure aux détails que l'on souhaite observer.

Il est important de bien faire la différence entre résolution, le pouvoir de séparation des détails, et définition, le nombre de pixels d'un système.

Il existe différents types d'affichage à haute définition. On peut citer les systèmes multi-écrans avec plusieurs projecteurs, et mono-écran avec plusieurs projecteurs. En général, même si le support est différent, on retrouve un dispositif multi-projecteurs comme source d'affichage. Cela est dû en grande partie à l'augmentation non linéaire du prix d'un dispositif en fonction de sa résolution. Il existe également des dispositifs multi-plans, qui permettent une meilleure immersion dans l'environnement virtuel (Exemple : un dispositif de type *Workbench*, avec deux plans orthogonaux, cf. figure 10).

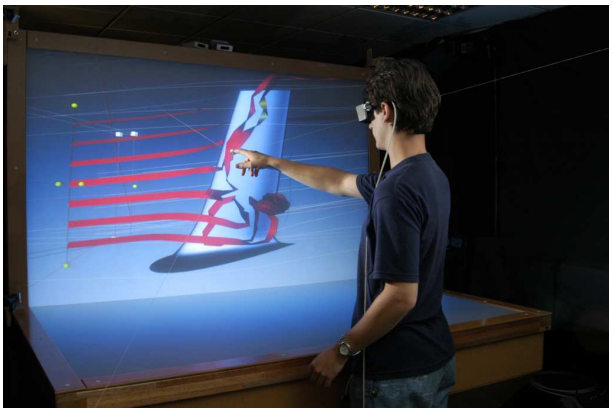


FIG. 10: Workbench de l'INRIA Grenoble

Comme la précision des résultats est de mise, toute forme de mise à l'échelle de l'image, d'anti-crénelage (*aliasing*), etc, est souvent proscrit. Il est donc nécessaire que le rendu des images se fasse à la résolution native (ou maximale) du système.

La capacité de rendu des cartes graphiques est globalement proportionnelle à leur puissance de calcul. Une très haute résolution pose donc quelques problèmes de dimensionnement, surtout si l'on veut que le taux de rafraîchissement soit adapté à l'application. De plus, pour le cas des murs d'images, chaque portion d'affichage doit afficher correctement sa partie de l'image, ce qui pose des problèmes de distribution des données et de passage d'informations (cf. chapitre sur le parallélisme).

Les systèmes permettant un affichage en stéréo sont souvent considérés comme HP par l'affichage, puisqu'ils nécessitent le double de pixels de rendu pour une même scène (cf. figure 11).

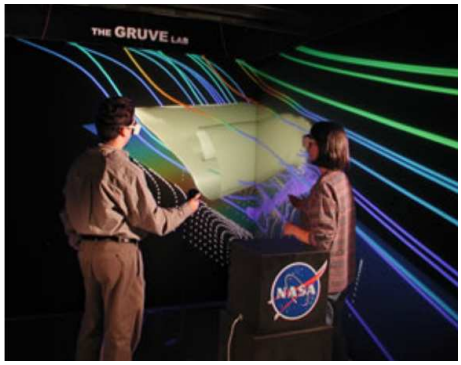


FIG. 11: Exemple d'HP par l'affichage : rendu multi-point de vue d'*Ensight* dans un CAVE

1.5.3 *HP par le rafraîchissement*

Le taux de rafraîchissement de l'image, ou FPS (Frame Per Second), donne le nombre d'images affichées par seconde par le système.

Bien que ce facteur soit souvent considéré comme secondaire en visualisation scientifique, surtout en comparaison avec la résolution ou la capacité du système à gérer de gros volumes de données, il peut néanmoins devenir un élément problématique dans certaines applications.

Dans d'autres domaines, en particulier le monde vidéo-ludique, on observe le phénomène inverse : il s'agit d'afficher le plus rapidement possible les éléments de la scène, en étant parfois moins regardant sur la précision des détails (méthode "quick and dirty").

Néanmoins, l'interactivité est de plus en plus présente dans le monde de la visualisation scientifique : la sensation d'interaction immédiate avec le système, en particulier avec la simulation, permet d'ouvrir d'autres horizons à ce domaine. On peut prendre comme exemple les simulations de docking[CV02], où l'expérience des biologistes est primordiale, la manipulation en temps réel de la simulation apporte donc un intérêt énorme à l'application. De plus, le mouvement de la scène peut apporter des informations supplémentaires à l'observateur (par exemple les mouvements de caméra permettent

une bien meilleure perception de la profondeur qu'une perspective statique[Bra84]), du moment que le rafraîchissement est suffisamment important pour donner une impression de fluidité.

On considère que le strict minimum pour une impression de fluidité est entre 10fps et 15fps, 0.1sec étant l'ordre de grandeur de l'effet de rémanence sur la rétine

Pour atteindre un tel taux de rafraîchissement avec des données de fort volume affichées sur beaucoup de pixels, il est souvent nécessaire d'avoir recours à des systèmes très performants, de réfléchir aux problèmes de latence et de vitesse de circulation des informations dans le système, voire de raisonner en temps contraint et de gérer des "budgets" de ressources.

1.6 LE CONTEXTE DE CETTE ÉTUDE

Le but de cette étude était d'apporter tous les éléments pour construire une solution d'exploration visuelle de simulations à grand nombre de particules en physique des matériaux. "Un grand nombre" signifie un ordre de grandeur de plusieurs dizaines de millions de particules.

Il s'agit bien évidemment d'un domaine d'étude extrêmement vaste. Nous avons donc dû restreindre cette étude.

Pour cela, nous avons demandé aux spécialistes de ce domaine, par l'intermédiaire de Laurent Soulard du CEA/DIF, quels étaient les besoins en matière de visualisation.

La réponse fut simple : avoir la possibilité d'explorer facilement et de façon interactive les données générées en amont par les simulations particulaires, et ce, sur un système à grande résolution, par exemple un mur d'images avec un cluster graphique.

Les contraintes étaient donc doubles :

DES CONTRAINTES SUR L'INTERACTION : l'exploration "facile", ou intuitive, sur un mur d'image implique de s'intéresser à l'interaction en environnement immersif (cf. le chapitre 3). De plus, cette solution s'adresse à des physiciens, qui n'ont le plus souvent aucune expérience des périphériques de réalité virtuelle. Il est donc important que le système d'interaction reste simple.

DES CONTRAINTES SUR L'INTERACTIVITÉ : les données explorées nécessitent le plus souvent plusieurs centaines d'heures de calcul, même sur des supercalculateurs. Elles sont donc "mortes" (pré-générées), ce qui interdit le pilotage interactif[Esn05]. En revanche, les besoins des physiciens impliquent quand même des techniques de visualisation "hautes performances", non seulement par les données (problème *out-of-core*), mais aussi par l'affichage (haute résolution). On peut alors difficilement se passer d'une architecture parallèle (cf. le chapitre 4).

2 VISUALISATION ET PHYSIQUE DES MATÉRIAUX

Ce chapitre présente rapidement le domaine de la Physique des Matériaux, et tente de décrire les principaux besoins en termes de visualisation scientifique.

2.1 UNE DÉFINITION RAPIDE

La physique des matériaux est la science qui étudie les propriétés et le comportement de la matière.

Il s'agit le plus souvent d'expliquer des phénomènes globaux (pour l'échelle étudiée), c'est-à-dire continus, à l'aide d'une représentation discrète (particules, atomes) de la matière.

2.2 DIFFÉRENTES ÉCHELLES POUR DIFFÉRENTS MODÈLES

2.2.1 *Nanoscopique*

Au niveau nanoscopique, les systèmes ne contiennent en général que quelques centaines d'atomes. Cette limite provient du coût élevé des calculs. Il s'agit typiquement de systèmes dont on étudie le comportement d'un point de vue quantique par des approches ab initio[BJM+03] , i.e. à partir de premiers principes et sans paramètres ajustables. Les atomes sont généralement représentés par deux types de données complètement différents : des noyaux ou ions, assimilés à des sphères, et des champs scalaires maillés, décrivant entre autre la densité électronique et la probabilité de présence des électrons.

En plus de la visualisation "atomistique" des noyaux, c'est-à-dire par des sphères colorées, il est alors intéressant de pouvoir afficher les champs sous forme d'isosurfaces, voire en rendu volumique (cf. figure 12).

2.2.2 *Microscopique*

C'est typiquement l'échelle d'étude de la dynamique moléculaire. Les particules, au sens de la mécanique "classique" newtoniennne, sont alors "simplifiées" dans leur représentation par rapport au nanoscopique, puisqu'elles sont la plupart du temps représentées par des sphères uniquement. En revanche, le nombre de paramètres associés à une particule peut être important : de l'ordre de 30 à 40.

De plus, les systèmes informatiques pouvant maintenant le permettre [BB92], les simulations sont souvent constituées d'énormément de particules : jusqu'à quelques centaines de millions, voire des milliards (cf. figure 13).

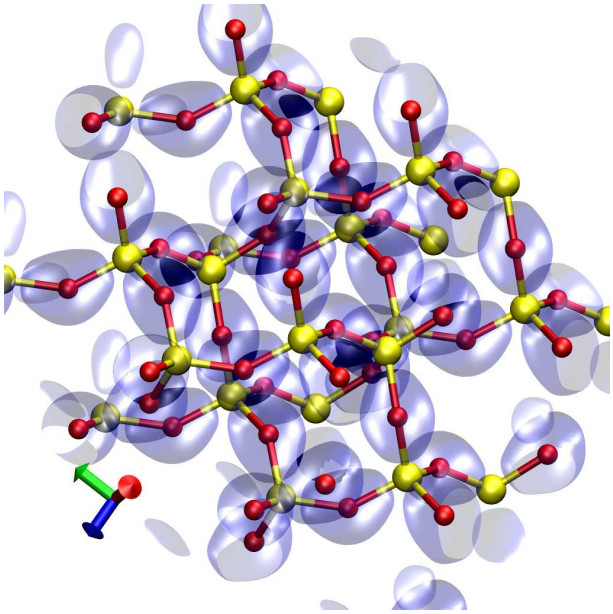


FIG. 12: Visualisation simultanées isosurfaces et noyaux d'un système ab initio.

Ces simulations peuvent être très longues, il n'est donc pas très pertinent d'y associer un système de *steering*, sauf sur des cas réduits, (en phase de mise au point d'un code de simulation ou de validation d'un calcul complexe avant lancement à grande échelle, par exemple). En revanche, il existe un réel besoin d'interactivité lors du dépouillement des données : le plus souvent, ce sont des comportements globaux de la matière qui sont recherchés, comportements qui dépendent de l'espace et du temps. De plus, les systèmes étant particulièrement volumineux (en nombre d'entités), il est assez utile de faire appel à des systèmes d'affichage restituant un maximum d'information, donc à haute résolution, comme les murs d'images.

Il s'agit donc du domaine ayant le besoin d'interactivité en visualisation particulaire le plus important : il s'agit d'afficher le plus rapidement possible un nombre très important de particules sur des écrans à grande résolution.

2.2.3 *Mésoscopique*

A cette échelle, la matière est peu discrétisée. Elle est étudiée sous forme de "grains", de dislocations, avec parfois des structures hybrides continues/discrète (cf. figure 14).

La visualisation pour ce domaine est très liée au modèle physique utilisé, il n'y a donc pas de représentation "classique" à l'aide de primitives simples. Cependant, un "grain" est souvent un assemblage polyédrique de mailles cristallines élémentaires (cf. figure 15).

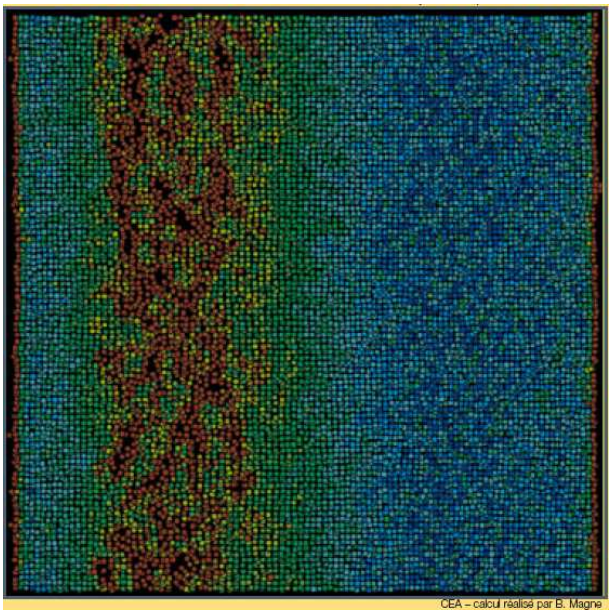


FIG. 13: Exemple de simulation avec un modèle de dynamique moléculaire (*Image crédit CEA*)

2.2.4 *Macroscopique*

Dans ce domaine, la matière est vue comme un milieu continu, régi par les équations de la mécanique des fluides ou des structures. Elle est donc représentée par un maillage.

On retrouve donc comme modes de représentation ceux déjà évoqués en section 1.4.

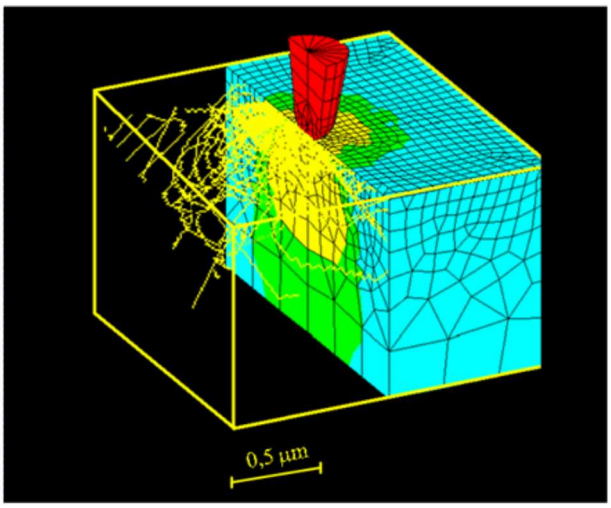


FIG. 14: Exemple de modèle hybride de dislocation couplant des méthodes dynamique moléculaire et d'éléments finis (*Image crédit CEA*)

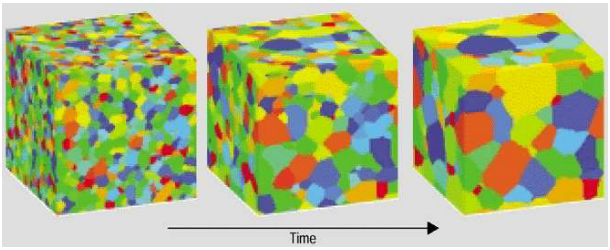


FIG. 15: Evolution dans le temps d'un système mésoscopique à "grains" (*Grain Growth Image Copyright (c) 2001 from Journal of Minerals, Metals and Materials(JOM)*)

2.3 PROBLÉMATIQUE DE VISUALISATION POUR LA DYNAMIQUE MOLÉCULAIRE

Si nous avons choisi d'axer cette étude sur la dynamique moléculaire, avec le laboratoire de dynamique moléculaire de Laurent Soulard (CEA/DIF), c'est avant tout pour les besoins spécifiques en matière de visualisation.

La principale utilisation de la visualisation était l'exploration de phénomènes globaux dans un système particulaire représentant un matériau. On peut prendre pour exemple la propagation d'une onde de choc dans la matière, avec des phénomènes de réflexion et de diffusion.

Les données ont donc les caractéristiques suivantes :

- Elles sont composées uniquement de particules, chacune d'entre elle était liée à une quarantaine de composantes (position, vitesse, énergies, etc).
- Afin de reproduire des phénomènes globaux, les simulations doivent être composées d'un nombre minimum de particules.

Or ce minimum est relativement important. L'ordre de grandeur va du million de particules, à la centaine de millions, et ces chiffres augmentent sensiblement tous les ans. La visualisation de l'ensemble du système, même pour une machine de type "station haut de gamme", est à la limite du *out-of-core*, et de toute façon le deviendra rapidement. Il nous a donc fallu mettre au point une solution évolutive, qui conserve une bonne efficacité si le système matériel augmente en puissance (on parle de solution "extensible", ou *scalable* en anglais).

– Une particularité de ces données est la représentation de la matière sous forme particulaire. Cela implique qu'une partie au moins de la simulation est considérée comme "dense", c'est-à-dire que les particules sont assez proches pour que le matériau soit considéré comme solide (ou liquide). Cette densité fait donc apparaître des solutions pour "alléger" le rendu de la visualisation, par exemple en mettant en place des techniques de gestion d'occultation car la densité des données opacifie le milieu, une part importante des particules étant masquée à l'affichage. (cf. section 4.3).

Rappelons que le but de cette visualisation est l'exploration de ces données pour l'interprétation de phénomènes globaux : il faut donc une certaine interactivité lors de cette phase exploratoire, en particulier un rafraîchissement minimum de l'affichage.

2.4 DOMAINES AUX PROBLÉMATIQUES DE VISUALISATION APPARENTÉES

2.4.1 *La biologie moléculaire*

La biologie moléculaire étudie les comportements et les interactions des molécules qui sont présentes dans les systèmes biologiques.

Le plus souvent, il s'agit de molécules organiques ne dépassant pas quelques dizaines ou centaines de milliers d'atomes.

Les données manipulées sont des atomes, dont on connait la position, ainsi qu'un certain nombre de paramètres, comme l'énergie cinétique ou potentielle. Ce nombre dépasse rarement une dizaine de paramètres. De plus, les systèmes étudiés étant des molécules, des liaisons sont associées aux atomes.

Bien qu'il arrive que la représentation des particules se fasse sans celle des liaisons, la plupart du temps, c'est une représentation "sphères-bâtons" qui constute le mode d'affichage de référence. Elle permet une vue détaillée de la molécule et de ses sites.

En revanche, il existe d'autres modes de représentations pour ces molécules : en effet, les molécules étudiées en biologie sont composées de séquences identifiées et dont les caractéristiques sont connues. On retrouve alors des visualisations incluant des flèches, des rubans, des tubes, par exemple (cf. figure 16), pour représenter des parties spécifiques de l'édifice complexe.

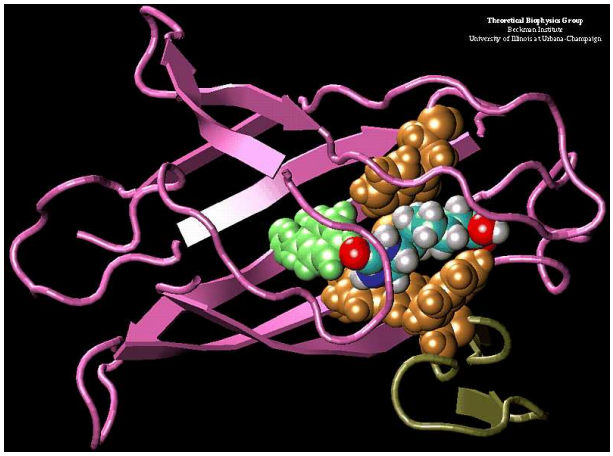


FIG. 16: Représentation mixte de deux protéines (avidine et biotine).

L'un des sujets les plus étudiés à l'heure actuelle est le repliement de protéines, et l'étude des sites actifs : il s'agit des modifications naturelles de la géométrie des protéines, et leur influence sur l'interaction entre certaines parties de cette molécule et d'autres agent biologiques (par exemple, le docking[CV02]). Pour ce type de système, la représentation sphérique est préconisée, puisqu'il s'agit d'étudier le comportement interatomique entre différentes molécules.

Certaines études sont presque entièrement basées sur l'expérience des experts dans ce domaine. Pour le docking par exemple, le nombre de possibilités de repliement de protéine est très important, et il est pour l'instant impossible de calculer l'ensemble des configurations pour savoir lesquelles peuvent réagir avec une autre entité. Pour ce genre de simulation, l'affichage n'est pas le paramètre critique : c'est l'interactivité, car c'est au biologiste de privilégier une géométrie au profit d'une autre, et cela pendant la simulation.

2.4.2 *La cristallographie*

La cristallographie est la science des cristaux, au sens large. Elle étudie : la formation, la croissance, la forme extérieure, la structure interne, et les propriétés physiques de la matière cristallisée. [Cos]

C'est donc un domaine un peu à part dans la physique des matériaux.

Bien que la périodicité, propriété intrinsèque de la forme cristalline, puisse faire supposer que peu d'atomes sont à représenter (redondance de l'information), il n'en est rien. En effet, bien que les systèmes étudiés soient spatialement périodiques, c'est par la répétition des structures dans l'espace qu'un certain nombre de phénomènes peuvent être observés (apparition de structures en tubes, par exemple).

En pratique, les systèmes étudiés en cristallographie sont localement périodiques, ou presque périodiques.

Les besoins sont donc très proches de la dynamique moléculaire.

3 PARLONS INTERACTION

Comme nous l'avons présenté en introduction, l'interaction est un des facteurs extrêmement importants de la visualisation scientifique, en particulier si les dispositifs matériels permettent une immersion importante.

L'interaction en evironnement immersif est un domaine très vaste, c'est pourquoi nous ne présenterons ici que la partie liée au contrôle d'application. Nous présentons ensuite les tentatives d'intégration de contrôle d'application en environnement immersif dans les solutions de visualisation scientifique.

Ce choix est justifié dans la deuxième section.

3.1 DÉFINITION

En s'inspirant des taxonomies de Hand, Hodges et Bowman [Han97] [BH99], on peut classer les différentes catégories d'interaction génériques en environnement virtuel immersif de la manière suivante :

- **La navigation**, à savoir le contrôle du point de vue de l'observateur ;
- **La sélection** d'objets ;
- **La manipulation** d'objets ;
- **Le contrôle d'application** ;

L'exploration, qui peut regrouper plusieurs des interactions précédentes, implique une visualisation ou une extraction d'informations, avec comme finalité l'interprétation des données concernées.

Un système immersif a pour but de plonger l'utilisateur dans un environnement virtuel où celui-ci cesse de se rendre compte de son environnement réel.

3.2 CHOIX DE L'ÉTUDE : LE CONTRÔLE D'APPLICATION

Lors de la phase de visualisation d'une simulation, énormément de paramètres doivent être pris en compte et réglés. Si cette simulation est représentée dans un environnement virtuel interactif, il est intéressant de pouvoir modifier à loisir lesdits paramètres, en particulier dans les phases d'activité exploratoire. C'est le champ du contrôle d'application.

Lors d'une étude préliminaire, nous avons remarqué qu'il existait extrêmement peu de solutions de contrôle d'application efficaces intégrées aux plateformes de visualisation scientifique. La navigation, la sélection et la manipulation sont souvent les premiers éléments implémentés, car ils sont directement liés aux périphériques spécifiques à la réalité virtuelle. Il s'agit des actions que l'utilisateur s'attend naturellement à effectuer dans un environnement virtuel immersif : se déplacer, appréhender un objet, le manipuler.

Le contrôle d'application est une action beaucoup moins naturelle, puisqu'elle consiste à modifier un paramètre précis d'un élément de

scène, ou de la simulation (ses résultats, voire son état et ses variables en cas de pilotage interactif), dans le cas de visualisation scientifique.

C'est pourquoi nous avons décidé de nous intéresser à cette problématique, avec les contraintes que nous avons précédemment citées (cf. chapitre 1).

3.3 LE CONTRÔLE D'APPLICATION

On regroupe, sous le terme contrôle d'application, l'émission de commandes dans le but de changer, par exemple, l'état du système ou le mode d'interaction, ainsi que l'édition des paramètres.[CFGP00]

Il s'agit donc d'un canal de communication entre l'utilisateur et l'application, basé sur des modalités plus ou moins complexes.

Lors de la manipulation de paramètres issus d'une simulation, il peut s'agir typiquement :

- de modifier la valeur d'une isosurface
- de modifier la correspondance entre une valeur et sa couleur
- de modifier un paramètre dans les préférences de l'application
- de changer de mode de sélection/manipulation

3.3.1 *Taxonomie*

Cette classification[CFGP00] utilise comme critère principal le mode d'émission de la commande. Les modes possibles sont rapidement détaillés ci-après, et par la figure 17.

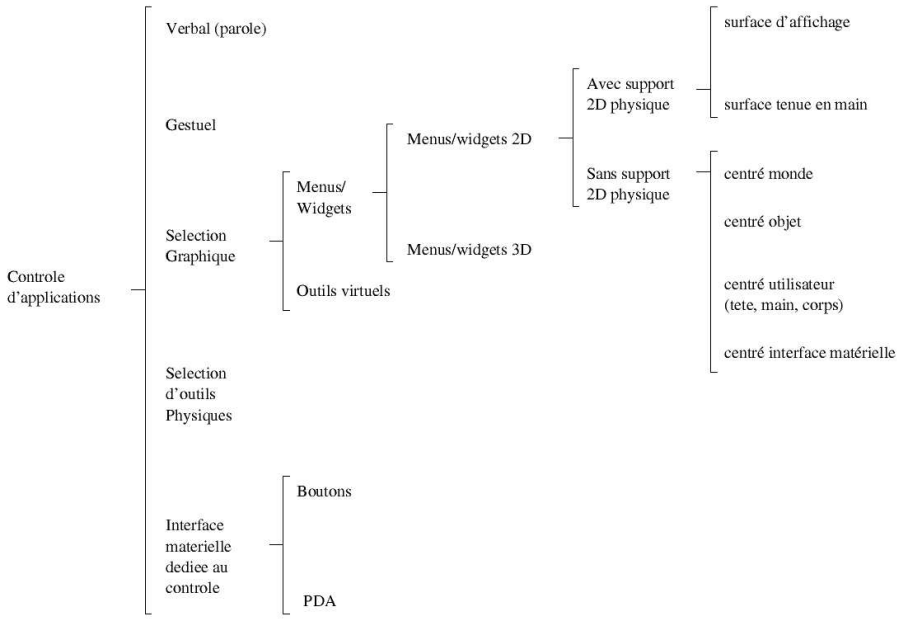


FIG. 17: Taxonomie selon Grosjean/Coquillart

Le mode verbal

Il s'agit d'associer au contrôle d'application un certain nombre de commandes verbales. En plus d'être très naturel, ce mode libère les mains et l'affichage. Néanmoins, il n'est pas souvent utilisé dans un environnement immersif, car plusieurs problèmes subsistent : à l'heure actuelle l'efficacité des logiciels de reconnaissance vocale progresse, mais reste assez limitée. De plus la question des langues force une spécialisation par langues du système. Enfin, l'utilisation de la parole comme contrôle d'application peut s'avérer difficilement utilisable en mode coopératif : non seulement les systèmes de reconnaissance peuvent avoir des problèmes de gestion de différents types de voix, mais aussi la discussion entre utilisateurs est un bruit qui n'est pas forcément facile à filtrer.

Le mode gestuel

Il s'agit d'activer une commande par geste ou posture. Deux technologies sont principalement utilisées pour cela : la reconnaissance de mouvement à l'aide de caméras, ou des gants de données, aussi appelées "pinch gloves". Néanmoins, sans retour graphique, l'apprentissage de ce genre de mode est assez difficile et peu intuitif. C'est pourquoi les systèmes de type *TULIP*[BW01] ou C^3[GC01], que nous détaillerons plus bas, couplent un mode graphique et un mode gestuel, afin de faciliter l'apprentissage : on a alors deux modes, un "novice" graphique et gestuel, et un "expert", sans support visuel.

Les "outils physiques" ("props" en anglais)

Il s'agit d'utiliser des outils spécifiques qui remplissent des fonctions déterminées. Un exemple classique est l'utilisation d'une tête de poupée[HPGK94] pour manipuler une tête virtuelle, associée à et une tablette symbolisant un plan de coupe pour des applications de neurobiologie. Souvent très efficace, ce mode est néanmoins très spécifique à l'application.

Les interfaces dédiées

Il s'agit de dissocier complètement l'interface associée au contrôle d'application du reste de l'interface homme / application. Un exemple des plus pertinents est par exemple l'utilisation d'un PalmPilot dans un CAVE[WDC99]. Bien qu'étant en général très adapté à l'émission de commandes, ce type de contrôle nécessite un matériel particulier. De plus, l'éloignement spatial (voire mental) entre la scène 3D de l'application et le contrôle nuisent souvent à l'immersion de l'utilisateur.

La sélection graphique

Il s'agit du type de contrôle d'application le plus utilisé : l'utilisation d'une représentation graphique pour sélectionner des commandes. On peut distinguer les menus, où le système de commandes est dissocié

du pointeur, et les outils virtuels, ou *widgets*, où une fonction est associée directement au pointeur.

Dans un environnement 2D, les menus sont devenus un standard en matière de contrôle d'application. Néanmoins, cela pose quelques problèmes de passage à un environnement 3D : un problème de sélection et un problème de positionnement. Détaillons ces deux aspects :

LA SÉLECTION de commande est intrinsèquement 1D, et l'ajout de dimension supplémentaire est souvent plus une gêne qu'un réel progrès. C'est pourquoi l'ajout d'un support physique sur lequel les menus sont affichés, et qui réduit donc le nombre de degrés de liberté, peut être intéressant.

LE POSITIONNEMENT : en l'absence de support physique, le positionnement n'est plus fixé. On a alors les choix suivants :
- positionnement centré monde : le positionnement est placé librement dans l'espace virtuel. C'est le moins contraignant, mais il pose des problèmes d'occultation par la scène.
- positionnement centré objet : il est souvent utilisé lors de l'utilisation de widgets.
- positionnement centré utilisateur : il peut être centré corps, main, tête etc. Le centrage main (ou objet tenu en main) est tout particulièrement intéressant car il permet de revenir à une sélection 1D.
- positionnement centré interface matérielle : par exemple, dans un workbench, il s'agit d'une position fixe par rapport aux écrans.

Enfin, il est assez rare de trouver des solutions de menus ou widgets qui utilisent véritablement la 3D, sans passer par une retranscription de menus 2D.

3.3.2 *Exemples de contrôle d'application*

Exemples de contrôle d'application en 2D

Les "gestures" font référence aux "mouse gestures", qui sont des combinaisons de mouvements de la souris et de clics que le logiciel utilisé reconnait comme une command spécifique. On peut considérer que la première mouse gesture est le "drag", inventé par Apple pour invoquer la commande "déplacer".

- Le type de contrôle d'application le plus répandu actuellement est très certainement le *WIMP* (Windows, Icons, Menus, Pointing), regroupant le système de fenêtres, menus déroulants et icônes d'une station de travail. Il se révèle relativement adapté aux environnement 2D (de type bureau virtuel). Néanmoins, il est peu maniable dans des environnement 3D, la troisième dimension posant de gros problèmes de sélection.[CHWS88]
- Le PieMenu : aussi appelé "menu camembert", il s'agit d'un menu en secteurs avec sélection radiale. Il est beaucoup plus efficace que son concurrent direct (le menu linéaire) pour un nombre de choix de sélection allant jusqu'à 12, nombre maximum de secteurs généralement conseillé. Plusieurs évolutions peuvent être notées, comme par exemple les *Marking menus*[KB93], souvent considérés comme des PieMenus hiérarchiques, qui offrent ainsi un mode

"expert" sous la forme de séquences de mouvement (*gestures*) (cf. figure 18).

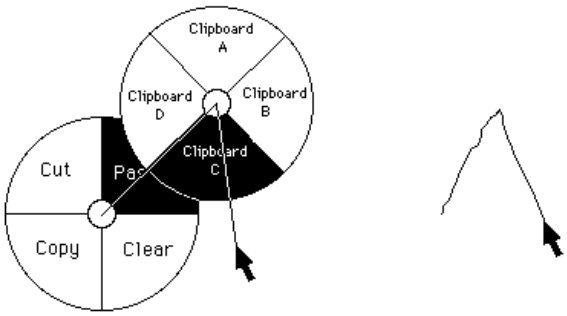


FIG. 18: Un exemple de *Marking Menu*

– Il existe également des tentatives d'interaction multimodales pour le contrôle d'application. On peut notamment citer le *Toolglass*[BSP+93], composé d'un curseur manipulé par la main dominante pour la sélection de l'objet, et d'un plan semi-transparent manipulé par l'autre main (par exemple à l'aide d'un trackball), destiné à sélectionner la fonction de manipulation de l'objet (cf. figure 19).

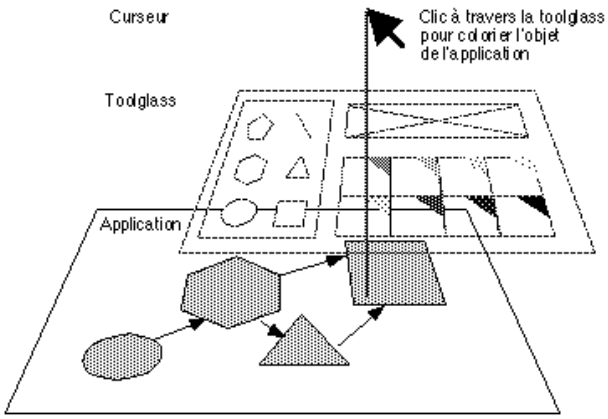


FIG. 19: Un exemple de *Toolglass*[BSP+93] : palette de formes, de couleurs et outil de suppression

Exemples de contrôle d'application 3D

Parmi les tentatives de contrôle d'application complètement 3D, on peut citer :

– Un menu en anneau[WD00], centré main et utilisant un tracker de type stylo. La sélection s'effectue à l'aide d'un rayon partant du centre et orienté par rotation du poignet (cf. figure 20).

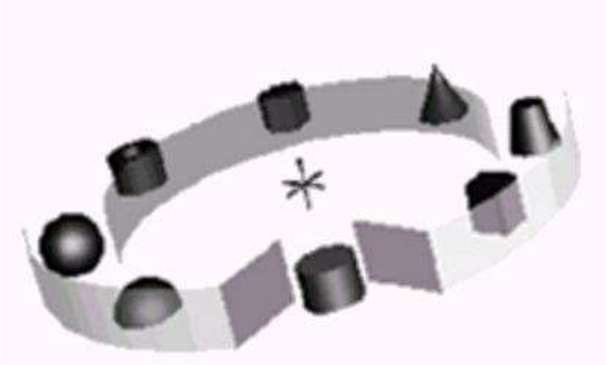


FIG. 20: Le menu en anneau

– L'interface TULIP[BW01], système utilisant des *pinchgloves*. Les menus s'affichent au bout des doigts, avec la possibilité d'avoir une hiérarchisation (interface centrée utilisateur) (cf. figure 21).

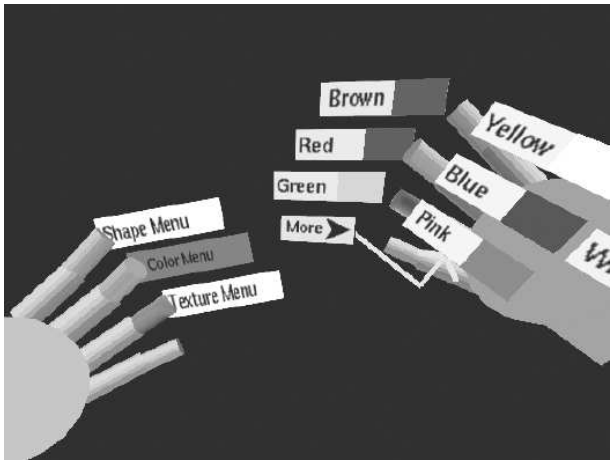


FIG. 21: L'interface *TULIP*

– Le C^3[GC01] (Control and Command Cube) est un cube composé de 27 sous-cubes représentant les commandes. La sélection est donc une extension de celle par orientation prônée par Kurtenbach[KB93], et permet aussi un mode expert par des *gestures* (cf. figure 22).

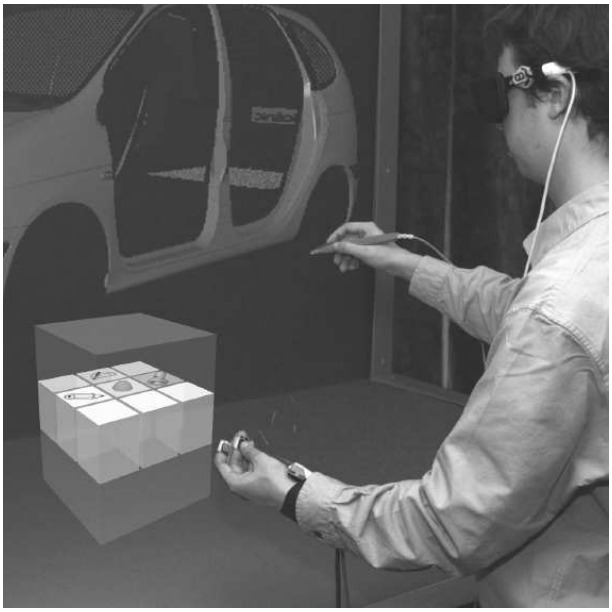


FIG. 22: Le C^3

3.4 PARMI LES SOLUTIONS DE VISUALISATION SCIENTIFIQUE

Les environnements immersifs devenant de plus en plus séduisants pour la visualisation scientifique, les logiciels de visualisation scientifique généralistes les plus utilisés ont adopté des extensions pour ce type de configuration (gestion de périphériques de sélection/manipulation/navigation, stéréoscopie, rendu multi-point de vue pour les CAVE etc). Néanmoins, peu se sont risqués à proposer des interfaces 3D réellement originales ou immersives, qui nécessitent souvent une refonte complète du système d'interface d'un logiciel.

- *Avizo*[Avi], anciennement Amira S[Amia] avec son extension RV, propose des menus linéaires 3D flottants : comme il est écrit ci-dessus, la sélection de ce genre de menus est relativement difficile, même avec l'aide d'un retour haptique.
- *Ensight*[Ens] offre par ses extensions RV des menus 2D simplement incrustés dans le mode plein écran (mécanisme dit de "head-up macros". Il s'agit d'une analogie avec les affichages tête haute incrustés dans les cockpits d'avion). Cela a l'avantage d'être parfaitement lisible et facilement manipulable par des périphériques de bureau, mais nuit souvent à l'immersion et à la sélection par des périphériques 3D (tracker, etc).
- *AVS*[AVS] propose des menus modulaires, permettant l'un ou l'autre des deux modes ci-dessus (cf. figure 23).

Les interfaces haptiques regroupent les interfaces manuelles tactiles et à retour d'effort[ePSoo].

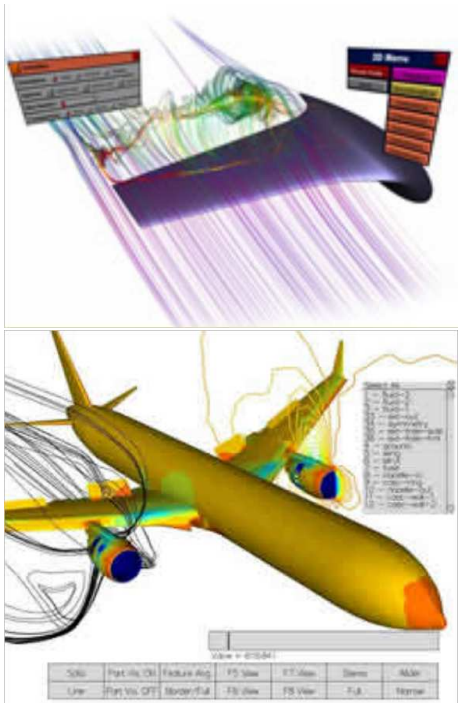


FIG. 23: Les menus flottants d'*Avizo* et les menus incrustés d'*Ensight*

3.5 LA MANIPULATION DIRECTE

La manipulation directe est une approche regroupant des moyens qui donnent à l'utilisateur la possibilité de manipuler des objets dans des environnements virtuels d'une façon naturelle et intuitive. Bryson dans [Bry04] en donne une analyse très fouillée dans le contexte de la visualisation scientifique. Cela permet également d'explorer de façon plus efficace des jeux de données multidimensionnelles complexes.

La manipulation directe peut être vue comme un essai de transposition des gestes humains du monde réel dans un univers abstrait. On admet[Bry04] qu'un tracker de position et d'orientation suffit pour reproduire de façon acceptable les comportements de la main. Vient ensuite le problème de la sélection de la fonction de manipulation : c'est là que généralement l'on reboucle avec les systèmes de contrôle d'application. Bryson propose comme exemple d'utiliser un système de boutons ou de gants pour effectuer une telle sélection.

On retrouve par exemple cette notion dans les classes de *Manipulators* de Open Inventor™[OIV] (cf. figure 24).

Plus récemment, la plateforme VTK/Paraview[Para] propose un système de *Widgets3D*, sous forme d'une suite de classes C++ qui aident à la manipulation d'objets de la scène (cf. figure 25).

On trouve de récentes tentatives d'intégration d'éléments génériques de RV dans VTK, en particulier VR-VTK[KvL07]

Par exemple, le widget 3D *boxWidget* est un outil unique qui permet le déplacement, la rotation, et la mise à l'échelle d'un objet. Son utilisation se fait à l'aide d'une souris (2D).

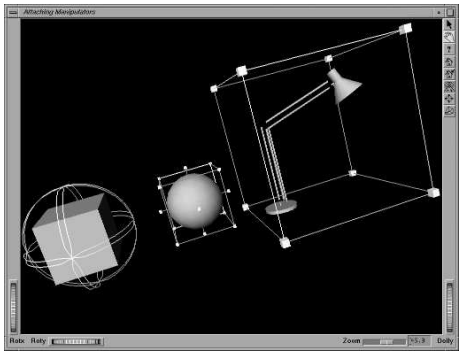


FIG. 24: Exemples de *Manipulators* dans Open Inventor™

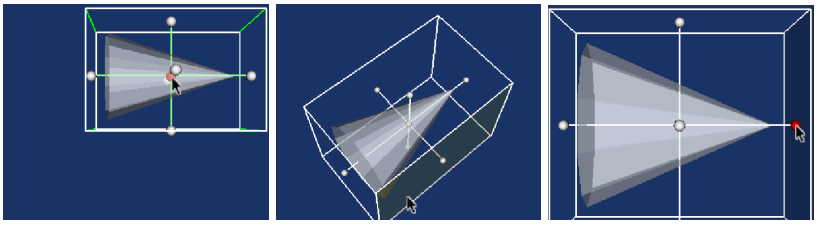


FIG. 25: Exemple de *widget 3D* dans VTK : le boxWidget, et son utilisation : positionnement (à gauche), rotation (au milieu), mise à l'échelle (à droite)

Il n'existe en revanche pas de classe intégrée à VTK permettant l'intégration de systèmes de Réalité Virtuelle (pas de notion de *tracker*, par exemple).

Notons qu'Avizo, précédemment cité, est basé sur OpenInventor et offre donc ses Manipulators sur ces objets de haut niveau de la visualisation scientifique, ce qui est au final proche des Widgets3D de VTK.

4 INTERACTIVITÉ

Le processus de rendu d'une scène complexe telle qu'un grand ensemble de particules est extrêmement coûteux en terme de calcul et de mémoire. Ce chapitre présente les principaux concepts d'architectures et d'optimisations qui permettent d'atteindre des taux de rafraîchissement assez élevés pour espérer une exploration interactive de telles scènes.

Nous abordons ici :

- Les techniques de rendu de primitives géométriques, en particulier OpenGL
- Les optimisations de rendu des ombres
- Les architectures parallèles de rendu
- Des méthodes et systèmes que l'on peut qualifier d'adaptatifs

La fin du chapitre est consacrée aux solutions logicielles les plus connues de la visualisation scientifique, permettant de prendre en compte les grands ensembles de particules ou atomes.

4.1 RENDU DES PRIMITIVES ÉLÉMENTAIRES

4.1.1 *OpenGL*

OpenGL[SA06] est un environnement de développement pour des applications graphiques interactives 2D et 3D. C'est dans ce domaine l'API (Application Programming Interface) la plus utilisée et la plus supportée, toutes plateformes confondues. Le rendu se base sur la projection géométrique des éléments de la scène dans un espace dit "image", permettant la représentation d'objets en perspective.

Architecture

OpenGL est en fait une machine à état. Chaque commande envoyée à l'interface OpenGL, que ce soit une modification des paramètres, ou un ordre de rendu d'un objet de la scène, se fait par une demande de changement de l'état de la machine OpenGL.

Nous ne présentons ici (figure 26) qu'une version simplifiée de l'architecture OpenGL.

Le processus de rendu OpenGL est le suivant : une fois que l'application qui utilise OpenGL a envoyé les informations nécessaires à l'affichage des éléments de la scène, elle demande un rendu de chaque élément.

OpenGL passe alors par les phases suivantes :

- une phase de calculs préliminaires pour les lumières et les matériaux (*Transform and Lighting*),
- une phase de calcul de la forme des éléments de la scène projetée dans l'espace image,
- une phase dite de "rasterisation", qui effectue une opération de tramage de l'image finale. Cela a pour effet de représenter les éléments de la scène en pixels.

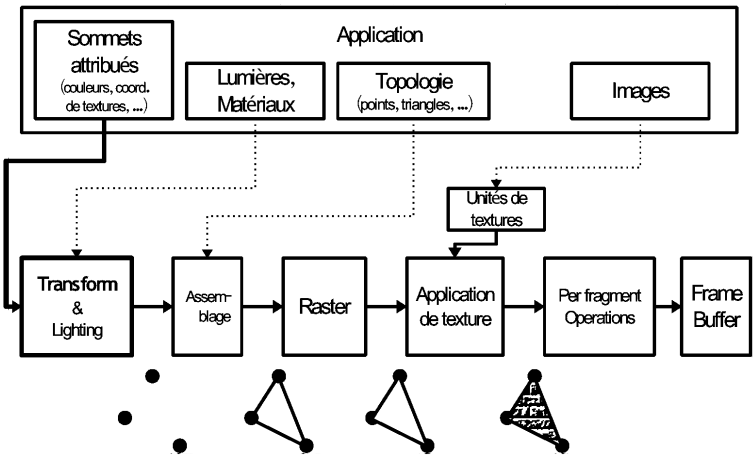


FIG. 26: L'architecture OpenGL

– une phase d'application de textures aux éléments de la scène,
– une phase de calcul de la valeur (couleur et profondeur) des pixels de la phase de rasterisation,
– enfin la phase d'écriture dans le framebuffer, qui correspond à une copie de l'image finale de rendu.

Les éléments de la scène OpenGL

L'élément de base de tout objet d'une scène 3D OpenGL est le sommet (*vertex*). Grâce à cette brique de base, OpenGL décrit les éléments suivants :
– les points,
– les lignes,
– les triangles,
– les polygones, qui sont en fait décomposés en triangles en interne.

Ce sont les seules formes géométriques que connait OpenGL. Les triangles peuvent être colorés soit par des couleurs unies, soit par des interpolations (linéaires) de couleurs, soit par des textures qui sont plaquées sur ces derniers.

Shading Language

A partir de la version 2.0 d'OpenGL, apparaît une modification de l'architecture (cf. figure 27), qui rend celle-ci beaucoup plus modulaire. Cette évolution est en fait conjointe de celle des architectures de cartes accélératrices graphiques (GPU = Graphical Processing Units), qui deviennent de plus en plus programmables.

Les phases de *Transform and Lightning* et d'application de texture peuvent alors être remplacées par des phases entièrement programmables : les *Vertex shaders* et les *Fragment shaders*. Le *Vertex shader* est exécuté au début de la phase de rendu d'un élément donné de la scène, et une seule fois pour chaque élément. Le *Fragment shader* est exécuté

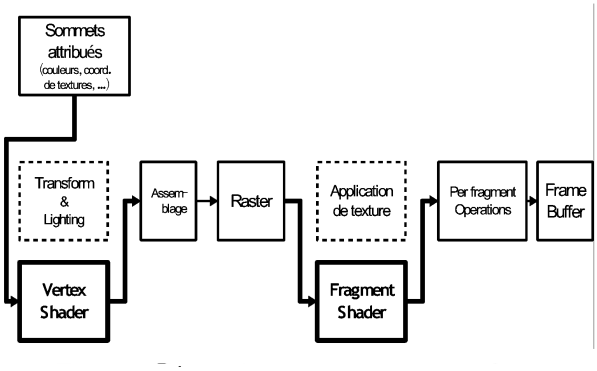


FIG. 27: L'architecture OpenGL avec les shaders

pour chaque pixel qui compose un élément, et a accès aux résultats du Vertex shader.

Les Vertex et Fragment shaders utilisent un langage spécifique, le Shading Langage, relativement proche du langage C[Kes06].

Cette technique est à l'origine destinée à effectuer des rendus d'objets non décomposables en polygones.

Les Shaders peuvent désormais être utilisés pour effectuer des calculs génériques : le GPGPU (General-Purpose computation on GPUs), qui est devenu un domaine d'activité intense. On peut citer la bibliothèque CUDA[CUD] de NVidia. Le Khronos Group, chargé de la standardisation de l'OpenGL, travaille également sur OpenCL (Open Computing Language), qui vise à généraliser la programmation GPGPU.

4.1.2 *Le rendu d'une sphère en OpenGL*

Nous allons présenter ici les principales méthodes de représentation de sphères en OpenGL.

Par un point

La représentation la plus simple d'une sphère en OpenGL est un point, qui est exprimé par un disque de couleur uniforme. Elle a le mérite d'être la plus rapide, puisque représentée par un seul vertex et une taille (qui représente le diamètre du point).

Cependant, cette représentation ne fait pas apparaître le fait qu'une sphère est un volume, et cela peut être perturbant pour l'observateur (cf. la section Les Ombres).

Par des polygones

Il s'agit de la méthode classique pour représenter des formes complexes en OpenGL. L'enveloppe est décomposée en un ensemble plus ou moins important de triangles. Plus le nombre de triangles utilisés est grand, plus la représentation de la forme est précise (cf. figure 28).

Cette représentation a l'avantage de fournir une véritable représentation, certes approchée, mais 3D d'une sphère. On peut alors y appliquer des effets de lumières, d'intersection entre sphères, etc.

Le problème est toutefois l'augmentation de la complexité de la scène : une scène avec N sphères n'est plus composée de N éléments, mais de N * P éléments, avec P le nombre de triangles par sphère.

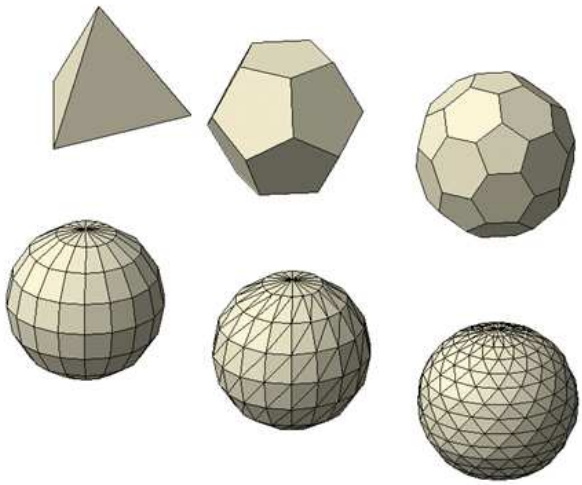


FIG. 28: Sphères représentées avec différents nombres de polygones

Par un PointSprite

Cette technique permet de réduire la représentation d'une sphère à un élément, tout en résolvant partiellement les problèmes de représentation 3D de la scène.

L'idée est de plaquer sur un point une texture (*PointSprite*) représentant une image de sphère rendue par un autre procédé.

Ceci permet de donner l'impression d'un rendu de sphère, mais reste relativement limité, en particulier :

- en cas de chevauchement de deux sphères, les PointSprites étant surfaciques, ils ne s'intersectent pas : l'un est représenté par dessus l'autre. L'observateur interprète alors que les sphères sont assez éloignées pour ne pas se toucher, bien qu'elles soient proches quelque soit la position de ce dernier. Ce phénomène peut être assez perturbant.
- la résolution de la texture est transmise à la sphère : si l'on utilise une image à faible résolution pour la texture, et que l'on effectue un gros plan sur une sphère, cette dernière est alors représentée sur une grande partie de l'image finale, et un effet de pixellisation apparaît.
- Les effets de lumière, principaux facteurs de perception du volume, ne dépendent que de la texture, et non de la lumière de la scène. Il est donc impossible de les modifier.

Par un shader

C'est la manière la plus aboutie pour représenter des formes difficilement décomposables en polygones.

L'élément de base est un point, mais on rajoute un couple Vertex-Fragment shader :

– le Vertex shader calcule la position du centre et le rayon de la sphère en espace image.
– pour chaque pixel, le Fragment shader calcule la profondeur et la couleur du pixel en fonction de sa distance par rapport au centre de la sphère, le tout en espace image (cf. figure 29).

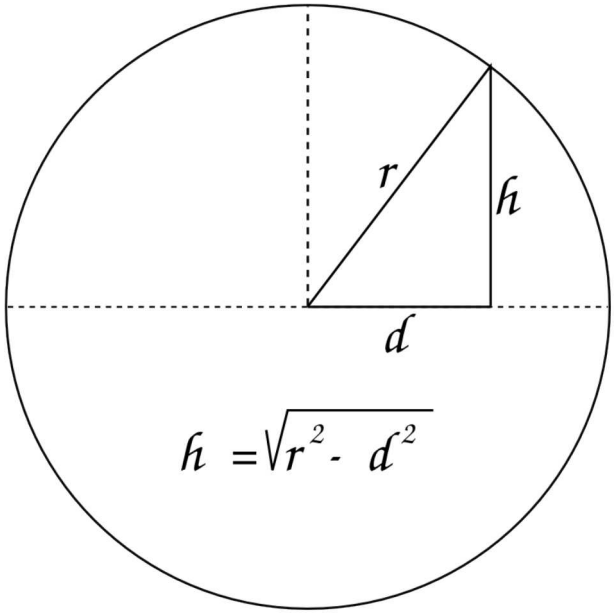


FIG. 29: Calcul de la profondeur/hauteur en fonction de la distance au centre : r est le rayon de la sphère, d la distance au centre, et h la hauteur.

On a alors une véritable représentation d'une sphère par un seul élément, qui prend en compte le caractère volumique de cette dernière.

Cette méthode de représentation devient procédurale, il s'agit bien ici d'un programme (ou de portions de programmes, respectivement attachés aux Vertex et Fragment shaders) qui contiennent implicitement la description des objets, et plus des paramètres fixés déclarativement.

4.2 LES OMBRES

Comme dit plus haut, la représentation des ombres apporte des informations très utiles pour la perception de la profondeur des éléments d'une scène 3D (cf. figure 30).

Nous présenterons ici seulement les modèles qui nous semblent les plus pertinents pour la représentation efficace de sphères. Il en existe bien d'autres, mais qui sont le plus souvent trop coûteux en ressources pour un affichage interactif d'un grand nombre d'objets.

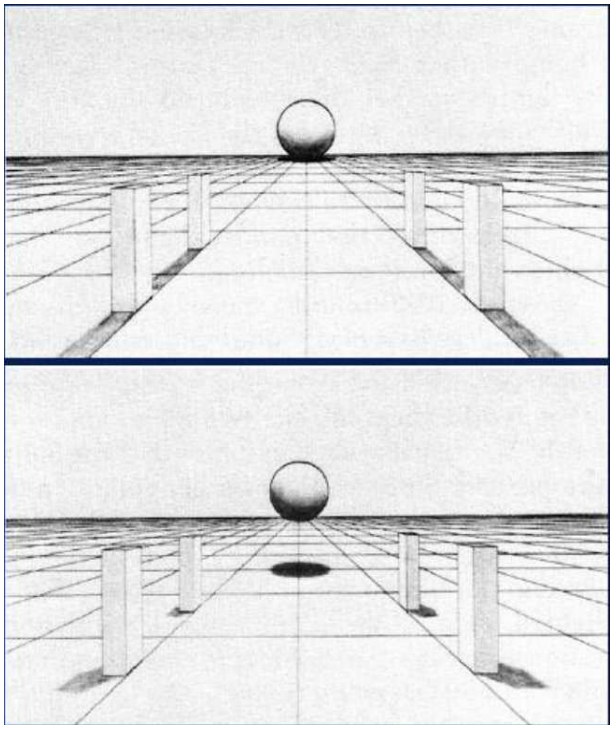


FIG. 30: Importance des ombres pour la perception de la profondeur : l'interprétation de la scène est totalement modifiée par un simple changement d'ombres portées.

4.2.1 *L'ombrage global*

Il s'agit de prendre en compte, lors du calcul d'illumination, la présence d'objets sur la trajectoire de la lumière, réduisant totalement ou partiellement l'intensité lumineuse. Cela permet en particulier une perception globale grandement facilitée de volumes complexes.

On peut citer, parmi les modèles les plus connus qui intègrent ce phénomène, le *Raytracing*, et les modèles de radiosité :

- Le *raytracing*[Exc90], ou lancer de rayons, simule le parcours inverse de la lumière des sources de lumières vers l'observateur. Il ne prend en compte que les phénomènes modélisés en optique géométrique (réflexion et réfraction).
- Les modèles de radiosité[GTG84] utilisent un modèle de propagation de la lumière issu des transferts thermiques, calculé grâce à des méthodes d'éléments finis.

On peut citer comme implémentations de raytracing en temps réel OpenRT[Opea], et sa version commerciale OpenRTRT commercialisée par Mercury Computer Systems et InTrace[Opeb]

Or ces modèles sont extrêmement coûteux en calcul, et à l'heure actuelle rares sont les solutions interactives de visualisation scientifique qui les implémentent. Et pour cause : prenons comme exemple un lancer de rayons pour un système de n sphères. Pour chaque élément, et pour chaque source de lumière, il faudrait effectuer au minimum $(n-1)$ tests, sans compter le calcul de la portion de la surface visible de la sphère à représenter. On a alors au strict minimum

une complexité en $O(n^2)$, qui déjà est innaceptable pour un grand nombre d'éléments.

Il existe néanmoins des méthodes de précalcul d'ombres pour des scènes où les objets sont fixes. On peut par exemple citer l'ambient occlusion[TCM06]. Cette technique prend en compte le fait que l'illumination est, pour chaque point d'un objet, fonction de la géométrie de la scène. On précalcule alors, pour chaque élément, une "carte d'occultation", stockée dans une texture, qui donne une estimation de l'apport de la lumière ambiante à l'illumination (cf. figure 31).

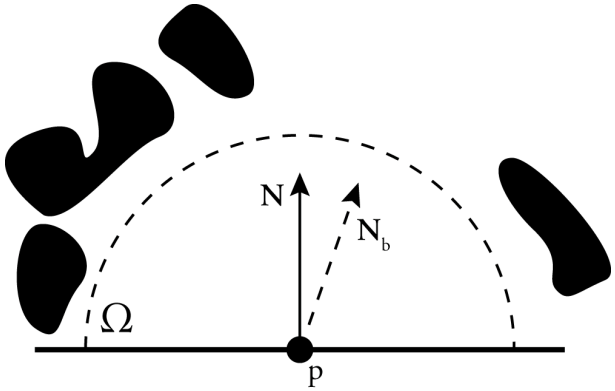


FIG. 31: Calcul de l'ambient occlusion pour un point de la surface d'un objet

Il s'agit bien sûr d'une approximation, mais qui reste suffisamment efficace pour aider à la perception de volumes complexes (cf. figure 32).

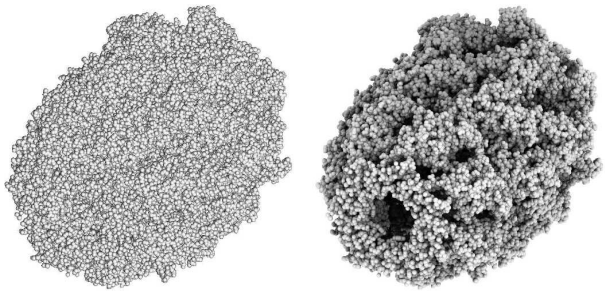


FIG. 32: Perception de la structure d'une molécule avec (droite) et sans (gauche) ambient occlusion

4.2.2 *L'envoi d'informations à la carte graphique*

Il s'agit d'un des principaux goulets d'étranglement d'une solution de visualisation. En effet, pour être affichées, les données de la scène doivent d'abord passer en mémoire centrale, puis être envoyées à la carte graphique. Nous présentons ici les principales méthodes OpenGL d'envoi de données. Nous verrons plus loin comment réduire la quantité d'information destinée au rendu.

glBegin / glEnd

Aussi appelé mode "immédiat", c'est la méthode la plus simple pour envoyer des ordres de rendu d'éléments. Toutes les opérations de description de primitives entre les instructions glBegin et glEnd sont interprétées par la machine OpenGL et transformées en pixels.

Le principal problème de cette méthode est la volatilité des données envoyées à la carte : ces dernières ne sont pas stockées, elles sont directement interprétées. Il est donc conseillé d'utiliser ce principe seulement lorsque la plupart des informations vont être modifiées entre deux rendus.

Display List

OpenGL permet de créer des listes de primitives, appelées Display Lists. Les instructions ne sont envoyées qu'une seule fois, et elles sont stockées en mémoire graphique.

Cette méthode est maintenant relativement désuète, et ce pour deux raisons : tout d'abord, il est impossible de modifier le contenu d'une Display List une fois celle-ci créée. Ainsi, si l'on doit modifier une partie de son contenu, il faut renvoyer toutes les informations. Ensuite, même avec des cartes graphiques récentes, elles sont relativement limitées en nombre d'éléments qu'elles peuvent mémoriser (de l'ordre de quelques millions de primitives, ce qui n'est pas suffisant pour les cas qui nous intéressent).

Vertex Buffer Object

La méthode préconisée pour la gestion de larges jeux de données en mémoire graphique est le *Vertex Buffer Object* (VBO)[NVI03]. Il se présente sous la forme d'un tableau de vecteurs à 1, 2 ou 3 coordonnées flottantes. On peut le remplir à l'aide d'un tableau similaire en mémoire centrale, de façon partielle ou totale (à la différence des Display Lists).

Il existe également des instructions permettant à la machine OpenGL d'utiliser tout ou partie du VBO comme base d'informations pour la génération de primitives. Il s'agit donc d'un moyen relativement modulaire de gérer la mémoire de la carte graphique.

Note importante : des tests de performance ont été effectués sur les différentes méthodes d'envoi de données dans un VBO. Nous prenons ici le cas d'une carte relativement ancienne, la Nvidia Quadro FX 4500, mais avec les derniers pilotes disponibles. La méthode glBufferData d'allocation de mémoire GPU et de copie totale des données offre une excellente efficacité. En revanche, la méthode glBufferSubData, qui permet la modification partielle des données du VBO, est d'une lenteur désastreuse. C'est également le cas si l'on tente d'utiliser des méthodes telles que glMapBuffer, qui permet d'accéder à la mémoire graphique via un pointeur en mémoire centrale.

Ce comportement nous porte à croire que ces générations de cartes graphiques ne sont pas prévues pour gérer une partie seulement d'un buffer qu'elles allouent. Pour cela, le driver semble récupérer

l'ensemble du buffer, modifier la copie, et renvoyer le tout à la carte graphique.

Il est donc important d'éviter d'utiliser les fonctions de modification partielle d'un VBO. Il est recommandé de partitioner les données en plusieurs VBOs, que l'on utilisera, ou détruira, selon le besoin.

4.3 PARALLÉLISME DE RENDU

4.3.1 *Classification des architectures de rendu parallèle par le tri*

Généralités

Molnar et al.[MCEF94] proposent une classification des systèmes de rendu parallèle suivant l'endroit du pipeline de traitement graphique qui est parallélisé. Cette classification prend en compte l'emplacement du tri (*sort*) des primitives géométriques dans le pipeline, 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* (cf. figure 33).

SORT-FIRST : la redistribution a lieu durant la phase géométrique. Les primitives brutes sont affectées arbitrairement aux unités de calcul (processeurs). Les primitives sont alors pré-transformées, puis, en fonction de leur position dans l'espace-écran, elles sont redistribuées, via le 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.

SORT-MIDDLE : les primitives sont affectées arbitrairement à des processeurs géométriques et transformées. Elles sont ensuite redistribuées à des processeurs de tramage (comme en *sort-first*, 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.

SORT-LAST : 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 *sort-last* 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).

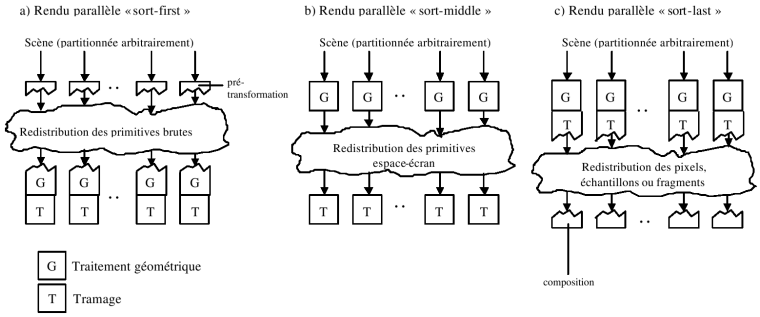


FIG. 33: classification des systèmes de rendu parallèle

Avantages et inconvénients des architectures "sort"

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 du *sort-first* et *sort-middle* 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 ci-dessous résume les grandes caractéristiques des trois méthodes :

	Avantages	Inconvénients
sort-first	Chaque noeud implante tout le pipeline pour sa région d'écran	Sensibilité au déséquilibre de charge de travail
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'à la composition des pixels Moins sujet au déséquilibre de charge	Le trafic en pixels peut être très important (nécessité d'un débit élevé)

En pratique, les cartes graphiques actuelles implantant l'ensemble du pipeline graphique, il est difficile de réaliser des architectures *sort-middle*. C'est pourquoi il n'en sera pas question par la suite.

Les deux principales solutions parallèles spécialisées dans la visualisation atomistique, Atomeye[Li05] et Atomsviewer[Sha02], se basent sur des architectures *sort-first*. En effet, dans le cas de l'utilisation d'un mur d'image, ou plus généralement d'un système d'affichage avec une résolution élevée, un important problème de débit de pixels peut rendre le *sort-last* moins performant que le *sort-first*.

Néanmoins, le *sort-first* possède un fort handicap dans le domaine de l'équilibrage des charges entre les noeuds : en effet, si la scène se concentre sur une petite partie de l'écran, les noeuds en charge du rendu des parties où rien n'est à afficher seront inutilisés. Cela, avec la forte augmentation du débit des réseaux des clusters et des entrées/sorties des cartes graphiques, permet d'envisager l'architecture *sort-last* comme concurrent sérieux au *sort-first*.

4.3.2 *Les solutions de sort-last pour les murs d'image*

L'efficacité du *sort-first* est moins liée au nombre de pixels à rendre que le *sort-last*. C'est pourquoi on le retrouve dans la plupart des architectures basées sur des murs d'images.

Chromium

Chromium[HHN+] est une bibliothèque opensource de rendu parallèle. Il s'agit d'un outil extrèmement générique, qui se base sur l'interception des commandes OpenGL, qui seront ensuite transférées en streaming vers les différents systèmes s'occupant du rendu. Elle permet de mettre en place différentes stratégies, en particulier le sort-first et le sort-last, sur la plupart des architectures, matérielles ou logicielles.

De par sa généricité, Chromium est considéré comme la solution classique pour le rendu *sort-first* et *sort-last* sur un mur d'image. Mais il existe des solutions beaucoup plus performantes.

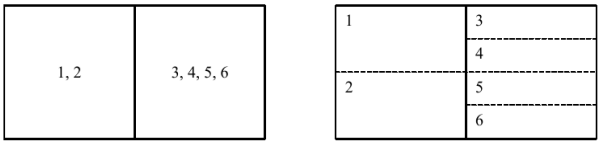
Ice-T

Ice-T (Image Composition Engine for Tiles) fait partie des rares tentatives d'architectures *sort-last* pour de telles configurations.

Le principe d'Ice-T est le suivant[MWP01] : la parallélisation en sort-last permet une distribution de la charge de travail pour le rendu beaucoup plus équilibrée que le rendu sort-first. Malheureusement, pour des systèmes d'affichage comportant beaucoup de pixels (comme les murs d'images), le trafic réseau engendré sature rapidement les systèmes à base de clusters de PC. Ice-T propose deux méthodes pour réduire ce trafic en optimisant les échanges d'information entre noeuds du cluster . Dans tout ce qui suit, on considère que l'on a un mur d'image composé de T sections, et N noeuds sur lequel le rendu est calculé, avec $N \geqslant T$. Puisqu'il s'agit d'un rendu sort-last, chaque noeud possède une image de même taille que la résolution du mur d'image, et rend une partie équivalente (en nombre de primitives géométriques) de la scène.

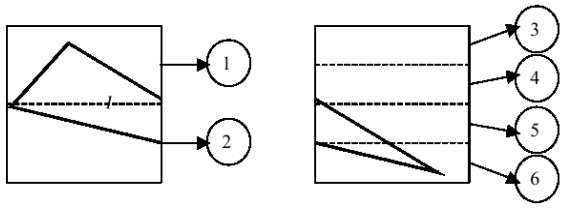
TILE SPLIT AND DELEGATE : chaque noeud est associé à une section. Chaque section est découpée de manière équilibrée en un nombre de parts égal au nombre de noeud associés. Chaque noeud associé est responsable d'une de ces parties, et uniquement ce noeud-là : une fois le rendu effectué, chaque noeud découpe l'image et envoie les parties aux noeuds responsables de ces dernières. Chaque noeud reçoit donc l'ensemble des parties qui lui permettent, en les composant, de posséder le rendu final de la partie de l'image dont il est responsable.

De plus, le matériel de rendu et de réseau étant le plus souvent séparés sur un noeud, les rendus, envois, et compositions se font en asynchrone pour maximiser les performances.

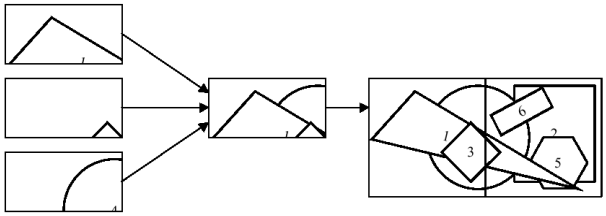


Les noeuds sont affectés aux sections de l'écran. Ici, les noeuds 1 et 2 sont affectés à la section 1, et les noeuds 3 à 6 sont affectés à la section 2. Puis les

sections sont découpées en parts égales et chaque partie est associée à un noeud.



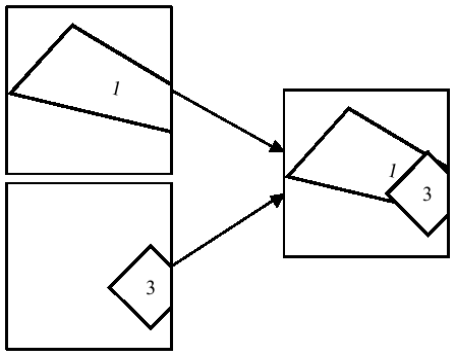
Le noeud 1 divise son image de rendu suivant le découpage précédent et envoie les parties aux noeuds responsables.

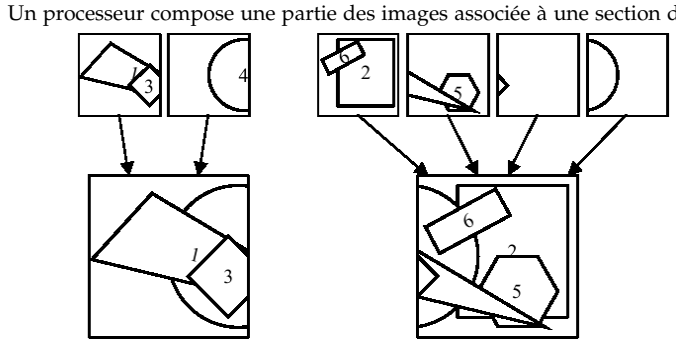


Le noeud 1 récupère la partie de l'image envoyée par les autres noeuds et les compose, puis renvoie la partie dont il est responsable pour l'affichage final.

Cette méthode donne de bons résultats, mais dans certains cas produit beaucoup de trafic réseau, puisque le nombre de messages à envoyer par rendu d'image est en $O(N^2)$.

REDUCE TO SINGLE TILE : comme la méthode précédente, chaque noeud est associé à une section, mais celle-ci n'est pas découpée. Après le rendu, chaque noeud possède donc une image finale comportant T sections. Il envoie alors les (T-1) parties auxquelles il n'est pas associé à leurs processeurs respectifs. L'envoi de ces sections d'image est telle que tous les noeuds associés à une même section reçoivent le même nombre d'images à composer. Une fois la composition effectuée, chaque noeud effectue l'algorithme *binary-swap*[McC98] avec les autres noeuds associés à la même section.





Chaque groupe de processeurs est indépendant des autres et contient l'ensemble des informations pour une section donnée. Chacun de ces groupe effectue une composition par binary-swap.

L'algorithme *binary-swap* consiste, lors de la composition de deux images, à n'envoyer, et donc réceptionner que la moitié de la zone à composer. Les deux noeuds composent alors la partie qu'ils ont reçue. Dans le cas d'une composition avec N images, les noeuds sont regroupés deux par deux, et on effectue $\log 2(N)$ étapes de composition (cf. figure 34).

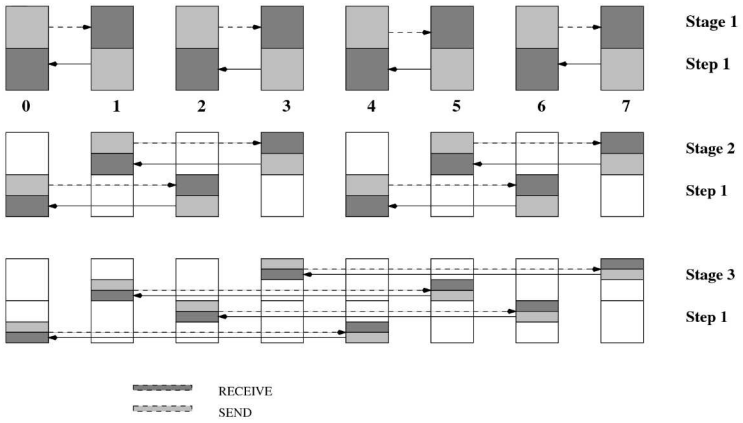


FIG. 34: Composition en *binary-swap* avec 8 images.

On obtient une répartition de charge presque aussi bonne que pour la première méthode. De plus, le nombre de messages est beaucoup moins important, puisqu'il est de l'ordre de $O(N * T + N\log N)$.

Optimisations supplémentaires d'Ice-T

Ice-T étant une solution sort-last, son goulet d'étranglement se situe principalement au niveau du débit réseau. Ses concepteurs ont donc intégré des optimisations pour essayer de réduire ce problème.

LE BUCKETING : il s'agit de découper la scène en larges régions, et de rapidement pré-trier lors du chargement des données les

éléments suivant leur appartenance à ces sous-espaces (*buckets*). Lors du rendu, un test sur l'emplacement de rendu des buckets est effectué pour savoir s'il doit être affiché ou pas. La version d'Ice-T intégrée à Paraview[Para] est complétée en amont du rendu par un système de Kd-Tree pour découper la scène.

Le Kd-Tree est détaillé en section 4.4

L'ENCODAGE DE PIXEL ACTIF : les différents noeuds de calcul ne rendent qu'une partie de la géométrie totale de la scène. L'image rendue est donc souvent composée de beaucoup d'espace vide, à savoir des pixels sans information de géométrie. Ice-T encode différemment les pixels "utiles" (actifs) et les pixels représentant la couleur de fond : la couleur de fond est envoyée une fois pour toute, et chaque pixel est marqué comme actif ou non. S'il est actif, son encodage est suivi des 3 composantes de couleur. Sinon, il n'est suivi d'aucune couleur.

LE FLOATING VIEWPORT : comme nous l'avons vu dans le point précédent, les images rendues par les différents noeuds de calcul comportent souvent beaucoup d'espace vide, donc sans information. Il est donc possible de réduire la taille des images envoyées sur le réseau, en envoyant seulement les parties où les éléments rendus ont été affichés. Si l'on prend l'exemple de la figure ci-dessous, en supposant que ce qui est représenté est la partie de la scène rendue par un unique noeud, si ce dernier utilise le floating viewport, il n'aura à envoyer que la partie de l'image délimitée par les pointillés, partie qui sera ensuite découpée en quatre pour être envoyée aux noeuds adéquats (cf. figure 35).

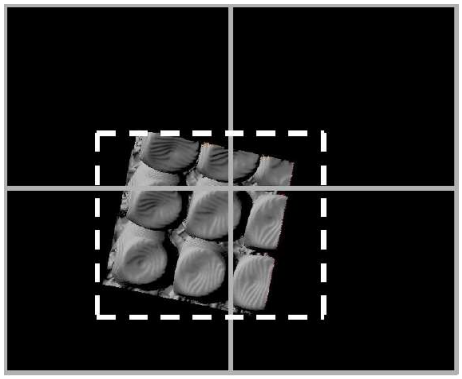


FIG. 35: Le Floating Viewport d'Ice-T

Les limitations d'Ice-T

Malgré les optimisations inclues dans Ice-T, qui rendent ce système largement plus optimal que Chromium[MT03], le goulet d'étranglement de cette solution reste le transfert des images entre les différents noeuds. Et cela est d'autant plus complexe qu'il se situe à deux niveaux : d'abord entre la carte graphique et la mémoire centrale, mais aussi au niveau du réseau.

Il est donc important de réduire à tout prix la taille des images, ou parties d'image, transférées lors du processus de rendu. Comme il existe déjà beaucoup de systèmes permettant de réduire la quantité d'information liée à l'image, comme l'encodage actif de pixels ou le floating viewport, il ne reste plus qu'une solution : réduire l'information en amont du processus de rendu, et l'organiser au mieux pour que le maximum de données utiles soient concentrées au final en un minimum de pixels.

Cela passe par :

- une réduction du nombre d'éléments qui seront affichés, par des procédés de *culling* (cf. section 4.4.1) par exemple. Ceci aura un double avantage : réduire le temps de rendu partiel des différents noeuds, et aura une influence sur la taille des images transmises.
- une répartition correcte de la charge de rendu, cette dernière allant forcément être perturbée par le *culling*. Il faut donc un système d'équilibrage des éléments à rendre par noeud.

Les deux premiers points font donc intervenir des éléments de l'architecture *sort-first*. Elles se situeraient en amont de la solution Ice-T.

On peut aussi imaginer d'autres points qui peuvent accélérer simplement le rendu. Ice-T ne prend par exemple pas en compte la cohérence qui peut exister entre deux images consécutives. Un système de cache serait donc intéressant pour réduire au maximum les informations envoyées à la carte graphique.

4.4 VISUALISATION ADAPTATIVE

Cette section regroupe des méthodes et techniques que l'on peut qualifier d'adaptatives car elles agissent à divers stades du processus de visualisation en tenant compte des caractéristiques des données en liaison avec les conditions d'affichage. La visualisation étant vue comme un filtre général entre les données et les images, on peut introduire des boucles de rétro-action partant de l'affichage pour adapter le filtre et l'optimiser. Ceci revient à opérer différents types d'adaptation sur les données transitant dans le pipeline de visualisation.

4.4.1 *L'élimination sélective*

Le principe

L'élimination sélective, ou *culling* est le procédé qui consiste à enlever le maximum d'objets pas ou peu visibles du pipeline graphique, et ce, le plus en amont possible.

Il en existe deux principaux[HV97] :

LE VFC (VIEW FRUSTUM CULLING) : élimination des éléments qui ne sont pas dans le champ de vision de la caméra. Tous les éléments visibles de la scène sont au moins en partie contenus dans le volume projeté, qui est le plus souvent un parallélépipède

Frustum, du latin frustum, "morceau", est une pyramide ou un cône tronqué par un plan. Il est à noter que pour le VFC nous ne tenons pas compte d'éventuelles surfaces de réflexion qui feraient apparaître des objets hors du frustum de vision.

rectangle (projection orthographique) ou une pyramide tronquée (projection en perspective) (cf. figure 36).

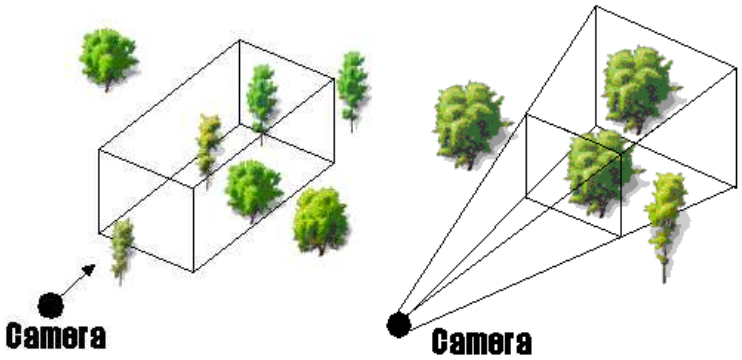


FIG. 36: Projections orthographique et en perspective

L'OCCUSION CULLING : élimination des éléments qui sont cachés par d'autres objets opaques de la scène. Ce processus est assez coûteux, puisqu'il nécessite un classement des objets dans l'ordre de profondeur croissante par rapport à la caméra, et un test de masquage entre les différents éléments.

Ces deux techniques sont dépendantes du point de vue : chaque mouvement de "caméra" exige de repasser dans la boucle d'adaptation.

Application à la visualisation particulaire

Les deux procédés ci-dessus s'appliquent bien évidemment à la visualisation atomistique.

Le VFC se base le plus souvent sur des arbres de découpage d'espace (cf. section suivante).

Les algorithmes d'*occlusion culling* sont assez nombreux, car en général spécifiques aux types de représentations des données. Dans le cas de la visualisation particulaire représentée par des sphères, l'avantage est que le test de masquage entre deux sphères est relativement simple, puisque ne dépendant que des centres et rayons des sphères. Un algorithme simple est de calculer les positions et rayons effectifs des deux sphères dans l'espace image, et de comparer la distance entre les deux centres et les rayons (cf. figure 37) :

$$\text{si } d(C1, C2) + R1 < R2 \text{ et } Z_C1 > Z_C2$$
$$\text{alors 2 masque 1}$$

On retrouve dans les logiciels existants des algorithmes plus performants : en particulier en utilisant des arbres (cf. section suivante). On peut aussi citer le principe d'occultation probabiliste dans Atomsviewer[Sha02] : prenons un exemple pour comprendre le principe. Supposons deux groupes de sphères A et B, de plus en plus éloignés de la caméra, et en partie alignés. En supposant que l'ensemble des sphères de A sont toutes complètement visibles, il

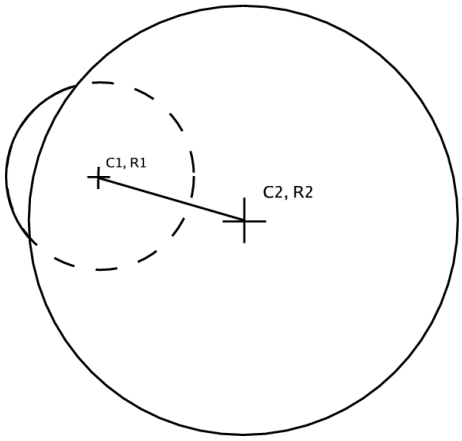


FIG. 37: *Occlusion culling* simple appliqué à deux sphères

existe une forte probabilité pour que les sphères de B soient cachées par celles de A. En calculant la densité de la région de A et la distance relative de A et B par rapport à la caméra, on peut obtenir une probabilité correcte du masquage de B. Dans la figure 38, le nombre de sphères a été réduit de 68% pour une détérioration minime de l'image.

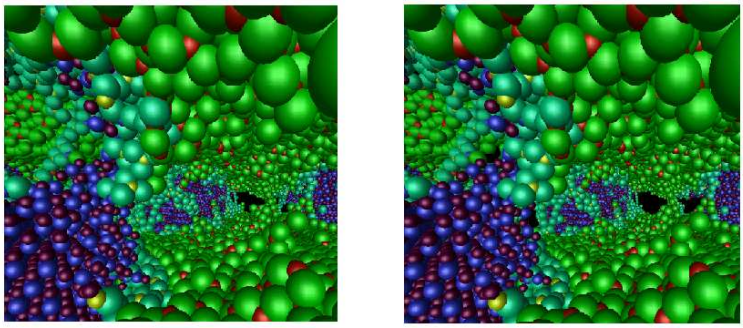


FIG. 38: Scène sans (à gauche) et avec (à droite) le *Probabilistic Occlusion Culling* d'Atomsviewer.

Il existe également dans la littérature une méthode[ZMHH97] appelée "Hierarchical Occlusion Maps" (HOM), remise au goût du jour depuis l'apparition de la programmation sur carte graphique. Le principe est le suivant : à chaque phase de rendu, chaque élément de la scène est pré-rendu indépendamment, avec la résolution souhaitée, et uniquement en blanc sur fond noir (pas de texture, ni d'effet de lumière, etc). A partir de cette image de base, qui consistue le masque de remplissage de l'élément dans le rendu final, on construit récursivement d'autres images, chacune ayant une résolution divisée par quatre par rapport à la précédente (divisée par deux pour chaque direction) (cf. figure 39).

Ceci permet d'effectuer un rendu très rapide de la scène, en basse résolution, par composition des masques des différents éléments, afin

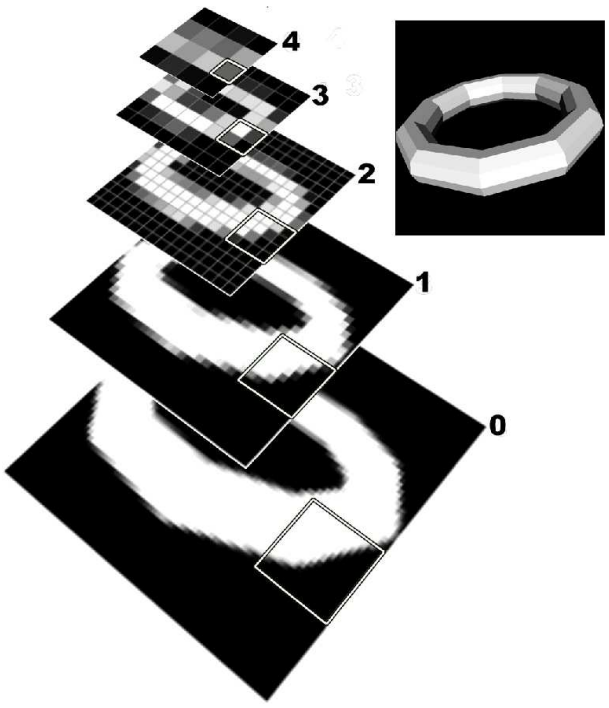


FIG. 39: Exemple de hierarchical occlusion map

d'éliminer les objets masqués du processus de rendu. S'il y a un doute sur l'occultation totale d'un élément sur un autre, un rendu partiel de l'image à une résolution plus importante est effectué.

4.4.2 *Niveau de détail*

Le Niveau de détail[LWC+02], ou LoD (Level of Detail) en visualisation est un principe qui se base sur le fait que des objets éloignés par rapport à la caméra sont nécessairement plus petits (en projection perspective). La résolution de l'écran étant fixe, la quantité d'informations amenée par la représentation de cet objet est donc seulement dépendante du nombre de pixels qu'il active, donc sa taille à l'écran. On peut alors simplifier la représentation des objets suivant leur distance à la caméra et/ou leur grosseur à l'écran : plus un objet est éloigné, plus on peut réduire le niveau de détails associés à sa représentation.

Nous n'abordons ici que l'application du LoD à la représentation de sphères.

Dans le cas d'une représentation polygonale

Dans le cas d'une représentation polygonale des sphères, le principe est de réduire le nombre de polygones en fonction de la distance à l'observateur et de la taille de l'objet.

Cette technique est par exemple utilisée dans Atomsviewer[Sha02]. Différents modèles de sphères, de 8 à 300 polygones, sont précalculés, puis suivant la distance de la sphère à la caméra, la résolution adéquate est choisie (cf. figure 40).

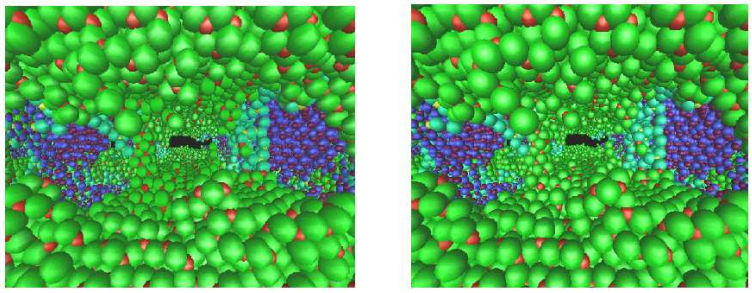


FIG. 40: LoD dans Atomsviewer : l'image de gauche utilise la même résolution pour toutes les sphères, l'image de droite utilise 8 résolutions différentes (de 8 à 300 polygones).

La réduction du nombre de facettes de cet exemple est d'environ 78%, pour une modification de l'image assez faible.

Dans le cas de l'utilisation de shaders

Rappelons ici qu'un shader est composé de deux parties, le Vertex shader, calculé pour un élément donné, et le Fragment shader, calculé pour chaque pixel qui compose l'image de l'élément.

La complexité de calcul de rendu de l'objet est de la forme :

$C = C_{Vertex} + N_{pixels} * C_{Fragment}$

avec N_{pixels} le nombre de pixels. On peut donc considérer que le LoD est intrinsèquement intégré à l'architecture du shader, puisque la complexité de calcul de ce dernier est fonction du nombre de pixels composant l'image de l'objet, à savoir la taille de l'objet à l'écran.

Dans le cas de la représentation d'une sphère, le Vertex shader ne calcule que la position du centre et le rayon en espace image. On peut donc considérer que cette partie du calcul du rendu est négligeable devant celle du Fragment shader, dès lors que le nombre de pixels associés à l'objet n'est pas trop faible (de l'ordre de 10, par exemple). On peut donc considérer que la complexité de calcul de rendu pour une sphère est presque proportionnelle au nombre de pixels à rendre.

4.4.3 *La hiérarchisation des données*

La hiérarchisation des données consiste à organiser, classer, et donner une échelle d'importance aux données manipulées suivant des critères spécifiques.

Généralités

Il existe bien évidemment énormément de types de hiérarchies de données, même en visualisation particulaire. On peut citer pour

exemple le regroupement des atomes par structures composant une protéine, en visualisation moléculaire. Cela permet ainsi de représenter ces structures de façon particulière, afin d'alléger la visualisation et faciliter l'étude fonctionnelle de la protéine (sites actifs, etc. cf. figure 16 et la section 2.4.1).

Cette étude se limitant uniquement à la représentation de sphères, nous n'allons présenter que la principale forme de hiérarchisation appliquée à ce domaine.

Les arbres de partitionnement d'espace

Les arbres de partitionnement d'espace sont des méthodes très générales de hiérarchisation des données.

Le principe est le suivant : les données sont stockées dans les feuilles d'une structure en arbre. Chaque noeud représente un sous-espace, et chaque fils de ce noeud est soit une feuille (des données), soit une subdivision de l'espace représenté par le père. On peut citer les principaux[HV97] :

LE BSP TREE : Le BSP (Binary Space Partitioning) tree est l'arbre le plus générique. Il s'agit d'un arbre binaire, où chaque noeud contient un hyperplan qui divise l'espace en deux. Dans le cas d'un espace 3D, l'hyperplan est donc un plan. Cette subdivision n'est pas forcément parallèle aux axes (cf. figure 41).

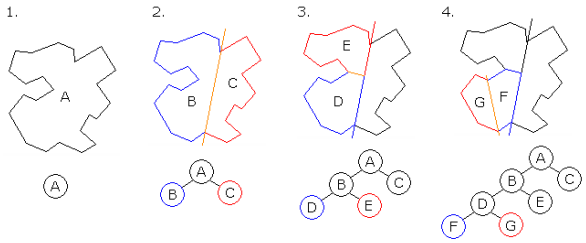


FIG. 41: Un BSP tree en dimension 2.

LE KD-TREE : Il s'agit d'un BSP tree dont chaque plan de séparation est parallèle à deux axes du repère (ce qui n'est pas forcément le cas pour un BSP). Le plus souvent, il y a alternance du parallélisme des plans (par exemple, le premier plan est normal à l'axe des x, ses fils à l'axe des y, ses petit-fils à l'axe des z, etc. cf. figure 42).

L'OCTREE : l'octree est une structure de données de type arbre dans laquelle chaque noeud peut compter jusqu'à huit enfants. Les octrees sont le plus souvent utilisés pour partitionner un espace tridimensionnel en le subdivisant récursivement en huit octants. La plupart du temps, les plans de coupe des octants sont parallèles aux axes du repère.

- Dans le cas d'un octree de type "point region" (PR), le noeud mémorise explicitement un point tridimensionnel qui est le

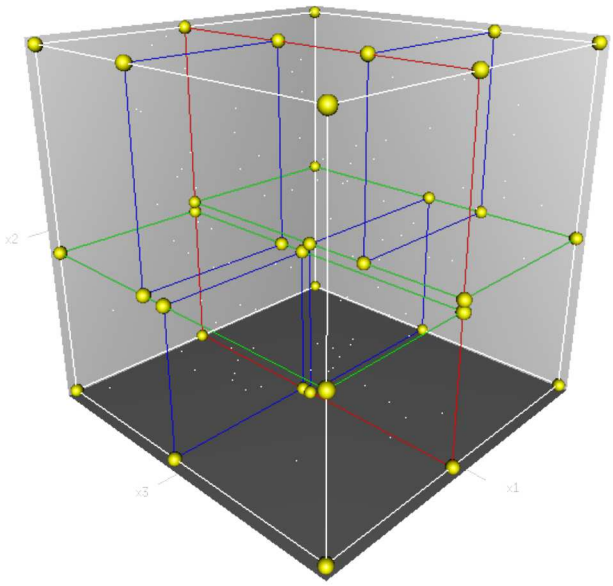


FIG. 42: Un exemple de Kd-tree : l'ordre des plans dans l'arbre est rouge, vert, puis bleu.

"centre" de la subdivision pour ce noeud, le point défini alors l'un des coins de chacun des huit enfants. Le noeud racine d'un octree de type PR peut représenter un espace infini.

– Dans un octree de type "MX" (MatriX), le point de subdivision est implicitement le centre de l'espace que le noeud représente. Le noeud racine d'un octree de type MX doit représenter un espace fini de manière à ce que les centres implicites des noeud soient bien définis.

Exemples d'utilisation des arbres de partitionnement d'espace

La principale utilisation de ces arbres en visualisation est l'accélération du VFC (view frustum culling). Le principe est le même pour tous les arbres : on parcourt l'arbre en vérifiant si chacun des noeuds fils est visible ou non. Si oui, on fait de même pour ses fils, jusqu'à arriver aux feuilles (les données). Sinon, les fils sont ignorés. En équilibrant l'arbre, c'est-à-dire en plaçant les sections de telle sorte qu'il y ait la même quantité de données dans les fils, on peut alors obtenir une réduction intéressante du nombre d'objets qui, au final, ne seront pas visibles.

Le choix de l'arbre est très lié au type de données manipulées : dynamiques, ou statiques (mortes). Dans le cas de données mortes, il est possible d'effectuer des calculs lourds permettant des optimisations puissantes des arbres, et des constructions d'arbres profonds (non nécessité de temps réel). On peut alors utiliser des arbres avec des géométries non régulières, et placer efficacement les sections pour équilibrer l'arbre. Dans le cas de données générées dynamiquement, il est plus difficile de construire de tels arbres. La plupart du temps, la

simulation en amont possède déjà une structure en cellules, qui sont ensuite réutilisées pour construire l'arbre.

On peut prendre pour exemple l'octree utilisé dans le solution Atomsviewer : chaque noeud découpe l'espace en 8 octants de même volume. Les feuilles de l'arbre sont reliées à une cellule (une subdibvision spatiale) de la simulation (cf. figure 43).

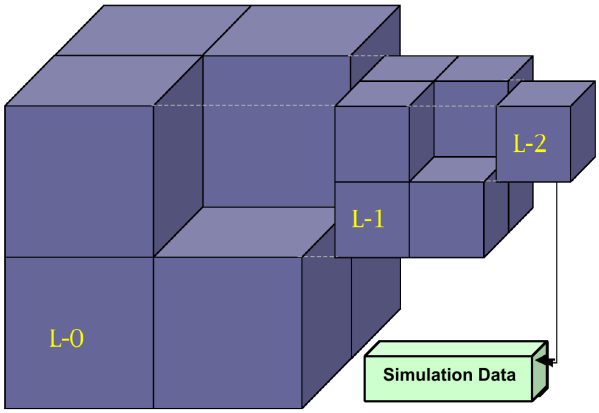


FIG. 43: L'octree d'Atomsviewer

L'octree est alors utilisé pour le VFC de la manière suivante :

- pour la profondeur de champ : chaque cellule est apparentée à la sphère englobante de celle-ci (la sphère contient les sommets de la cellule). Ensuite, on considère que l'ensemble des cellules visibles pour une profondeur de champ donnée p sont incluses dans une sphère de centre à distance $p/2$ du point de vue, et de rayon $p/2$. Il est alors simple de parcourir l'octree pour savoir quelles cellules (sphères) sont incluses dans la grande.
- pour le frustum de visualisation : une fois la liste des cellules visibles partiellement réduite, on intersecte avec le cône de visualisation. On réduit alors la liste au strict minimum des cellules visibles (cf. figure 44).

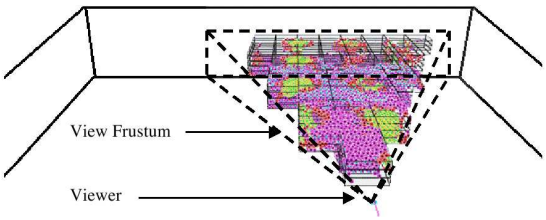


FIG. 44: Le VFC dans Atomsviewer

On peut également utiliser ces arbres pour réduire la complexité de l'*occlusion culling* : lorsque les données sont représentées par des sphères, comme en visualisation atomistique, il est plus simple de d'abord trier les cellules par ordre de profondeur par rapport à la caméra pour réduire au maximum le nombre de tests superflus. Ainsi

les tests de masquage des sphères ne se feront qu'entre sphères d'une même cellule, ou des cellules plus éloignées.

Enfin une utilisation possible est la simplification de la scène pour garder un rafraîchissement minimum. En effet, les arbres peuvent découper l'espace pour que les données soient uniformément réparties suivant les branches, celles-ci contenant ainsi la même quantité d'information. Lors du rendu de la scène, si le nombre d'objets total est trop important pour garder le rafraîchissement minimum voulu, on peut volontairement tronquer des parties de l'arbre qui sont peu visibles (ce qui peut par exemple être déterminé assez facilement en fonction de la profondeur du noeud et la profondeur du centre de ce noeud). Bien que cela soit déconseillé dans certains cas (impossibilité de connaître la validité de la visualisation image par image), cette fonctionnalité peut néanmoins donner des résultats intéressants, en particulier pendant l'exploration initiale (perception de la profondeur par l'animation, de phénomènes globaux etc).

4.5 UN BREF APERÇU DES SOLUTIONS EXISTANTES

Cette section présente une liste non exhaustive des solutions permettant la visualisation atomistique. Nous essayons néanmoins de recenser les plus représentatives de ce que l'on peut trouver dans la littérature.

Cette liste est organisée suivant les caractères suivants :

- cette solution est-elle très spécifique à un domaine (orientée "métier") ?
- est-elle adaptée à un système de visualisation parallèle ?

4.5.1 *Non parallèle et spécifique*

RASMOL[SMW95] ET MOLSCRIPT[KRA91] : Il s'agit de deux des premiers logiciels de visualisation moléculaire complets et gratuits (1991-1993). RasMol est l'un des premiers logiciels de visualisation atomistique à être interactif, ou tout du moins à permettre des animations (pour percevoir la profondeur).

XCRYSDEN[KOK03] : Logiciel spécialisé dans la visualisation de structures cristallines. Il sert d'interface graphique pour un certain nombre de simulations ab initio.

AVIZ[WH00] : Logiciel spécialisé dans la visualisation atomistique. Son grand avantage est sa légèreté et sa simplicité de modification (et donc d'adaptation à des besoins spécifiques).

VMD[HDS96] : Logiciel spécialisé dans la visualisation moléculaire. Gratuit, "scriptable" (pilotable par langage de programmation), modulaire, il possède de nombreuses fonctionnalités spécifiques à la représentation de molécules, ce qui en fait une des solutions les plus connues et les plus utilisées en biologie moléculaire.

CRYSTALMAKER[CRY] : Logiciel propriétaire de visualisation de structures cristallines. Considérée comme une des références en

matière de graphismes dans ce domaine, il reste un logiciel payant et fermé.

TEXMOL[BDST04] : Un des premiers logiciels de visualisation atomistique à utiliser la programmation de carte graphique pour accélérer le rendu.

MOLEKEL[MOL] : Logiciel opensource (sous licence GPL), développé par le CSCS, et spécialisé dans la visualisation de molécules. Il est basé en partie sur VTK[VTK]. C'est l'un des premiers logiciels à utiliser la programmation GPU pour accélérer le rendu en visualisation moléculaire.

4.6 PARALLÈLE ET SPÉCIFIQUE

De telles solutions son souvent créées par les laboratoires spécifiquement pour leur matériel et leurs besoins. Il est quand même intéressant d'en citer au moins deux :

ATOMEYE[LI05] : Solution spécialisée dans la visualisation atomistique pure. Légère et optimisée pour des machines de type cluster de PC, elle n'utilise pas l'architecture OpenGL pour le rendu : son système se base sur des routines uniquement CPU optimisées pour l'affichage de sphères et de cylindres.

ATOMSVIEWER[SHA02] : Solution spécialisée dans la visualisation atomistique interactive de très gros jeux de données. Basée sur une architecture parallèle spécifique, elle introduit un certain nombre d'optimisations très intéressantes, qui ont été citées en sections 4.4 . Son architecture est décrite par la figure 45.

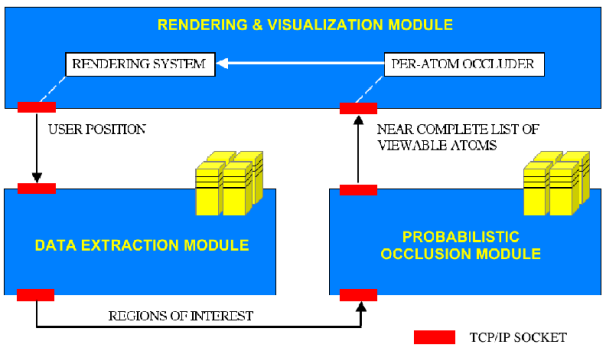


FIG. 45: L'architecture d'Atomsviewer

Le module d'extraction envoie les données contenues dans la région d'intérêt (le champ de vision de la caméra) au module d'occlusion probabiliste. Ce dernier réduit encore la liste des éléments à afficher en enlevant les éléments peu visibles, ou qui ont une faible chance de l'être (cf. section 4.4.1), et envoie cette liste au module de rendu. Ce système permet d'enlever énormément d'informations superflues, et donc de grandement accélérer le rendu des données.

4.6.1 *Les extensions spécifiques des solutions génériques*

Les solutions payantes

AVS : AVS[AVS] est solution très généraliste de visualisation, de monitoring, et de présentation de résultats scientifiques ou industriels. Un module "molecular" existe, ainsi qu'une version parallélisée, utilisant la technologie MPI (mpich).

AVIZO : Avizo[Avi], anciennement Amira[Amia] est un logiciel de visualisation 3D générique, spécialisé notamment dans le rendu volumique. Il possède néanmoins une extension pour la visualisation moléculaire, AmiraMol[Amib]. Amira possède également une version cluster (rendu parallèle de type sort-first), permettant un affichage sur de grands écrans de type mur d'images, ou des environnement de réalité virtuelle, tels qu'un "Holobench" ou un "CAVE", ainsi que des périphériques de tracking 3D.

VTK et Paraview

VTK[VTK], *Visualization Toolkit*, est une bibliothèque opensource et multiplateforme destinée à la visualisation scientifique. Ecrite en C++, elle possède aussi des interfaces vers des langages interprétés, comme Tcl/Tk, Python, ou Java.

VTK ne propose pas directement de solution pour la visualisation atomistique, mais sa généricité permet néanmoins d'afficher des sphères, des points, ainsi que d'autres figures géométriques utiles à la visualisation moléculaire (cylindres, etc). La manipulation des shaders a récemment été intégrée dans la branche principale, mais il n'y pas accès à l'intégralité du potentiel des cartes graphiques (pas de VBO en particulier).

Enfin, VTK possède un mécanisme général de parallélisme implanté dans les couches objets, ainsi qu'un certain nombre de classes de services spécifiques additionnels. Le type de parallélisation par défaut est le *sort-last*, généralement utilisé pour des jeux de données volumineux, mais les classes sont assez génériques pour pouvoir être utilisées avec d'autres algorithmes.

Paraview[Para] est un logiciel de visualisation scientifique dont le coeur est basé sur VTK. En plus d'ajouter, entre autre, une interface graphique, Paraview possède des fonctionnalités avancées en visualisation parallèle.

Paraview utilise une parallélisation de type sort-last[Parb], y compris pour l'affichage sur un mur d'images. En effet, les classes implémentées à cet effet sont basées sur la bibliothèque IceT[MT03].

4.6.2 *Tableau récapitulatif*

Les tableaux suivants regroupent les informations importantes des différentes solutions citées :

TYPE : donne le type de solution : il peut s'agit d'une bibliothèque, d'un framework, ou d'un logiciel complet

VISUALISATION PARALLÈLE : indique si la solution peut être utilisée sur une architecture parallèle, et quel type de parallélisme il utilise.

INTERFACES : donne les bibliothèque et language de script utilisables pour le contrôle d'application de la solution

EXTENSIONS : spécifie, si elles existent, les extensions de la solution permettant son utilisation dans un environnement immersif : RV (réalité virtuelle), utilisation d'un mur d'images

SORTIES : donne les sorties graphiques possibles : Soft (sans accélération graphique), OpenGL, images (captures d'écran), vidéo, shaders (programmation de cartes graphiques)

MODULARITÉ : indique la facilité d'intégrer à la solution de nouvelles fonctionnalités

	RasMol	XCrysDen	Aviz	VMD	Crystalmaker	TexMol
Type	logiciel	logiciel	logiciel	logiciel	logiciel	logiciel
License	open-source	GPL	GPL	open-source	propriétaire	open-source
Visualisation parallèle	non	non	non	non	non	non
Interfaces	native	tcl-tk	Qt	tcl-tk python	native	Qt
Extensions	.	.	.	RV	.	.
Sorties	Soft images	OpenGL images video	OpenGL images	OpenGL images video	Soft images video	OpenGL shaders
Installation/Prise en main	très facile	facile	facile	facile	facile	facile
Modularité	- -	- -	- -	+ +	- -	- -
Systèmes d'exploitation	Win,Mac,Linux	Win,Mac,Linux	Linux	Win,Mac,Linux	Win,Mac	Win,Linux
Orientation métier	moléculaire	cristallographie	moléculaire	moléculaire	cristallographie	cristallographie

	Atomeye3	Atomsviewer	AVS	AmiraMol	VTK/Paraview
Type	logiciel	toolkit	logiciel	logiciel	framework/logiciel
License	open-source	open-source	propriétaire	propriétaire	open-source
Visualisation parallèle	sort-first	sort-first	.	.	sort-last
Interfaces	raccourcis	.	.	.	C++,java python,qt
Extensions	mur d'images	mur d'images	RV mur d'images	RV mur d'images	RV mur d'images
Sorties	Soft images	OpenGL	OpenGL images video	OpenGL images video	OpenGL shaders images video
Installation/Prise en main	facile	difficile	facile	facile	difficile/facile
Modularité	- -	- -	+ +	+ +	+ +
Systèmes d'exploitation	Win,Mac,Linux	Win,Mac	Win,Mac,Linux	Win,Mac,Linux	Win,Mac,Linux
Orientation métier	atomistique	atomistique	visu sci	visu sci	visu sci

5 CONCLUSION ET PROPOSITIONS

Cette première partie a permis de faire apparaître deux besoins forts dans le domaine de la visualisation scientifique appliquée à la physique des matériaux :

- un besoin en **interactivité**, qui passe par la conception d'une solution de rendu efficace pour les scènes composées d'un grand nombre de particules,
- un besoin en **interaction**, en particulier une solution de contrôle d'application permettant la saisie d'assez de paramètres pour explorer de telles scènes, mais aussi capable de s'intégrer à un environnement immersif simple, comme un mur d'images et un système de pointeur 3D.

Or aucune des solutions de visualisation scientifique, spécifiques ou non, n'apporte de solution véritablement satisfaisante à l'un ou l'autre de ces besoins.

Le but de cette étude étant de fournir une solution complète à ce problème, il a donc fallu étudier ces deux points principaux. De plus, les deux concepts sont intrinsèquement liés lors de la phase d'exploration de la scène :

- l'interactivité apporte énormément d'information en terme de perception lors de l'exploration,
- une interaction immersive adaptée facilite grandement la phase exploratoire,
- sans interactivité, l'interaction immersive perd grandement de son utilité.

Chacune des deux parties suivantes propose donc d'étudier une des problématiques précitées.

La partie sur l'interaction est une étude sur l'adaptation d'un système de contrôle d'application intuitif et efficace, le *FlowMenu*, à un environnement immersif simple.

La partie sur l'interactivité propose un choix d'architecture adaptée au rendu de scènes à grands nombre de particules, ainsi que des optimisations permettant d'augmenter le taux de rafraîchissement sans altérer la qualité des images.

Deuxième partie

CONTRÔLE D'APPLICATION D'UNE SIMULATION EN ENVIRONNEMENT IMMERSIF

Cette partie tente de répondre à un des principaux problèmes d'interactivité de toute solution de visualisation scientifique : trouver un système de contrôle d'application adapté à l'exploration de la scène représentant une simulation.

Cette réponse est, bien sûr, restreinte aux problématiques de visualisation de simulations de Physique des Matériaux d'une part, ainsi qu'à un environnement immersif simple d'autre part.

Le chapitre 3 de la première partie a permis de dresser un rapide portrait du contrôle d'application en environnement immersif, ainsi que des solutions que l'on retrouve dans les systèmes de visualisation scientifique.

Nous présenterons ici un système de menu particulier, le *FlowMenu*, ainsi que les choix qui ont amené à étudier plus en détail ce menu plutôt qu'un autre. Nous développerons ensuite les travaux que nous avons effectués pour tenter d'adapter ce menu à un environnement immersif, et enfin nous étudierons les résultats que nous avons obtenus sur l'efficacité d'une telle solution.

6 LE FLOWMENU : UNE SÉLECTION DE CHOIX

Ce chapitre présente le *FlowMenu*, un système de menu 2D particulier, ainsi que les raisons qui ont conduit à se tourner vers une telle solution dans notre étude sur la conception de contrôle d'application en environnement immersif.

6.1 CADRE DE L'ÉTUDE SUR LE CONTRÔLE D'APPLICATION EN ENVIRONNEMENT IMMERSIF

La problématique liée au choix du contrôle d'application est un sujet extrêmement vaste, et il n'est pas question de l'aborder complètement dans cette étude.

Nous avons donc choisi de cibler notre étude en réduisant le dispositif matériel nécessaire, le contrôle d'application dépendant fortement de ce qui est disponible. Nous proposons de restreindre le dispositif nécessaire à :

- Un système d'affichage de grande surface et/ou résolution (mur d'images, grand écran de bureau, etc)
- Un système de capture de position et/ou d'orientation (un *tracker*) avec un bouton. Cela peut être une simple souris de bureau, mais cet outil n'étant pas très adapté dans un environnement 3D, on peut envisager un système magnétique (comme le Fastrack), vidéo (avec reconnaissance de gestes), ou infra-rouge, par exemple.

Une telle configuration est a priori le minimum que l'on peut qualifier d'immersif. On peut même envisager une compatibilité avec un environnement 2D, en remplaçant le dispositif de capture par une souris, par exemple.

6.2 MENU 2D VS MENU 3D

La question de savoir si on doit perdre un ou plusieurs degrés de liberté au profit de la lisibilité et de la simplicité est primordiale. Comme nous l'avons vu plus haut, l'utilisation sans modification d'un menu 2D dans un environnement 3D crée des contraintes et force à faire des choix (positionnement, sélection) pour qu'il devienne utilisable.

Cette section décrit les éléments qui nous ont amené à préférer un menu 2D intégré dans un environnement 3D (le *FlowMenu* en particulier), au détriment de menu directement 3D.

6.2.1 *La lecture*

Un système de contrôle d'application ayant pour but d'envoyer des commandes, il est bien sûr nécessaire que l'utilisateur puisse comprendre le plus rapidement et le plus facilement possible ce que provoque l'envoi de celles-ci.

Bien que le système d'icônes du WIMP soit souvent utilisé comme représentation d'une tâche ou d'une fonction, sans un "dictionnaire" faisant le lien entre une icône et l'explication de la commande, l'utilisateur novice est perdu. C'est d'ailleurs pour cela que la plupart des logiciels de gestion de bureau virtuels, les *window managers* (Gnome, Kde, Windows par exemple) associent toujours du texte à leurs icônes.

L'intuitivité des menus, 2D ou 3D, passe donc beaucoup par la lecture purement textuelle, seul mode de communication visuel capable de véhiculer des informations assez précises et compréhensibles pour que l'utilisateur sache ce que les différentes commandes effectuent comme opérations.

Or, la lecture est essentiellement une tâche 1D. De plus, l'apprentissage de la lecture se fait sur un environnement 2D, comme un livre ou un tableau, par exemple. La lecture de texte en 3D, c'est-à-dire avec un texte dont la direction n'est pas parfaitement parallèle aux axes horizontaux (resp. verticaux pour les écritures) par rapport au regard de l'utilisateur, est donc a priori très peu intuitive. Cela milite en faveur d'un menu 2D perpendiculaire à la direction de vision (tout effet de perspective appliqué au texte, comme par exemple la modification de la taille des caractères ou un texte "penché", rend la lecture très inconfortable).

Des tests effectués par Chen, Pyla et Bowman[CPB04] sur l'insertion d'informations textuelles dans un environnement virtuel tranchent clairement : le HUD (*heads-up display*) est plus confortable pour l'utilisateur que le WWD (*within-the-world display*).

6.2.2 *Simplicité vs fonctionnalités*

Si l'on admet qu'une grande surface d'affichage est déjà un environnement immersif, on a tout intérêt à rechercher un système de contrôle d'application simple.

Les exemples de contrôle d'application 3D cités plus haut, comme le C^3 ou l'interface TULIP, semblent obtenir de très bons résultats en matière de précision et de rapidité de sélection de commandes. Néanmoins, la plupart de ces systèmes nécessite un matériel soit coûteux, soit encombrant pour l'utilisateur, et le plus souvent peu commun. Outre les raisons financières, il ne faut pas oublier pour certains sites des questions liées à la sécurité (interdiction du wifi, etc.), voire des questions de crédibilité face à des systèmes qui peuvent paraîtrent quelquefois des "gadgets" aux yeux de certains néophytes sceptiques.

Plaçons-nous dans cette configuration réduisant le matériel nécessaire au strict minimum pour une manipulation 3D immersive : un

écran de grande dimension (sans forcément avoir de stéreoscopie), et un outil facilement manipulable permettant de donner la position (un tracker, donc), avec un bouton (le Fastrack est un bon exemple, mais il peut s'agir d'un PHANToM, un Spidar, etc.).

En réduisant ainsi le matériel nécessaire, il devient relativement difficile d'utiliser la plupart des systèmes purement 3D pris en exemple. Se touner vers des systèmes de menus/widgets ne fonctionnant qu'avec un seul bouton, mais offrant quand même un maximum de possibilités de types de commande (sélection classique, sélection continue de type "défilement", grand nombre d'éléments etc.) est alors inévitable.

6.3 LE FLOWMENU

Cette section présente le FLowMenu tel qu'il est décrit par Guimbretière[GWoo], c'est-à-dire en environnement 2D. On verra qu'il comporte la plupart des fonctionnalités dont disposent les autres menus 2D.

6.3.1 *Présentation*

Le FLowMenu est un PieMenu à 8 secteurs, la partie intérieure de l'octogone étant creuse. De la même manière que son prédécesseur, il vient se placer autour du curseur lorsqu'un événement déclenche son apparition (click souris, par exemple).

La sélection des commandes se fait en déplaçant le curseur sur l'un des quartiers, puis en revenant au centre (cf. figure 46).

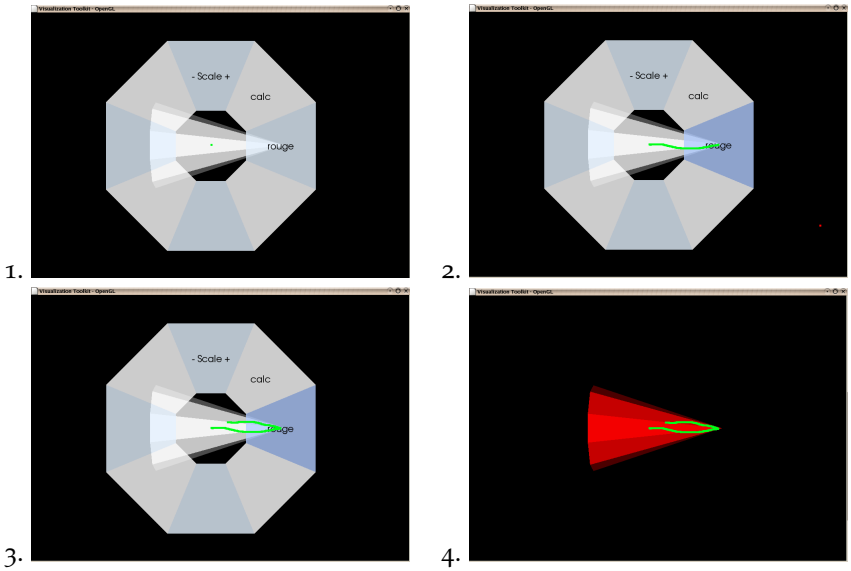


FIG. 46: La commande dans un FLowMenu : elle est sélectionnée par le passage du curseur (étape 2), puis validée lors du retour du curseur vers le centre (étape 3).

6.3.2 *Le Quikwriting*

Le Quikwriting[Per98] est un système de sélection de caractères permettant l'écriture de texte de façon continue. Il se retrouve souvent sur les organiseurs électroniques, car il permet une écriture assez rapide et sans lever le stylet.

Le principe est le suivant : l'espace est découpé en 8 zones, chacune correspondant à plusieurs choix de caractères possibles. La sélection s'effectue en déplaçant le curseur vers le caractère souhaité, puis en revenant au centre, mais elle dépend des interfaces entre zones qui ont été coupées par le curseur.

Par exemple, la figure 47 montre l'écriture du mot "the" :

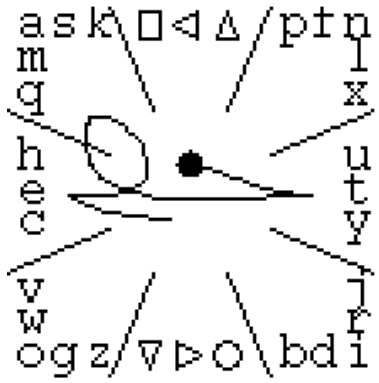


FIG. 47: Exemple de Quikwriting

- le caractère "t" est sélectionné en rentrant et sortant de la zone est sans couper de ligne
- le caractère "h" est sélectionné en rentrant dans la zone ouest, en coupant l'interface avec la zone nord-ouest, puis en sortant par cette dernière
- le caractère "e" est sélectionné de la même manière que "t", mais avec la zone ouest.

C'est ce principe qui est aussi utilisé dans le FlowMenu : l'écriture de caractères est une des fonctions possibles, mais en général il s'agit de sélections de commandes plus ou moins complexes (cf. figure 48).

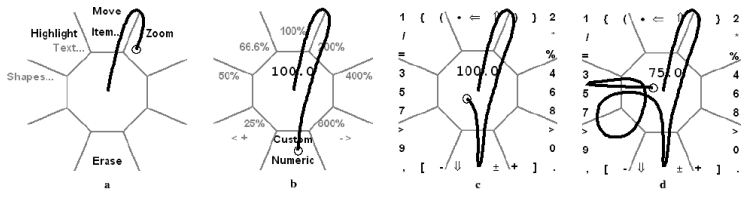


FIG. 48: Un exemple de sélection de commande du FlowMenu

6.3.3 *Fonctions avancées*

La Scrollbar

Une extension fort intéressante au principe de Quikwriting intégré au FlowMenu est le principe de "scrollbar", ou de barre de défilement (cf. figure 49).

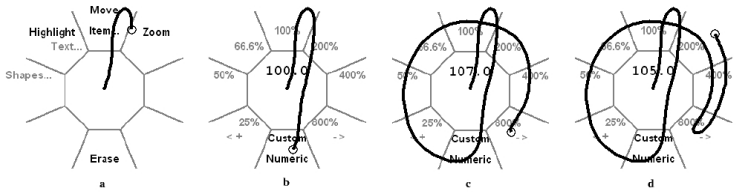


FIG. 49: La scrollbar du FlowMenu

Comme le montre l'image ci-dessus, une fois la fonction nécessitant une scrollbar sélectionnée (un zoom ici), c'est le fait de tourner autour du menu dans le sens direct ou indirect qui modifie la valeur associée à cette barre de défilement. L'arrêt de cette fonction s'effectue, comme toutes les autres, en ramenant le curseur au centre du menu.

Les gestures

Les *gestures* réalisées par le curseur ne représentent véritablement aucune figure graphique. il est pourtant assez facile de les mémoriser, tout comme celles des marking menus[KB93].

C'est pourquoi le FLowMenu admet un mode "expert", dans lequel le menu ne s'affiche, au bout d'un laps de temps court, que si le curseur reste immobile pendant ce moment. Un utilisateur expérimenté ne laissera alors plus s'afficher le menu, connaissant déjà les "gestures" à effectuer pour lancer la commande souhaitée.

Les sous-menus

Tout comme l'association avec le *Quikwriting* utilise l'activation de plusieurs quartiers séquentiellement pour l'écriture de caractères, on peut aussi utiliser cette multi-activation pour l'utilisation de sous-menus : Le premier quartier sélectionné fait alors apparaître dans les autres parties du FlowMenu les options du sous-menu. La sélection et la validation se font alors respectivement en déplaçant le curseur dans la section correspondante, et en revenant au centre du menu.

6.4 CONCLUSION

Ci-dessous sont résumés les points forts du FlowMenu pour une utilisation par des utilisateurs expérimentés ou non en environnement immersif :

– Le FlowMenu possède une interface assez simple (seulement 8 secteurs), avec la possibilité de décrire les fonctions de manière

explicite (en utilisant des textes), mais avec un nombre de possibilités de sélection intéressant (entre 24 et 28 possibilités avec un quikwriting complet), ainsi que l'intégration d'une barre de défilement. Il s'agit donc d'un menu complet mais pas trop chargé en informations.

- Selon Callahan[CHWS88] et Komerska[KW04], la sélection radiale est plus adaptée qu'une sélection linéaire (d'un menu linéaire classique, par exemple), que ce soit dans un environnement 2D ou 3D. Le FLowMenu fait donc partie des menus 2D les plus enclins à être adaptés en 3D.
- Le principe de validation de commande par retour au centre permet une utilisation simple du FlowMenu même avec un seul bouton. De plus, la manipulation/modification d'objets virtuels devient plus simple : le bouton peut être utilisé pour faire apparaître/disparaître le menu, tandis que la position initiale du curseur peut par exemple sélectionner un objet ; le FlowMenu permet alors de sélectionner une commande, pour enfin revenir à la manipulation de l'objet. On a donc un mélange de contrôle d'application et de manipulation directe (cf. figure 50).
- Le FlowMenu a été au départ conçu pour fonctionner sur de grands systèmes d'affichage, avec un périphérique de type stylo, mais des tests ont montré qu'il était aussi efficace dans un environnement plus classique (souris).

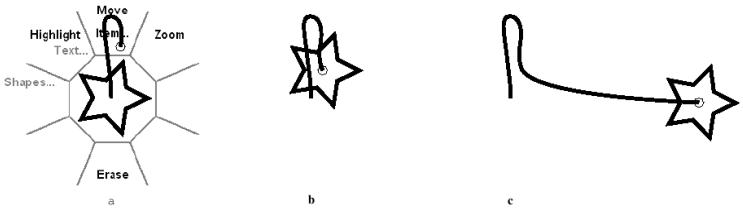


FIG. 50: Sélection d'une commande et manipulation directe d'un objet

- Enfin, le mode "expert" qui passe par l'utilisation de gestures permet un apprentissage du menu afin d'augmenter la vitesse de sélection des commandes.

Rappelons que le FlowMenu est un menu 2D, et que tel quel, il n'est pas utilisable efficacement dans un environnement immersif. Le chapitre suivant présente l'étude préliminaire de son intégration à un tel environnement.

7

INTÉGRATION EN ENVIRONNEMENT IMMERSIF : LE *FLOWMENU3D*

Ce chapitre décrit les choix possibles pour adapter le *FlowMenu* à un environnement immersif. Comme nous l'avons vu au chapitre 3 de la première partie, il existe plusieurs possibilités d'intégration.

Il existe également des problématiques de taille, de forme, d'utilisation de la couleur, entre autre.

7.1 LE POSITIONNEMENT

Comme cela a été décrit dans le chapitre 3, le problème du positionnement est critique. Si l'on souhaite garder les fonctions de sélection/manipulation directe de l'objet, le centre du menu doit automatiquement se positionner autour du tracker, ou tout du moins sur la droite parallèle à la direction de vision et passant par le tracker. De plus, le FLowMenu devant être lisible facilement, il doit être normal à la direction de vision. Cela ne laisse donc que peu de choix pour son positionnement.

7.1.1 *Positionnement centré tracker, dirigé tête*

Le FlowMenu est centré sur le tracker, parallèle au plan de projection (donc face à la direction de vision de l'utilisateur). Si la distance entre le plan de projection et le tracker vient à changer, le FlowMenu se déplace en conséquence sur la droite perpendiculaire aux plans.

Remarques :

- Si le tracker est muni d'un système de retour de force, il serait sans doute judicieux de contraindre le déplacement du tracker au seul plan contenant le FlowMenu, afin que la sélection des commandes soit plus facile et plus rapide[KW04].
- Il peut y avoir des problèmes d'occultation du menu par les objets virtuels présents entre ce dernier et l'observateur. Néanmois, cela peut être résolu facilement en ajoutant un plan de coupe (*clipping plane*) de la scène, confondu avec le plan du menu, qui s'active au moment de l'utilisation du menu.

7.1.2 *Positionnement centré et dirigé tracker*

On postitionne le menu de la même façon que la méthode ci-dessus, mais le FlowMenu est perpendiculaire au tracker, plutôt qu'à la direction de vision.

Comme nous le verrons dans le chapitre suivant, ce type de positionnement devient peu lisible dès qu'il est oblique, et n'est pas du tout intuitif.

7.1.3 *Positionnement centré tête, dirigé tracker*

Le FlowMenu est alors comme "plaqué" sur l'écran : la position sur la perpendiculaire au plan de projection est fixe.

Remarques :

- Cela peut être assez dérangeant comme positionnement s'il y a la stéréo car le menu est virtellement proche de l'oeil de l'utilisateur.
- Il est probable qu'il faille rajouter un pointeur 2D qui montre la position relative du tracker sur le menu afin que l'utilisateur sache ce qu'il est en train de sélectionner.

7.2 TAILLE, CURSEUR, ET FORME

En plus du positionnement, il nous faut savoir travailler sur d'autres caractéristiques du FlowMenu : sa taille, la sensibilité du curseur, et sa forme.

7.2.1 *Taille du menu : Encombrement vs Ergonomie*

Le principal inconvénient a priori d'un menu 2D dans un environnement 3D est son intrusion dans la scène : lorsqu'il est visible, il couvre une partie de l'image que perçoit l'observateur.

Si l'on utilise le tracker 3D comme curseur dans le menu, on voit apparaître le phénomène suivant : si par exemple on déplace le tracker d'un centimètre vers la gauche, le curseur se déplacera exactement de la même façon. On a donc une échelle 1 :1 entre le déplacement du tracker et celui du curseur.

Or la facilité de sélection d'une commande d'un menu 2D dépend fortement de la sensibilité du déplacement du curseur. Dans notre cas, cette dernière est fixe. C'est donc la taille du menu dans la scène qui influe sur cette caractéristique.

Le choix de la taille du menu devient alors problématique : plus il est grand, plus il est facile de sélectionner sans erreur une commande, mais plus il est "intrusif" dans la scène.

Nous avons donc eu l'idée de dissocier le curseur du menu du tracker.

Pour cela, nous rendons presque complètement transparent l'objet symbolisant le tracker (un cône), puis nous insérons un nouvel artefact (une sphère). Cela permet de réduire considérablement les confusions sur les objets manipulés.

Ensuite, le nouveau curseur se déplace uniquement dans le même plan que le menu. Si la position du tracker quitte ce plan, la position du curseur est le projeté de celui-ci sur le plan du menu.

Enfin, nous appliquons un coefficient multiplicatif à la distance entre le tracker et le centre du menu, afin de calculer la position du curseur. Si ce coefficient est supérieur à 1, la "sensibilité" du curseur est plus grande que le tracker. Sinon, elle l'est moins.

Au final, la formule vectorielle de position du curseur est :

$CM = p_{menu}(CT) * \alpha$

où C est le centre du menu, M la position du curseur, p_{menu} la projection sur le plan du menu, T la position du tracker, et enfin α le coefficient de sensibilité.

Dans le chapitre suivant, nous verrons quelle valeur de α permet une sélection efficace.

7.2.2 *Hypothèses sur les conséquences de la forme du menu*

Le FlowMenu3D a au départ été implémenté en gardant globalement la même forme que le FlowMenu : un octogone régulier.

Mais la question de la légitimité de cette forme se posait : l'octogone était-il adapté pour ce menu dans un environnement 3D ?

Lors de l'implémentation du FlowMenu3D, nous n'avons pas envisagé de limite supérieure à la distance du curseur par rapport au centre du menu : même "hors" du menu, la sélection est encore prise en compte (seule l'angle est pris en compte). Cela permettait en particulier d'atteindre de bien meilleures performances en mode expert, où la représentation du menu importe moins.

Nous avons donc décidé de tester une autre forme de menu, sans changer son mode de fonctionnement : un carré (cf. figure 51).

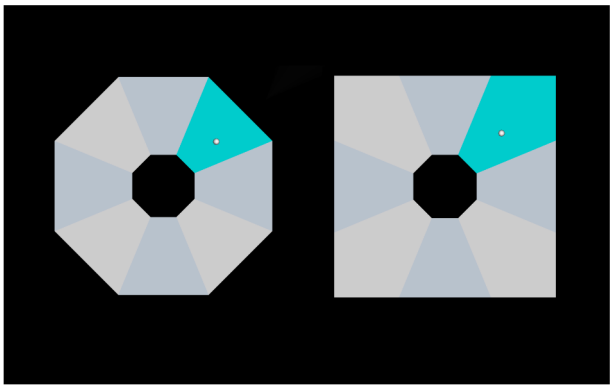


FIG. 51: Deux formes possibles du FLowMenu3D

Nous avons émis les hypothèses suivantes concernant l'influence de la forme du menu sur la sélection de commandes :

– la possible modification des gestes des utilisateurs en mode apprentissage. Nous avons pensé que cela pouvait modifier la position que les personnes "visent" pour sélectionner un onglet. En particulier, nous avions émis l'hypothèse que la forme en carré, en différenciant les onglet cardinaux (nord, sud, est et

ouest), permettrait de réduire le nombre d'erreurs de sélections de commande.
- la facilité d'apprentissage du mode expert, pour la même raison de différenciation des onglets cardinaux et non cardinaux.

7.3 UTILISATION DE LA COULEUR

Nous n'avons pas effectué d'étude poussée sur l'utilisation de la couleur pour ce menu. Néanmoins, nous avons noté certains points :

- l'utilisation du FlowMenu3D étant basée sur la sélection des onglets qui le composent, il est extrêmement important que l'on puisse facilement et rapidement distinguer ces derniers. Afin de ne pas surcharger le menu avec des artefacts graphiques (lignes, reliefs, etc), nous avons utilisé deux couleurs (deux types de bleu) que nous appliquons alternativement aux onglets. De plus, le fond de la scène observée est le plus souvent noir. Nous avons donc choisi des nuances de bleu relativement claires, pour qu'il n'y ait pas de difficulté à distinguer le menu.
- pour une lecture sans effort du texte affiché dans les onglets, il faut un contraste fort entre la couleur du texte et celle de l'onglet. Le point précédent précise que nous avons choisi des couleurs claires, ce qui est parfaitement en accord avec ce phénomène.
- il est important pour l'utilisateur d'avoir un retour d'information sur la sélection des différents onglets. La modification de la couleur est un moyen simple et très efficace, à condition de bien choisir les couleurs de signalisation : il est par exemple déconseillé de choisir une couleur rouge pour indiquer une sélection correcte, au risque de perturber l'utilisateur, la couleur rouge étant le plus souvent utilisé dans les signalétiques pour une erreur ou une interdiction. Notre choix s'est donc plutôt porté sur la couleur verte (cf. figure 52).

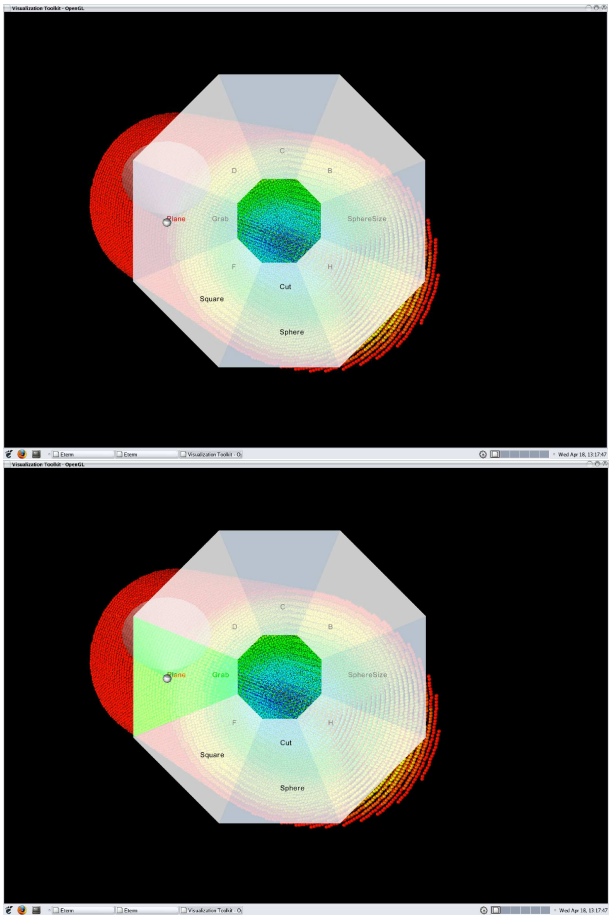


FIG. 52: Sélection avec et sans modification de la couleur

7.4 FLOWMENU ET MANIPULATION DIRECTE

Le FlowMenu est a priori parfaitement adapté pour une telle tâche :

- il ne demande qu'un tracker de position et d'orientation dans le cas d'un environnement 3D, et d'un seul et unique dispositif de sélection discrète (bouton), réduisant au strict nécessaire les périphériques d'entrée pour la manipulation directe.
- il permet à l'utilisateur, soit à l'aide du menu lui-même (mode novice), soit par une "gesture" (mode expert) de sélectionner la fonction qu'il souhaite et ensuite de la manipuler à souhait.
- là où le FlowMenu se démarque vraiment du reste des autres types de contrôle d'application, c'est par son retour au centre pour la validation de la fonction : en effet, parmi les prérequis de visualisation de système variant avec le temps, par exemple, il est nécessaire que la position spatiale des outils de visualisation (des widgets par exemple) soit fixe.

Le seul point qui peut sembler problématique est sans doute l'écart de position qui apparait lors de la validation d'une commande dans

le FlowMenu : en effet, elle s'effectue lorsque le tracker revient dans la zone "neutre" du menu, ce qui crée un écart équivalent à la distance entre le centre et un quartier au maximum entre la position d'origine et la position finale. Cet écart, pour des systèmes nécessitant une grande précision, pourrait être problématique. Deux solutions semblent envisageables :

- pour des environnements de réalité augmentée (ajout de virtuel dans un environnement réel), il est vraisemblablement nécessaire de garder l'écart entre la position de départ et celle d'arrivée, et de pallier ce problème par un artifice graphique (cela peut être un simple trait, ou une flèche), à moins de disposer de périphériques à retour d'effort (comme le *Spidar*[TCH+03], par exemple), et dans ce cas-là, forcer le retour du tracker au point d'origine.
- pour des environnements de réalité virtuelle pure (sans réalité augmentée), il est envisageable de simplement ne pas tenir compte de cet écart, et de replacer la représentation du tracker ou le widget au centre du menu : la position/orientation du tracker n'étant que relative, ce petit écart ne devrait pas perturber l'utilisateur.

8 IMPLÉMENTATION ET RÉSULTATS

Ce chapitre présente les expérimentations et résultats que nous avons mis en oeuvre sur les différentes possibilités d'intégration du *FlowMenu* dans un environnement immersif. Il présente également les outils logiciels et matériels qui nous ont permis de mettre en place les prototypes que nous avons conçus.

8.1 IMPLÉMENTATION : VTK ET VRPN

Afin de pouvoir utiliser facilement les différents travaux que nous avons effectué pour cette étude, il a fallu choisir une base logicielle commune.

VTK ne possède pas en propre d'extensions pour l'utilisation de périphériques de réalité virtuelle. Néanmoins, l'équipe du laboratoire de visualisation scientifique du service SNEC du CEA/DIF/DSSI a fourni[CG05] des classes permettant d'encapsuler les événements VRPN[VRP] dans des événements VTK. Ces classes apportent aussi en particulier des implémentations génériques pour des trackers.

VRPN (Virtual Reality Peripheral Network) est une bibliothèque de gestion de périphériques de réalité virtuelle par réseau. Elle intègre un serveur et un client générique, et des classes C++ permettant l'intégration de l'un ou l'autre dans un autre programme.

C'est par ces deux bibliothèques et ces extensions que nous avons implémenté le FLowMenu3D. Le menu en lui-même est implémenté dans une classe indépendante (affichage des onglets, gestion des événements du menu), et une classe fille a été créée pour l'intégration au code de vtkVRPN.

8.2 LE MATÉRIEL

L'équipement de test était composé :

- d'un ordinateur équipé d'un écran 22 pouces,
- d'un PHANToM Omni de la société SensAble[Sen]. Il s'agit un petit modèle de bras à retour d'effort, relativement bien adapté pour une utilisation de bureau. C'est principalement comme stylet "tracker" qu'il a été utilisé ici, plus que comme dispositif haptique (cf. figure 53).

Il faut noter que l'ensemble des tests ont été effectués sur station "desktop" uniquement pour des raisons de contraintes matérielles. Nous sommes conscients que le comportement des sujets pourrait être assez différent dans un environnement plus immersif (workbench, mur d'image, etc). En particulier, les taux d'erreurs et les vitesses de sélection pourraient varier de façon non négligeable. Néanmoins, il est peu probable que les résultats que nous présentons ici soient perturbés,

FIG. 53: le PHANToM Omni

ces derniers se basant non seulement sur une comparaison relative des données recueillies, mais aussi compte tenu de la nette différence de valeurs des données entre les différents modes testés.

8.3 LE POSITIONNEMENT

Comme nous l'avons vu dans le chapitre précédent, nous avions trois possibilités de positionnement pour le FlowMenu3D. Le positionnement centré tête a rapidement été écarté : en plus d'une intrusion complète dans le champ visuel, elle est extrêmement perturbante pour l'utilisateur, à cause de la trop grande distance dans la scène entre le curseur et le tracker.

Nous avons donc opté pour un positionnement centré tracker. Le tracker pouvant se déplacer dans toute la scène, et donc potentiellement "à l'intérieur" des objets de la scène, il a fallu réfléchir au moyen de rendre visible le menu chaque fois qu'il est appelé et utilisé. Nous avons choisi de faire simple : un plan de coupe de la scène, confondu avec le plan du menu, est activé chaque fois que le menu apparaît. Il est bien sûr désactivé lorsque la sélection de la commande est terminée, ou que le menu n'est plus activé.

Il nous restait donc à tester l'orientation du FlowMenu3D.

Lors des tentatives de l'orientation tracker (cf. figure 54), deux problèmes importants sont apparus :

- le menu est difficilement lisible avec un dispositif simple, en particulier sans stéréo, dès lors que le tracker n'est pas aligné avec la direction de vision. En pratique, si l'angle entre la direction du tracker et la direction de vision fait plus de 45 degrés, la lecture du texte devient problématique.
- avec cette orientation, la sélection des onglets est très peu intuitive : tous les utilisateurs qui ont essayé d'utiliser le FlowMenu3D dans ces conditions ont commencé par déplacer le tracker parallèlement au plan de perspective. Lorsqu'ils ont essayé de s'adapter à cette situation, il leur a été très compliqué de maitriser le

mouvement du curseur, ce dernier se déplaçant automatiquement dans le plan du menu. Il en est ressorti qu'aucun utilisateur ne pensait que le FlowMenu3D n'était utilisable tel quel.

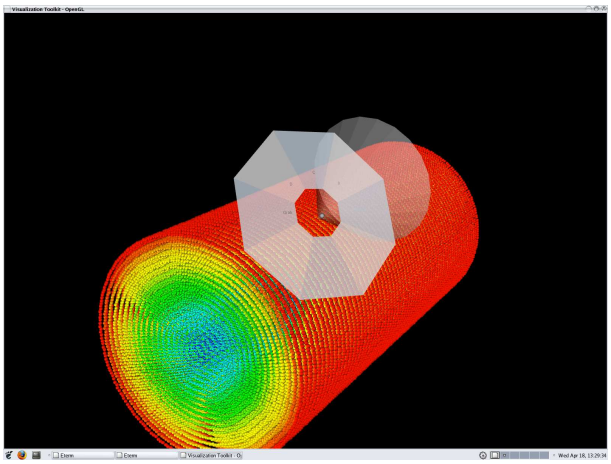


FIG. 54: Le FlowMenu3D, orienté tracker (prototype avec jeu de données représentatif d'une application à la physique des matériaux)

Nous avons donc choisi un positionnement centré tracker, et orienté tête, avec un plan de coupe de la scène pour la visibilité.

8.4 NOTE SUR L'ÉCHANTILLON D'UTILISATEURS POUR LES TESTS

Nous tenons à souligner que l'échantillon des sujets qui ont bien voulu participer à cette étude ne peut raisonnablement pas être représentatif de l'ensemble des utilisateurs potentiels de la solution.

En effet, cet échantillon est composé de 12 personnes, essentiellement de sexe masculin, avec une moyenne d'âge de l'ordre de 24 ans.

Néanmoins, nous estimons que les résultats que nous présentons ci-dessous sont assez flagrants pour laisser peu de place à la réserve concernant leur généralisation.

8.5 ETUDE SUR LE RAPPORT TAILLE DU MENU/SENSIBILITÉ DU CURSEUR

Le protocole de test est présenté en annexe de ce document.

Nous avons demandé aux sujets d'effectuer une cinquantaine de sélections d'un menu et d'un sous-menu, avec un retour au centre pour finir, et ce, pour quatres tailles et sensibilités différentes.

Pour éviter des problèmes de lecture de texte, nous avons identifié les onglets à sélectionner par une couleur différente.

Nous avons pris comme facteur de performance pour le menu :

- le pourcentage d'erreur de sélection
- le temps de sélection

Les sujets n'étaient pas au courant que l'opération était chronométrée. Il ne leur était donc pas demandé d'effectuer les sélections le

Nous rappelons que la sensibilité est ici le rapport entre la distance réellement parcourue par le tracker au centre du menu, et celle du curseur (virtuel), toujours au centre du menu. Par exemple, une sensibilité de $\frac{1}{2}$ signifie qu'à tout moment, la distance du curseur au centre du menu dans l'espace virtuel est deux fois moins importante que celle du tracker dans l'espace réel. Il faut alors faire "parcourir" le double de distance au tracker pour déplacer le curseur dans le plan du menu.

plus vite possible, mais plutôt d'éviter de commettre des erreurs de sélection.

Le diamètre du menu est soit de 10cm, soit de 5cm. La sensibilité du curseur est soit celle du tracker, soit deux fois moindre.

La figure 55 montre les résultats en terme de pourcentage d'erreur de sélection : rouge pour la faible sensibilité et la faible taille du menu, vert pour la sensibilité normale et la grande taille, et bleu pour la grande sensibilité et la petite taille. Les résultats pour la petite sensibilité et la grande taille ne sont pas présentés ici : les résultats donnent une moyenne d'erreur catastrophique (90% environ), la sélection des onglets n'étant réalisable qu'en limite d'amplitude du PHANToM Omni.

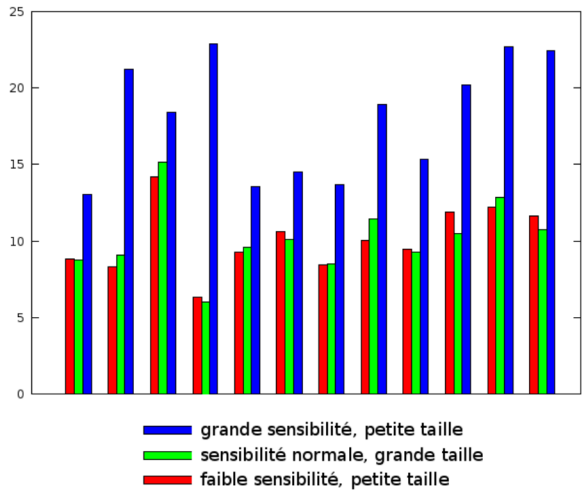


FIG. 55: Pourcentage d'erreur suivant la modification de la taille du menu et la sensibilité du curseur

Nous rappelons que la sensibilité doit être vue ici comme l'amplitude de déplacement pour parcourir le menu.

Un premier examen nous indique que le taux d'erreur augmente considérablement pour une sensibilité identique à celle du tracker et une petite taille de menu. Cela est compréhensible : le menu est plus petit, il faut donc effectuer de plus petits mouvements pour sélectionner les onglets. On passe d'une moyenne de 10,09% à près du double (18,08%). Sachant que le but de cette étude était de savoir si l'on pouvait réduire la taille du menu, une première réponse est bien évidemment : pas dans ces conditions.

En revanche, on peut aussi remarquer que pour la quasi-majorité des sujets, le taux d'erreur est pratiquement le même pour un petit et un grand menu, dès lors qu'on modifie la sensibilité du curseur en conséquence : les moyennes sont de 10,11% et 10,09%, et pour la différence entre les deux taux d'erreur, on a 0,02% de moyenne, et surtout un écart-type de 0,85%, ce qui est très faible.

On pourrait alors se dire que les sujets compensent la petitesse du menu en prenant plus de temps pour obtenir le moins d'erreurs possibles. Or en moyenne, nous avons 1,34s pour le grand menu, et

1,36s pour le petit, soit une augmentation de 1,5%. L'écart-type de la différence est de 0,07%, ce qui montre bien la régularité de chaque sujet.

Ceci nous permet donc d'affirmer que les sujets manipulent le FlowMenu3D avec la même efficacité, quelle que soit sa taille, dès lors que la sensibilité du curseur est modifiée en conséquence. Il y a bien sûr une taille minimale du menu, en dessous de laquelle le texte n'est plus lisible sans effort.

Remarque : la dissociation visuelle du tracker et du curseur apporte un phénomène intéressant, que nous n'avons pas étudié de façon quantitative. Il apparait que les utilisateurs, bien qu'ils n'aient aucune contrainte sur le déplacement en profondeur du tracker, restent globalement proche du plan du menu. Il est probable que le curseur, qui reste constamment sur le menu, soit à l'origine de ce comportement.

8.6 ETUDE SUR LA FORME DU MENU

Les mêmes conditions que dans la précédente étude sont utilisées ici. Les facteurs de performance sont les mêmes (pourcentage d'erreur et temps de sélection).

La sensibilité et la taille sont fixées, mais la forme est modifiée. Nous avons testé deux formes possibles : le carré, et l'octogone (cf. figure 56).

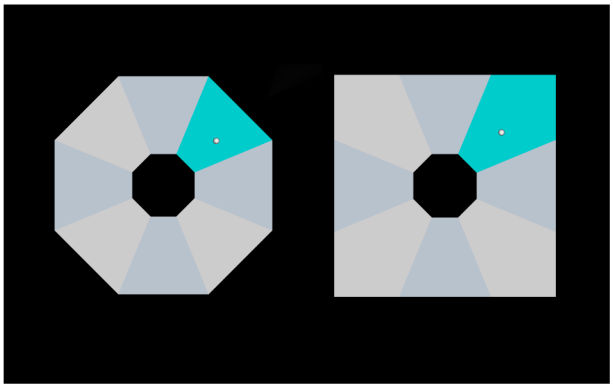


FIG. 56: Deux formes possibles du FLowMenu3D

Les résultats de cette étude sont beaucoup moins significatifs que dans la précédente étude. La figure 57 présente la différence entre le pourcentage d'erreur avec la forme carrée et celui avec la forme octogonale.

La moyenne de la différence est de -1,72%. Il ne se dégage apparemment aucune préférence globale, et l'écart-type de la différence le confirme : 5,1%, ce qui est élevé pour des valeurs comprises entre $-11,67$ et $4,03$. Cela permet donc de raisonnablement mettre en doute le lien entre changement de forme et performance.

Les temps de sélection n'apportent pas plus de résultats tangibles : 1,39s pour le carré, contre 1,37s pour l'octogone en moyenne, avec un écart-type de la différence de 0,19s, là aussi relativement important.

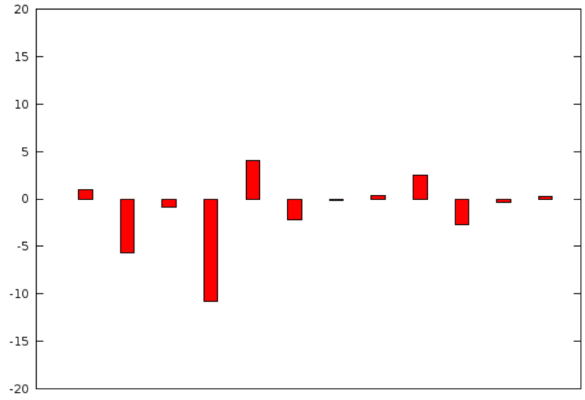


FIG. 57: Différence de pourcentage d'erreur entre la forme carrée et la forme octogonale : pour chaque sujet (en abscisse), l'ordonnée représente : Erreur_Forme_Octogonale – Erreur_Forme_Carrée

Les chiffres présentés fournissent une moyenne faible, avec un gros écart-type. Il est donc très difficile de croire qu'il se dégage un comportement général, comme cela est le cas pour l'étude précédente.

Il semble donc que la forme du menu n'est pas significativement liée à son efficacité. L'hypothèse de modification du comportement d'utilisation du FlowMenu3D suivant sa forme est peut-être vraie, mais de toute façon elle n'apporte aucune amélioration d'efficacité du menu.

8.7 MANIPULATION DIRECTE

Nous avons implémenté deux fonctions pour illustrer la manipulation directe avec le FlowMenu3D.

La première fonction est la manipulation d'un objet sélectionné par le tracker. Il s'agit d'une manipulation au sens de Hand, Hodges et Bowman[Han97][BH99], c'est-à-dire qu'il est possible de modifier sa position et son orientation (cf. figure 58).

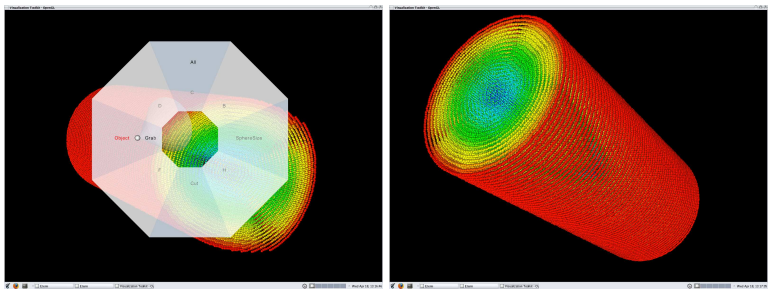


FIG. 58: Manipulation directe : manipulation d'un objet de la scène

On pointe le tracker sur l'objet, puis on appuie sur le bouton, ce qui fait apparaître le FlowMenu3D. Une fois que la commande

"Grab | Object" est sélectionnée, le menu disparaît. Tant que le bouton est enfoncé, le tracker est remplacé par l'objet : la manipulation s'effectue donc comme si l'objet virtuel se trouvait dans la main de l'utilisateur.

L'autre fonction implémentée est le plan de coupe, présentée en figure 59

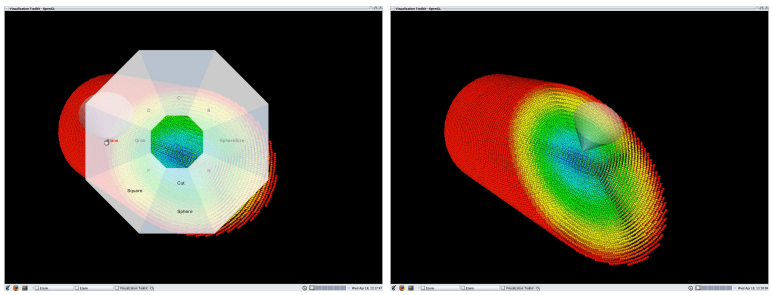


FIG. 59: Manipulation directe : le plan de coupe

De la même manière que la précédente fonction, une fois que la commande "Cut | Plane" est activée, le menu disparaît, et un plan de coupe, normal à la direction du tracker et passant par sa position, est ajouté à la scène. Tant que le bouton reste enfoncé, le plan reste normal à la direction du tracker.

Il est relativement difficile de concevoir des protocoles quantifiant l'intuitivité de telles fonctions. Néanmois, tous les sujets qui ont essayé ces fonctions déclarent n'avoir eu aucune difficulté à manipuler ces commandes, dès lors qu'ils ont compris le principe de sélection du FlowMenu3D.

8.8 REMARQUE SUR LA MANIPULATION DU FLOWMENU3D

Nous finirons ce chapitre par une remarque d'ordre pratique sur la manipulation du FlowMenu3D.

Lors des nombreuses manipulations du FlowMenu3D avec le PHANToM Omni, nous avons remarqué qu'un mouvement du bras particulier permettait d'être beaucoup plus efficace.

Il suffit, lors de la sélection des onglets, de garder le bras et l'avant-bras le plus immobile possible, et d'utiliser uniquement le poignet pour déplacer le curseur (cf. figure 60).

Cela permet non seulement de garder le tracker le plus proche possible de son point d'origine, mais aussi un apprentissage beaucoup plus rapide du mode "expert". Il est probable que tout cela provienne de la réduction du nombre de degrés de liberté que l'on applique au bras : le tracker n'est plus dirigé qu'avec deux degrés de liberté, en rotation.

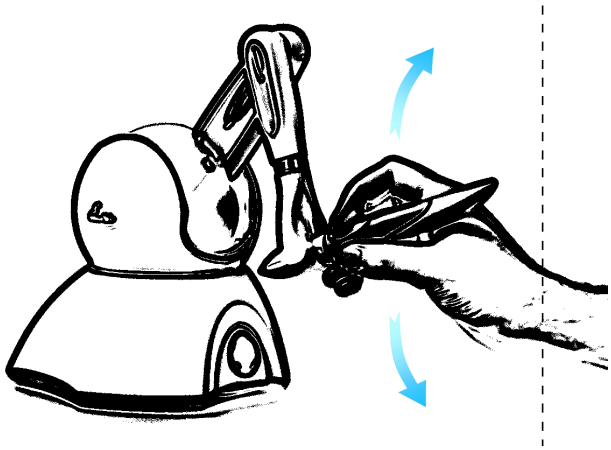


FIG. 60: Recommandation sur la manipulation du FlowMenu3D.

9 CONCLUSION

Les résultats que nous avons obtenus semblent principalement montrer, encore une fois, malgré le petit nombre de participants aux expérimentations, que :

- L'intégration du *FlowMenu* en environnement immersif est relativement efficace, d'un point de vue du taux d'erreur (10% en moyenne) et de vitesse (1,35s en moyenne) de sélection, du moment que le menu est centré tracker, et orienté tête.
- Il existe un rapport optimal entre la taille du menu et la sensibilité du tracker. Il est donc possible de réduire l'encombrement visuel du *FlowMenu3D*, tout en gardant une efficacité optimale de sélection (en gardant à l'esprit que les différents onglets doivent rester visibles sans effort).

Il reste énormément de sujets d'étude possible sur l'adaptation du *FlowMenu* en environnement immersif. Parmi ceux-ci, les points suivants nous semblent les plus intéressants :

- Il serait intéressant d'effectuer les mêmes tests avec un système matériel de positionnement différent. On pourrait imaginer des tests avec un système de tracker magnétique, de type Fastrak, ou mécanique, comme le Spidar. Les perceptions liées à ces deux solutions ne sont absolument pas les mêmes qu'avec un bras haptique (inertie du tracker modifiée pour le Spidar, et absence de véritable lien matériel pour le Fastrak). Une étude sur l'influence de ces perceptions sur l'utilisation du *FlowMenu3D* pourrait être envisagée.
- L'utilisation du retour d'effort du bras haptique est peut-être possible. Il pourrait simplement servir pour la restriction du mouvement (dans le plan du menu, par exemple), ou d'aide à la sélection (en attirant le tracker vers le centre des quartiers).
- Nous n'avons utilisé la couleur dans cette étude que de façon basique. Il pourrait être intéressant d'étudier plus précisément le retour d'information quelle pourrait apporter à un tel menu (distinction plus facile des quartiers, apprentissage plus rapide du menu), et également son influence sur la lisibilité du menu et de ses onglets.
- En plus du lien possible entre apprentissage et couleur, il serait probablement intéressant de travailler sur le passage du mode "novice" au mode "expert". On pourrait très bien imaginer modifier le temps où le menu reste visible, ajouter une latence d'apparition, réduire la taille du menu en fonction de la vitesse des précédentes sélections, par exemple.

Troisième partie

VISUALISATION DE SYSTÈMES DENSES À GRAND NOMBRE DE PARTICULES

Cette troisième partie présente l'ensemble des solutions choisies ou conçues pour la conception d'un système d'exploration d'une scène composée d'un grand nombre de particules représentant un matériau dense.

Le but est d'essayer d'atteindre sur un mur d'images piloté par un cluster de rendu, donc sur une architecture parallèle à haute résolution, un taux de rafraîchissement suffisamment élevé pour permettre une exploration de la scène.

Sont présentés ici :

- les optimisations que nous avons choisies d'utiliser pour le rendu de sphères colorées,
- notre proposition de solution de visualisation adaptative conçue spécialement pour les systèmes particulaires, basée sur une simplification de la scène par occlusion statistique,
- la description de notre architecture parallèle adaptée à la précédente solution.
- notre implémentation, et les résultats que nous avons obtenus sur un petit mur d'images pour un cas typique de scène visualisée en Physique des Matériaux.

10 LE RENDU DE SPHÈRES

Ce chapitre évoque la question du mode de représentation par sphères et la méthode d'optimisation simple retenue.

10.1 POURQUOI DES SPHÈRES ?

On peut se demander si la représentation des particules sous forme de sphères est bien adaptée à la représentation de phénomènes globaux en physique des matériaux.

Il est probable que la raison première de l'utilisation fréquente d'une telle représentation est historique : elle vient du modèle des sphères dures, représentant au départ les atomes, modèle qui est encore souvent utilisé dans d'autres domaines, comme la cristallographie.

On pourrait aussi penser que la sphère est utilisée car c'est la seule forme géométrique à symétrie sphérique. Elle est donc parfaitement adaptée à la représentation d'objets dont les caractéristiques étudiées sont considérées comme isotropes, dans le modèle considéré, ou à l'ordre o d'un modèle plus élaboré, par exemple.

Or, dans le domaine de la physique des matériaux, c'est le comportement global des particules qui est étudié, et non pas chaque particule indépendamment. C'est pourquoi il est raisonnable de penser que la sphère est utilisée car c'est la forme géométrique tridimensionnelle nécessitant le moins de paramètres.

Pour définir une sphère dans une scène, trois paramètres suffisent : une position, un rayon, et une couleur. Mais ces paramètres n'ont absolument pas le même impact en terme d'information pour l'observateur de la scène.

- La position donne l'emplacement précis de l'objet. Elle permet donc de localiser l'information contenue dans ce dernier.
- Le rayon influe directement sur la taille de l'objet, ce qui modifie sa visibilité. Il faut donc garder à l'esprit que la modification de ce paramètre agit à la manière d'un filtre sur l'information produite par la visualisation de la scène. Plus le rayon d'une sphère est grand, plus celle-ci contribue au rendu final qui sera interprété par l'observateur. Il est aussi intéressant de noter que le rayon d'une sphère n'agit pas proportionnellement sur la contribution à l'information globale. Considérons que l'impact de l'objet sur l'interprétation de la scène (pour des phénomènes globaux) est lié au nombre de pixels rendus, donc à sa surface. On peut alors supposer qu'il vaut mieux s'intéresser au carré du rayon, plutôt qu'au rayon lui-même, pour évaluer la quantité d'information apportée par la sphère à la scène. Ou, inversement, si la surface apparente doit traduire visuellement l'intensité d'une variable, on peut utiliser un disque (projection de la sphère) de rayon en racine

carrée de la variable, afin de compenser cet effet et "linéariser" la perception.
- La couleur est en fait le véritable vecteur d'information pour la visualisation de phénomènes globaux particulaires. Elle permet par exemple de faire apparaître des zones d'intérêt, ou de montrer l'évolution d'une valeur dans l'espace.

On pourrait alors se demander si l'utilisation d'autres formes géométriques, qui ont besoin de plus de paramètres, est pertinente. Par exemple, des formes telles que les ellipses peuvent être utilisées pour la représentation de certains tenseurs[Gum03], car ces derniers sont constitués d'une matrice, qui ne peut donc pas être représentée par une seule valeur scalaire.

On pourrait donc imaginer utiliser des ellipses au lieu des sphères pour représenter d'autres informations, comme la vitesse, l'accélération, ou des paramètres scalaires (énergies, charges, etc). Mais cela risque de surcharger la scène, Même si l'idée de visualiser la vitesse en utilisant sa direction comme axe principal d'une ellipse peut sembler intéressante, il faudra prendre en compte l'augmentation de la surface visible de l'ellipse par rapport à la sphère pour rester cohérent.

De plus, il est essentiel de ne pas lier plusieurs paramètres à un seul canal d'information, car l'observateur aura beaucoup de difficultés à interpréter la scène. Si nous reprenons comme exemple l'ellipse, et que nous utilisons ses trois axes principaux pour représenter l'énergie potentielle, l'énergie cinétique, et la charge de la particule, il devient impossible d'interpréter correctement la scène.

En résumé, la représentation sphérique des particules est une des plus maitrisées et utilisées, car une des plus simple. Il faut toutefois faire attention à lier les bons paramètres aux bonnes valeurs, afin d'obtenir une scène qui permette une visualisation optimale.

10.2 RECOMMANDATIONS SUR LE CHOIX DES COULEURS

Comme nous l'avons vu dans la section précédente, la couleur de la sphère est le véritable vecteur d'information lors de la visualisation de systèmes de particules sphériques. C'est pourquoi le choix de la transposition de la variable en couleur (*colormap*) n'est pas à prendre à la légère.

La colormap, ou carte de couleurs, est une fonction de transfert, établissant la correspondance entre une échelle de valeurs et une gradation de couleurs.

Rogowitz et al. proposent des règles de sélection de carte de couleurs, nommée PRAVDA[RTB96] (Perceptual Rule-based Architecture for Visualizing Data Accurately).

On y retrouve entre autre les idées suivantes :
- une retranscription "naïve" en RGB (rouge,vert,bleu) est très éloigné de la façon dont un humain perçoit les couleurs. Une représentation de la couleur plus en adéquation avec la perception humaine est le système HSV (luminance, teinte, saturation).
- Pour la luminance ou la saturation, la grandeur perçue est en fait une loi en puissance : il faut donc faire des corrections de transposition en conséquence.

– PRAVDA propose donc des cartes de couleur dites "isomorphiques", où chaque différence de valeur est transposée en différence perceptuelle équivalente.
– Suivant l'importance des fréquences spatiales dans l'interprétation des données, il est possible de choisir des cartes de couleurs plus adaptées. Les variations de luminances permettent par exemple de mieux percevoir les informations à hautes fréquences spatiales (les détails), alors que la teinte et la saturation sont plus indiquées pour des informations à basses fréquences (les structures).

La figure 61 présente un exemple de modification de la perception avec l'utilisation de différentes cartes de couleur : la carte de couleurs "hautes fréquences" (à gauche), basée sur la luminance, révèle plus de détails sur les données du radar (en bas). La carte de couleurs "basses fréquences" (à droite), basée sur la saturation fait plus ressortir la structure des données météorologiques (en haut).

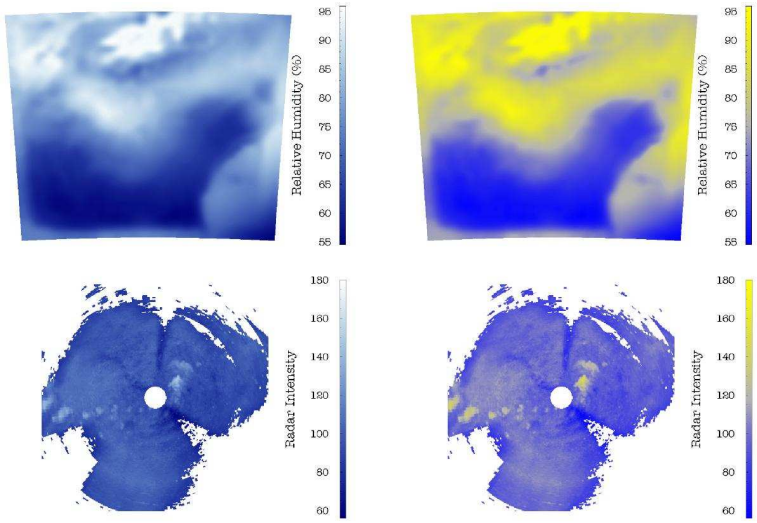


FIG. 61: PRAVDA : influence des *colormap* sur la perception des structures et des détails de haute ou basse fréquence spatiale.

10.3 L'ILLUMINATION

Représentation du volume d'une sphère

Il existe peu de procédés qui amènent à la perception du volume d'une sphère. Excepté la représentation d'une intersection avec un autre élément de la scène, seul un modèle d'illumination surfacique pertinent peut donner des informations directement interprétables pour l'observateur.

Le modèle de Phong

Bien qu'il existe de nombreux modèles d'illumination, celui de Phong[Pho75] donne d'excellents résultats sur une sphère, et a le mérite d'être un modèle local, ce qui permet une implémentation efficace dans un Fragment shader.

La formule locale de l'illumination donnée par Phong est donnée par :

$I = I_a k_a + I_i(k_d(L.N) + k_s(R.V)^n)$

Avec :

- I_a et I_i respectivement les intensités lumineuses ambiante et incidente,
- k_a, k_d, et k_s respectivement les coefficients de réflexion ambiante, de réflexion diffuse, et de réflexion spéculaire,
- n la "réflectivité" de la surface,
- L et R respectivement la direction de la lumière diffuse et spéculaire,
- N et V respectivement la normale à la surface en ce point et la direction d'observation.

On ne parle pas ici de "réflectivité" au sens optique du terme. Ce paramètre est bien lié à la réflectance du matériau, c'est-à-dire le rapport de l'énergie réfléchie sur la puissance incidente, mais c'est un modèle très simplifié de son comportement vis-à-vis de la réflexion de la lumière

Application à une sphère dans un couple Vertex/Fragment shader

Le Vertex shader étant exécuté une seule fois pour un élément donné, nous devons y récupérer, ou calculer s'il y a lieu, les valeurs qui serviront pour l'ensemble des points visibles de la sphère :

- les intensités lumineuses,
- les coefficients de réflexion et la réflectivité,
- les directions des lumières : on peut supposer que pour une sphère donnée, la lumière est relativement loin, ce qui permet de considérer les directions comme fixes pour cet élément,
- la direction de l'observateur, triviale en espace image (vecteur unitaire dans la direction de la profondeur).

Il ne reste plus au Fragment shader qu'à calculer la normale à la surface. Pour ce faire, nous utiliserons une formule simple qui dépend de la distance au centre de la sphère (distance en x,y dans le plan de l'observateur, et non pas distance réelle, cf. figure 62).

Le Fragment shader peut alors assigner au pixel la bonne couleur, ainsi que la profondeur.

Cette méthode est fortement inspirée de la solution TexMol[BDST04].

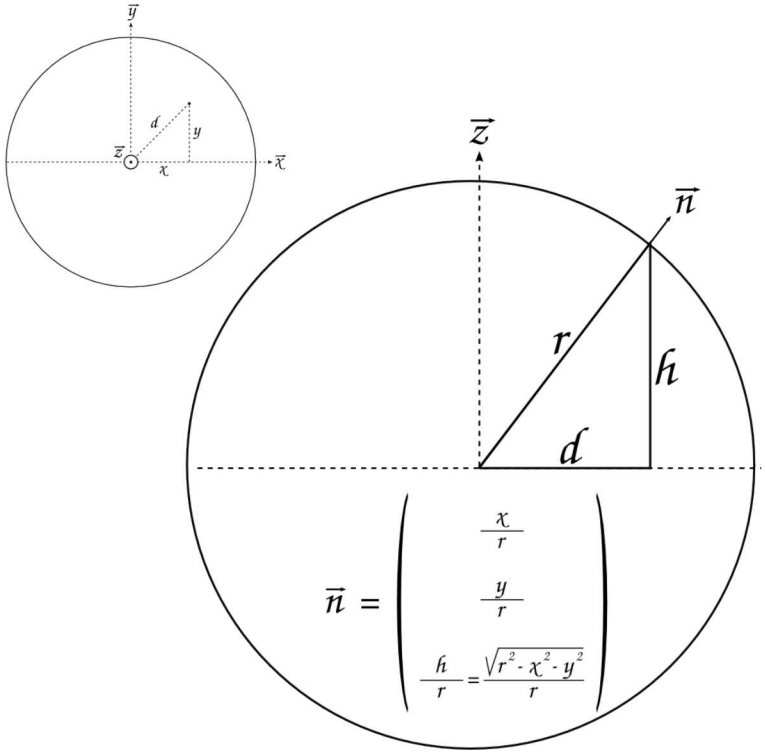


FIG. 62: Calcul de la normale à une sphère en un point

10.4 BILAN : QUE CHOISIR POUR REPRÉSENTER UN GRAND NOMBRE DE SPHÈRES EN OPENGL ?

L'architecture OpenGL actuelle nous a donc poussé à choisir un modèle d'éclairage de Phong, implémenté dans un couple Vertex/-Fragment shader, avec les données nécessaires à la représentation des sphères stockées dans des VBOs. Nous verrons plus loin que le choix des VBOs est très important dès lors que les données manipulées sont hiérarchisées : on peut alors utiliser ces derniers comme un système de gestion de cache de la mémoire graphique.

11 VISUALISATION ADAPTATIVE

Les jeux de données particulaires en Physique des Matériaux étant particulièrement imposants, il est impératif, pour conserver un rafraîchissement acceptable, d'éliminer en phase de pré-rendu le maximum d'éléments de la scène qui ne seront pas visibles lors de l'affichage.

Le problème de ce type de scène est que le calcul d'occlusion réelle des sphères par leurs congénères n'est pas raisonnablement possible, puisque la complexité atteint l'ordre de N^2, N étant le nombre de sphères, et ce pour chaque image rendue (l'occlusion dépend bien évidemment du point de vue utilisé pour visualiser la scène).

Ce chapitre propose une méthode d'estimation statistique de l'occlusion de groupes de sphères par d'autres, avec un découpage de la scène en Kd-Tree.

Il est à noter que cette solution s'applique lors du pré-rendu, ce qui la classe parmi les méthodes *sort-first*.

11.1 UNE APPROCHE DE *culling* POUR UN SYSTÈME PARTICULAIRE DENSE

11.1.1 *Choix de la stratégie de culling*

Nous avons choisi de nous inspirer de la technique d'Atomsviewer, plutôt que de partir sur une technique de HOMs, et ce pour plusieurs raisons :

Les HOMs sont décrits en section 4.4.1

- l'efficacité des HOMs est notable pour des scènes comportant des objets complexes et peu nombreux. Avec des systèmes représentés par des sphères, nous ne sommes pas vraiment dans ce cas de figure. De plus, en utilisant des systèmes de hiérarchisation de données, par exemple des arbres de partitionnement d'espace, qui amènent à manipuler des boîtes englobantes, la complexité des objets de la scène décroît fortement.
- Pour une utilisation efficace des HOMs, les "prérendus" sont effectués par la carte graphique. Ces derniers consomment donc une quantité de mémoire GPU non négligeable, au détriment des données réutilisables pour de futurs rendus (notion de cache, cf. le chapitre 3).

11.1.2 *Notion de densité*

Sharma et al.[Sha02] introduisent une notion de densité de particules dans une cellule de l'octree qui est utilisé dans Atomsviewer. Cette densité est utilisée pour réduire le nombre de particules affichées en jouant sur la probabilité qu'elles soient cachées par d'autre particules.

Il est bien évident que cette densité n'est pas nécessairement la quantité de particules par unité de volume.

L'enjeu est ici de trouver un moyen de quantifier la capacité qu'une cellule a de cacher une autre cellule. En cela, on se rapproche plus de la notion d'opacité.

Nous nous sommes d'abord intéressés à la définition "naïve" de la densité, à savoir celle qu'utilise Atomsviewer.

$$D = \frac{\text{Volume des sphères}}{\text{Volume de la cellule}}$$

De par sa simplicité, cette forme de densité est très rapide à calculer. On pourrait penser que cela donne une bonne estimation de la capacité d'occultation de la cellule, mais cela n'est vrai que si les sphères sont uniformément réparties dans la cellule. Or si cela peut être vrai dans le cas de biologie moléculaire, comme l'affirment les auteurs, où les particules sont structurées, c'est beaucoup plus rare en physique des matériaux. Il n'est par exemple pas rare de voir apparaître des fissures ou des trous dans les modèles étudiés.

De plus, dans le cas où les sphères ne s'intersectent pas, cette densité ne peut dépasser un seuil souvent largement inférieur à 1. Par exemple, dans le cas de sphères de même rayon, le taux de remplissage (ou compacité en cristallographie) maximum est de $\frac{\pi}{3\sqrt{2}}$ soit environ $0,74$. Il est donc relativement difficile d'interpréter une telle valeur en terme de capacité d'occultation.

Nous avons donc étudié une autre approche de la notion de densité, pour les cellules du Kd-Tree que nous utilisons pour notre part (détails dans section 11.1.3). Nous sommes partis du constat que la capacité d'occultation se rapprochait de la notion d'opacité, donc, en simplifiant, la manière dont la lumière traverse ou pas l'objet.

L'utilisation d'un Kd-Tree est similaire à celle d'un octree dans ce cas, mais elle permet d'adapter le découpage hiérarchique de l'espace suivant la répartition des données (c.f. section 11.2).

Cette densité devient alors la probabilité qu'un rayon lumineux a de traverser la cellule. Le calcul s'effectue en utilisant une méthode de Monte-Carlo (cf. figure 63).

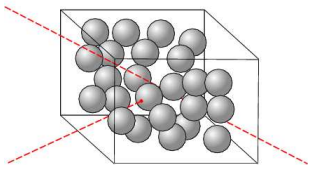


FIG. 63: Méthode de Monte-Carlo pour le calcul de densité

Nous envoyons N rayons de position et direction aléatoire à travers la cellule, et nous vérifions pour chaque rayon qu'il ne percute pas une sphère. En comptant le pourcentage de rayons qui traversent la cellule sans intersecter de sphère, on obtient une estimation de la densité précédemment définie.

Le problème de cette méthode est bien évidemment de trouver un nombre N de rayons qui permet d'obtenir une estimation correcte de la densité, sans toutefois trop augmenter le temps de calcul. En effet, chaque rayon possède 5 degrés de liberté (3 en position, 2 en orientation), donc chaque lancé s'accompagne de 5 générations de nombres aléatoires. Il est donc important de réduire le plus possible le nombre de lancés.

La théorie de Monte-Carlo donne comme vitesse de convergence de l'erreur $1/\sqrt{N}$. Cela ne permet pas de donner une estimation de N, mais nous pouvons quand même en déduire une évaluation empirique.

D'après les tests que nous avons effectué, il semble que, pour obtenir une estimation à 1% près de la densité d'une cellule, il faille un nombre de lancés de l'ordre de 100000.

De par sa nature, nous avons appelé cette méthode "statistical occlusion culling".

11.1.3 *Occultation d'une cellule par une autre*

Maintenant que nous avons une estimation de la capacité d'occultation des cellules, il faut savoir si une cellule est effectivement devant une autre, au sens de la position de l'observateur.

Comme nous l'avons vu plus haut, nous n'avons pas choisi les HOMs pour cela. Nous avons plutôt opté pour une stratégie de calcul de silhouette, et de comparaison des sommets de la cellule en espace image.

Rappelons qu'une cellule, pour un octree ou un Kd-Tree, est un parallélépipède.

Silhouette

Nous cherchons à identifier les sommets de la cellule qui entrent véritablement dans le processus d'occultation. En pratique, dans le plan de vision de l'observateur, la cellule est un polygone donc les sommets coïncident avec une partie de ceux de la cellule.

Pour effectuer une comparaison d'occultation, il suffit alors de comparer la silhouette avec les points de la cellule potentiellement cachée, et d'effectuer un test de profondeur (en espace image, cf. figure 64).

L'algorithme de génération de silhouette est le suivant : nous calculons d'abord les coordonnées des 8 sommets de la cellule en espace image. Puis nous appliquons l'algorithme de marche de Graham[Gra72] afin de trouver les points qui composent l'enveloppe convexe de la silhouette.

La marche de Graham :

- Nous trions les points par ordonnée croissante. Nous connaissons alors le sommet le plus bas et le plus haut.

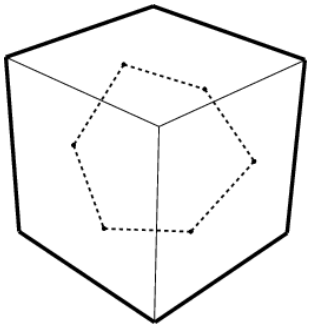


FIG. 64: Silhouette d'une cellule cachant une autre cellule

– Nous séparons les points suivant qu'ils sont à gauche ou à droite de la droite passant par le sommet le plus haut et le plus bas. Nous avons alors deux listes de points.
– Pour la liste gauche, nous vérifions que trois points consécutifs ne forment pas un angle supérieur à π. Si tel est le cas le point central est suprimé de la liste. Ceci est effectué pour tous les triplets consécutifs de la liste.
– Nous appliquons la même méthode à la liste droite, avec un angle qui doit être, pour sa part, supérieur à π.

La marche de Graham fournit donc deux listes de points qui correspondent aux deux branches de la silhouette (gauche et droite).

Position d'un point par rapport à une silhouette

Nous cherchons maintenant à savoir si un point est à l'intérieur de la silhouette (cf. figure 65).

Pour cela, on compare son ordonnée à celles des éléments des listes. Si le point est plus bas que l'élément le plus bas, ou plus haut que l'élément le plus haut, la réponse est immédiate. Sinon, cela permet de trouver les segments gauche et droit qui possèdent un point avec cette ordonnée.

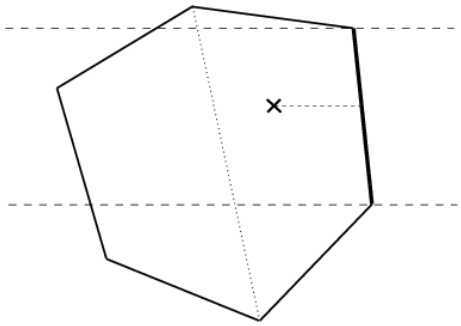


FIG. 65: Test de la position d'un point par rapport à une silhouette

On vérifie alors que le point est à droite du segment gauche, et à gauche du segment droit. Si oui, il est à l'intérieur de la silhouette. Sinon, il est à l'extérieur.

Profondeur

Dans ce qui précède, la profondeur du point n'est pas prise en compte. Et pour cause : la marche de Graham n'est qu'un algorithme 2D. Pour pouvoir utiliser tout cela dans un algorithme de *culling*, il manque un test de profondeur.

Un test simple est de vérifier que la profondeur du point est supérieure à celle de tous les points de la silhouette. Or cela ne prend pas en compte le fait que l'occultation est un phénomène surfacique : seule l'enveloppe visible de la cellule cache l'autre cellule. On a alors souvent toute une partie de l'espace cachée par la silhouette qui n'est pas prise en compte par ce test (cf. figure 66).

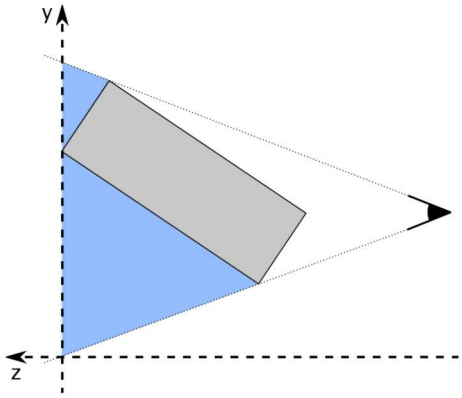


FIG. 66: Test de profondeur simple : perte d'espace caché

C'est pourquoi nous avons conçu un autre test de profondeur.

Nous prenons comme hypothèse de départ que le point n'est pas à l'intérieur de la cellule. Ceci est parfaitement cohérent avec l'utilisation de l'algorithme de culling, puisque ce point est a priori un des sommets de la cellule qui est potentiellement cachée.

On reprend les deux segments utilisés dans le calcul de position du point par rapport à la silhouette. Pour chacun, on utilise le sommet de l'autre segment pour définir un triangle. On a alors un ou deux triangles, qui définissent un ou deux plans. Si un point est derrière la silhouette, il est forcément derrière le ou les plans ainsi définis (cf. figure 67).

Il suffit donc de calculer la position du point par rapport à ces plans pour savoir si ce dernier est bien caché par la silhouette.

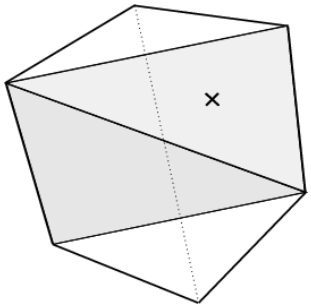


FIG. 67: Vérification de la profondeur : triangles définissant les plans

11.2 LA HIÉRARCHISATION DES DONNÉES

Utilisation d'un Kd-Tree pour la recherche de sous-espaces denses

Comme nous l'avons évoqué au chapitre 2, nous nous intéressons à des systèmes particulaires figurés par un grand nombre de sphères, le plus souvent représentant partiellement de la matière dense.

Nous sommes partis du constat simple suivant : un système dense de particules est en grande partie composé de zones ayant un nombre élevé de particules et de zones vides. Notre but étant de mettre en place un système de culling, il nous faut détecter à tout prix les zones vides, afin d'obtenir des cellules les plus denses possibles, et qui pourront potentiellement cacher d'autres cellules.

Choix d'un arbre pour des systèmes de particules denses

Nous avons choisi un Kd-tree comme arbre de partitionnement d'espace. Ses avantages sont les suivants :

- il est composé de cellules uniquement parallélépipédiques (contrairement au BSP tree). Cela réduit fortement la complexité de création de silhouette par marche de Graham, et permet donc au système d'occultation précédemment cité d'être efficace.
- il permet un positionnement variable des plans de partition de l'espace, contrairement à l'octree. Il permet donc de s'adapter à la densité des données, en particulier pour permettre un découpage plus précis des zones non vides.
- il est très rapide à calculer, puisque les plans de coupe sont parallèles aux axes : les tests d'appartenance à une branche de l'arbre sont de simples comparaisons entre une coordonnée et une valeur.

On peut noter comme principal inconvénient qu'il est beaucoup moins performant pour des scènes où les surfaces englobantes ne sont alignées avec aucun axe du repère choisi. Il est alors difficile de choisir un découpage efficace.

Arbre de détection de zones vides

Nous avons d'abord conçu un algorithme très simple de génération d'arbre (cf. figure 68) :

- Pour chaque noeud de l'arbre, un plan de coupe est sélectionné (comme tout Kd-Tree). Nous alternons suivant x, y, z.
- On recherche alors les espaces vides à gauche et à droite de la cellule : si un tel espace est trouvé, on place le plan de coupe à sa limite. Si l'on trouve un espace vide à gauche et à droite, le plan de coupe est choisi pour que la partie vide, après coupe, ait le plus grand volume.
- Si aucun espace n'est trouvé, le volume est coupé en deux cellules de volumes égaux.
- On recommence pour les fils de ce noeud, jusqu'à avoir atteint la profondeur de l'arbre voulue.

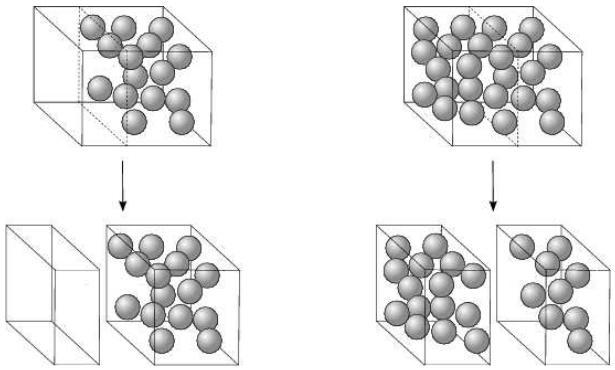


FIG. 68: Génération du Kd-Tree pour la recherche d'espace vide

En pratique, nous procédons comme suit :

- On crée un vecteur d'entiers, composés des entiers de 0 à $n-1$, correspondant aux n particules.
- Le vecteur est parcouru afin de trouver le minimum et le maximum des positions des particules selon l'axe. Ces extrema sont comparés aux limites de la cellule pour savoir s'il existe du vide à gauche et/ou à droite de celle-ci.
- Dans le cas où la cellule est découpée par le milieu, on effectue un début de tri par pivot (quicksort) : les particules à gauche de la coupe sont dans la partie gauche du vecteur, et respectivement celles à droites de la coupe vont à droite du vecteur.
- L'opération est reproduite pour les cellules filles, en utilisant respectivement les parties gauche et droite du vecteur.

La génération a donc une complexité en $O(n\ln(n))$, ce qui est l'équivalent des algorithmes de tri efficaces.

Optimisation de l'arbre

Nous avons ensuite cherché à optmiser l'arbre pour que ce dernier soit plus efficace pour l'opération d'occultation.

Le but principal du Kd-Tree est de fournir des cellules dont la densité est suffisamment forte pour qu'elles puissent être considérées comme opaques, mais aussi dont le volume est le plus grand possible, afin qu'elles cachent plus facilement les autres cellules. L'optimisation consiste donc à faire remonter dans l'arbre les coupes les plus efficaces, à savoir celles qui séparent les zones vides des autres zones.

Voici comment nous avons procédé : au cours de l'algorithme précédent, si aucune zone vide n'est détectée, la coupe s'effectue au milieu, mais la branche partant de ce noeud de l'arbre est temporaire. La génération de cette partie de l'arbre s'arrête lorsqu'une coupe de zone vide est trouvée, ou lorsqu'on atteint la profondeur d'arbre souhaitée. Cette coupe remplace alors celle du premier noeud, et l'algorithme est appliqué à ses fils. Dans le cas où aucune coupe de zone vide n'est trouvée, la branche est gardée telle quelle (cf. figure 69).

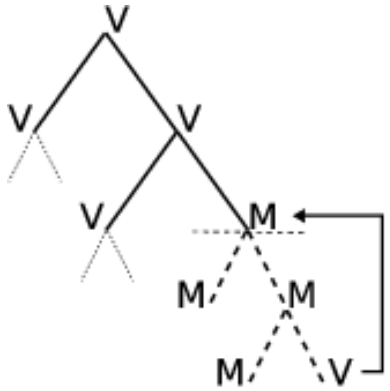


FIG. 69: Optimisation du Kd-Tree : remontée des coupes de zone vide (V) au détriment des coupes par le milieu (M)

Cela a pour effet de faire remonter toutes les coupes de zone vide le plus haut possible dans l'arbre.

Il est plus compliqué de trouver la complexité d'un tel algorithme. En pratique, nous avons constaté une très faible différence de temps de génération d'un arbre de même profondeur avec les deux méthodes présentées, pour une augmentation notoire des performances en matière de *culling*.

Autre tentative d'optimisation : le problème des données discrètes

Une autre idée qui avait été proposée pour l'optimisation de l'algorithme de départ était l'étude des variations de densité volumique dans la cellule pour sélectionner la position du plan de coupe. Nous pensions pouvoir isoler des "profils" de variations qui correspondraient à des zones d'intérêt, en particulier pour une détection de zones vides.

Or nous nous sommes heurté à un problème intrinsèque au type d'objet étudié : les données que nous manipulons sont discrètes, ce

qui a pour effet de faire apparaître dans des fonctions continues, comme la densité, des phénomènes à hautes fréquences, c'est-à-dire des variations très rapides. Il est alors très difficile de distinguer, par exemple, une fin de zone vide, et un espace très faible entre deux sphères. De plus, des filtres de type passe-bas, qui permettraient de se soustraire à ce genre de phénomènes, supprimeraient aussi des informations utiles, comme la limite d'une zone non vide.

Nous avons donc décidé de ne pas aller plus loin dans cette voie, qui nous a semblé être une impasse.

Performances et Profondeur de l'arbre

Il est relativement difficile de prédire a priori la profondeur de l'arbre qui donnera les meilleurs résultats en matière d'occultation. Plus l'arbre est profond, plus la scène est découpée, et plus on se rapproche de la quantité maximum d'éléments pouvant ne pas être affichés. En contrepartie, l'algorithme d'occultation est beaucoup plus long, puisqu'à chaque profondeur supplémentaire, on double le nombre d'éléments de l'arbre.

Deux critères semblent toutefois donner de bons résultats pour l'évaluation de l'efficacité d'un arbre. Il s'agit essentiellement de résultats empiriques, relevés lors des tests effectués pendant l'implémentation et l'utilisation de la solution créée.

LE NOMBRE DE PARTICULES PAR FEUILLE DE L'ARBRE , en moyenne : il très facile à calculer, puisqu'il est égal à :

$$\frac{\mathrm{NombreTotaldeparticules}}{2^{\mathrm{Profondeurdel'arbre}}}$$

C'est en fait le rapport entre la "granularité" des particules et celle de l'arbre. Il sert le plus souvent à donner un ordre de grandeur de la profondeur nécessaire. Pour la plupart des tests effectués dans cette étude, ce nombre était de l'ordre de 1000, ce qui nous semble être un nombre moyen relativement raisonnable pour effectuer des calculs statistiques d'occultation sur une feuille.

LE RATIO DES VOLUMES OCCULTANTS / NON VIDES : ce critère permet d'affiner l'évaluation de la profondeur de l'arbre a priori. On choisit d'abord la densité minimale pour laquelle une cellule est considérée comme opaque. Puis on évalue la quantité suivante :

$$\frac{\mathrm{Volumedesfeuillesopaques}}{\mathrm{Volumedesfeuillesnonvides}}$$

Les feuilles vides ne sont pas prises en compte dans l'algorithme d'occultation, puisqu'elle sont automatiquement considérées comme "cachées" (cf. la section suivante). Ce critère est donc le ratio entre le volume "efficace" au sens de l'occultation et le volume réel de la scène. Le profil de cette valeur est souvent de la forme de la figure 70.

On remarque que l'augmentation de cette valeur ralentit de façon assez nette à partir d'une certaine valeur de la profondeur. En pratique, cela veut dire que le découpage spatial supplémentaire

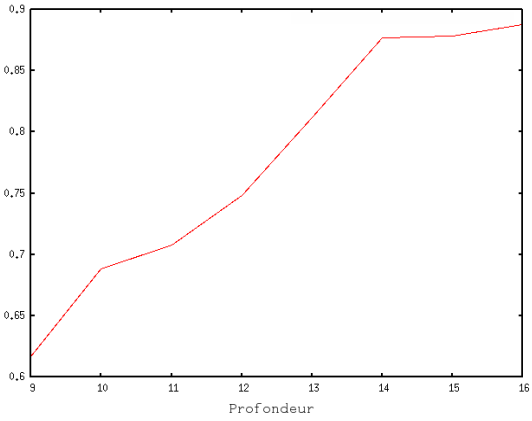


FIG. 70: Profil du critère Volume opaque / non vide.

apporté par les nouvelles profondeurs de l'arbre n'apporte presque plus d'information utile pour l'occultation.

Il apparaît que les performances en terme de vitesse d'affichage, avec la méthode d'occultation présentée dans cette étude, sont optimales pour cette profondeur, et ce pour tous les tests que nous avons pu effectuer.

Note sur le dépassement des sphères hors des cellules

L'algorithme de génération du Kd-tree présenté ci-dessus ne prend en compte que le centre des sphères pour trier celles-ci dans les différentes cellules de l'arbre. Il arrive donc que certaines sphères soient considérées comme à l'intérieur de la cellule, mais qu'au final elles "dépassent", d'au maximum leur propre rayon.

Nous n'avons malheureusement pas trouvé de solution élégante à ce problème. En effet, il est extrêmement fréquent de se trouver avec un plan de coupe de la cellule qui coupe automatiquement au moins une sphère (cf. figure 71).

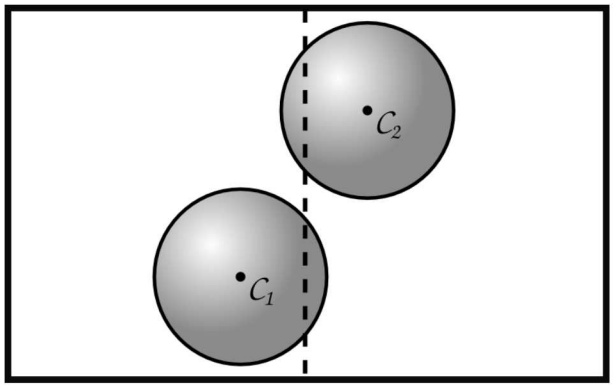


FIG. 71: Sphère et plan de coupe : exemple de cas à problème.

Nous avons donc essayé d'évaluer rapidement l'impact de ce phénomène.

Lors d'un test d'occultation d'une cellule par rapport à une autre, la silhouette de la cellule opaque frontale est comparée aux sommets de la cellule potentiellement cachée. S'il est probable que la cellule cachée possède des sphères qui "dépassent", il en est de même pour la cellule opaque, qui non seulement possède un taux de remplissage élevé, mais dont les sphères sont représentées plus grandes, puisque cette dernière est plus proche de l'observateur. De plus, avec un nombre moyen de sphères relativement élevé, l'erreur commise sur l'image finale est souvent très faible.

En pratique, pour des systèmes comportant plusieurs millions de particules, pour une résolution de 2048x1536, aucun artefact de ce type n'est apparu, probablement parce que l'erreur effectuée est plus petite qu'un pixel.

11.3 KD-TREE POUR LA VISUALISATION DE MATIÈRE DENSE DISCRÈTE

11.3.1 *Couplage culling et Kd-tree*

Nous nous sommes inspirés des travaux de Coorg et Teller[CT97], qui avaient proposé un algorithme intéressant pour le culling dans un kd-tree, mais avaient opté pour une méthode de culling assez différente, à base de plans tangents.

Les données manipulées étant mortes, nous pouvons facilement calculer certaines valeurs des noeuds de l'arbre avant toute opération de visualisation. Ainsi, pour notre méthode, nous devons calculer une estimation de la densité de tous les noeuds, ainsi que le maximum de densité pour la branche issue du noeud (cf. la sous-section suivante).

Chaque noeud possède trois états possibles : non occulté, partiellement occulté, et occulté. Par défaut, tous les noeuds sont considérés comme visibles. On considère qu'un noeud est partiellement caché lorsque ses fils ne sont ni tous les deux occultés, ni tous les deux non occultés. Les feuilles de l'arbre n'ayant pas de fils, elles ne peuvent être partiellement occultées.

Ci-dessous un tableau montrant l'état du noeud père en fonction de ses fils :

Fils 1 / Fils 2	Occulté	Partiellement	Non Occulté
Occulté	Occulté	Partiellement	Partiellement
Partiellement	Partiellement	Partiellement	Partiellement
Non Occulté	Partiellement	Partiellement	Non Occulté

Il est intéressant de noter qu'en codant l'état d'un noeud sur deux bits de la manière suivante, on obtient une relation très simple :

occulté	Partiellement	Non occulté
01	11	10

$$\text{Etat du noeud} = \text{Etat du Fils gauche} | \text{Etat du Fils droit}$$

| étant l'opération binaire "OR".

Ainsi, si un noeud change d'état, il est très rapide de modifier l'état de ses pères, s'il y a lieu.

La première étape de notre méthode est, comme indiqué plus haut, de fixer un seuil de densité à partir duquel on peut considérer que le noeud de l'arbre est opaque. Dès lors, les noeuds de l'arbre sont classés en deux catégories : occultant, et non occultant.

On parcourt ensuite l'arbre en profondeur, afin de trouver les noeuds de densité supérieure au seuil les plus proches du noeud racine.

Pour chacun de ces noeuds occultants (NO), on effectue un test d'occultation avec l'ensemble des noeuds de l'arbre, que nous appellerons noeuds potentiellement occultés (NPO) (cf. figure 72) :

- Si le NPO ne contient aucune particule, il peut être considéré comme caché, quelle que soit sa position par rapport au NO.
- Il en est de même si le NPO est déjà marqué comme occulté dans une précédente étape.
- Si le NPO appartient à la branche issue du NO, on effectue un test entre chaque fils du NO et le NPO. En effet, il est très dangeureux, avec notre système de test d'occultation par marche de Graham, de tester l'occultation d'une partie de l'objet par lui-même. Nous rappelons qu'une des hypothèses du test d'occultation est que les deux éléments sont disjoints. En pratique, cette opération permet bien de potentiellement éliminer une partie du NO par occultation.
- Si le NPO et le NO représentent le même noeud, et que ce dernier n'est pas une feuille, on effectue un test entre le NO et les fils du NPO, pour la même raison que nous avons évoquée dans le point précédent.
- Si le NPO est déjà marqué comme partiellement occulté, cela veut dire qu'une partie de la branche issue de lui-même est marquée comme occultée (au moins une feuille). On effectue donc un test entre le NO et les fils du NPO.
- Dans les autres cas, on effectue un test d'occultation direct entre le NO et le NPO, en construisant la silhouette du NO, et en testant les différents sommets du NPO. Si le NPO est effectivement caché par le NO, le NPO et l'ensemble de la branche de l'arbre issue de lui-même est marquée comme occultée. Si le NPO n'est que partiellement occulté, on effectue le test avec ses fils, s'il en a.

Notons que si un test d'occultation est effectué avec les fils du NPO, l'état du NPO est recalculé juste après.

11.3.2 *Note sur l'occultation multiple*

En théorie, cette méthode ne prend pas en compte le fait que plusieurs NO peuvent cacher ensemble un NPO sans pour autant le faire complètement individuellement.

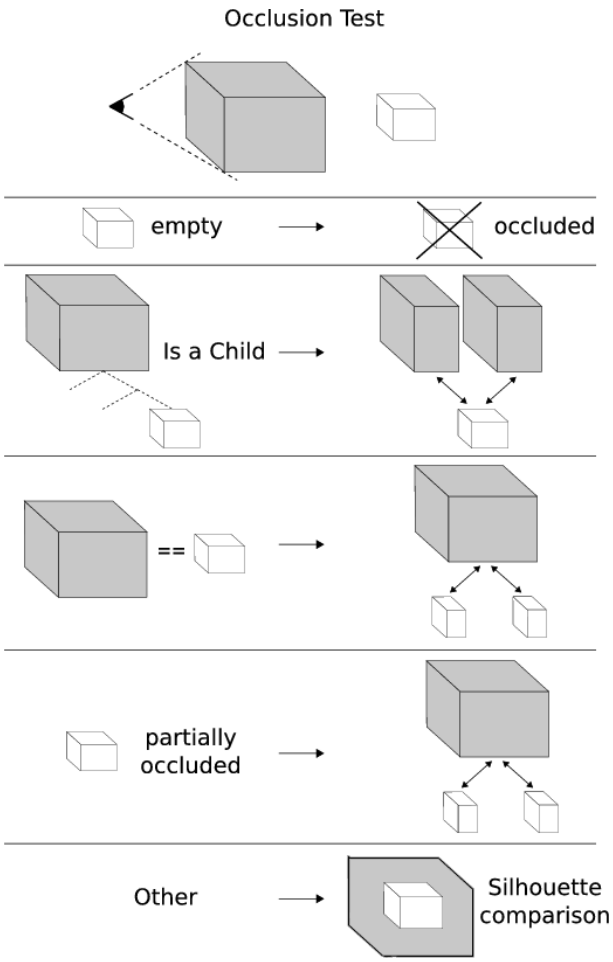


FIG. 72: Test d'occultation dans le Kd-tree

En pratique, cette perte d'exhaustivité permet un test d'occultation global beaucoup plus rapide, et est compensé par la profondeur de l'arbre. En effet, grâce au dernier point de la méthode, l'erreur commise ne se portera que sur l'occultation multiple des feuilles, et non des noeuds entiers. Tout dépend donc des dimensions des cellules représentées par les feuilles par rapport à la taille globale de la scène.

11.3.3 *Calcul de la densité des noeuds de l'arbre*

Comme nous l'avons vu plus haut, le calcul de la densité par méthode de Monte-Carlo n'est pas immédiat. Il serait donc très coûteux de l'effectuer sur l'ensemble des noeuds de l'arbre.

Nous avons donc essayé de trouver une relation entre la densité d'un noeud et celle de ses fils, afin de ne réellement calculer que la densité des feuilles.

Reprenons donc la méthode de Monte-Carlo, en considérant que la cellule est composée de deux parties, représentant les deux fils (cf. figure 73).

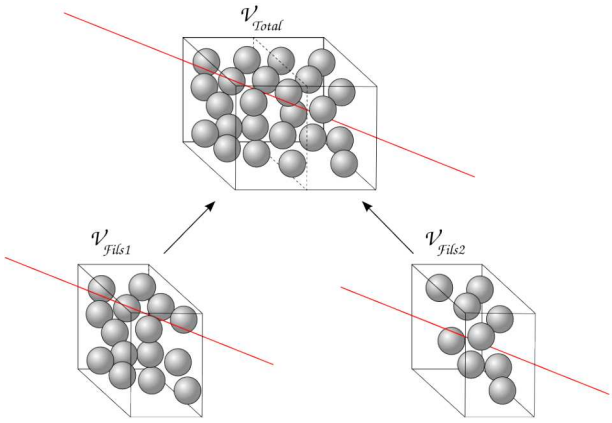


FIG. 73: Décomposition de la méthode de Monte-Carlo pour les fils

En prenant V_{Fils1}, V_{Fils2} et V_{Total} respectivement le volume du Fils 1, celui du Fils 2, et le volume total de la cellule, chaque rayon qui traverse la cellule entière a une probabilité $\frac{V_{Fils1}}{V_{Total}}$ de traverser le Fils 1, et $\frac{V_{Fils2}}{V_{Total}}$ de traverser le Fils 2. On sait que la densité est la probabilité que le rayon a de ne pas être arrêté. On a alors la relation suivante :

$$d \geqslant \frac{V_{Fils1}}{V_{Total}} d1 + \frac{V_{Fils2}}{V_{Total}} d2$$

avec d1, d2 et d respectivement les densités du Fils 1, du Fils 2, et de la cellule mère.

Cette formule permet tout d'abord de montrer que l'opacité est une caractéristique conservative : si $d1 \geqslant d_{seuil}$ et $d2 \geqslant d_{seuil}$, alors $d \geqslant d_{seuil}$.

Ensuite, cette formule nous donne une estimation basse de la densité de la cellule mère, qui peut être utilisée pour calculer très rapidement les différentes valeurs associées aux noeuds de l'arbre. Il suffit en effet d'utiliser la méthode de Monte-Carlo sur les feuilles de l'abre, puis, pour chaque noeud père, utiliser la formule donnée, et ce jusqu'à remonter au noeud racine.

Cette méthode permet de réduire sensiblement le temps de génération de l'arbre : le calcul de l'estimation est presque immédiat par rapport au calcul par méthode de Monte-Carlo, donc le temps de calcul est globalement divisé par deux. Rappelons que pour une profondeur de n, un arbre binaire équilibré possède 2^n feuilles, et $2^{n+1} - 1$ noeuds au total.

11.3.4 *Le frustum culling*

Si nous n'avons abordé que l'*occlusion culling* dans cette partie, c'est tout simplement parce que le frustum culling est extrêmement facile à intégrer à une telle méthode.

En effet, lors du test de comparaison des sommets d'un NPO avec la silhouette d'un NO, les coordonnées sont calculées en espace image . Tout ce qui est visible possède donc des coordonnées comprises entre -1 et 1.

Donc si pour une coordonnée donnée (x, y ou z), tous les sommets d'un NPO ont une valeur soit inférieure à -1, soit supérieure à +1, le NPO est en dehors du frustum de vision de l'observateur. Il est donc marqué comme occulté.

Le frustum culling est particulièrement efficace pour l'exploration de très grandes scènes, ou lors de scénarios au cours desquels l'observateur se rapproche virtuellement du jeu de données (zoom).

11.4 CONCLUSION

Ce chapitre nous a permis de décrire les techniques que nous avons conçues et utilisées pour trier en prétraitement les données à visualiser, et pouvoir ainsi évaluer lors de la phase de rendu quelles sont les parties à réellement afficher.

La génération du Kd-Tree détectant les espaces vides, couplée au calcul de densité des cellules et aux techniques d'occultation nous permettront de fortement réduire le nombre de particules qui feront réellement partie du flux de rendu.

12 PARALLÉLISME

Ce chapitre présente les choix d'architecture de rendu parallèle qui ont été faits lors de cette étude, ainsi que les solutions qui ont été apportées pour y intégrer les optimisations et méthodes présentées dans les chapitres 10 et 11 de cette partie.

L'ensemble peut se définir comme une solution hybride *sort-first/sort-last* conçue pour la visualisation de systèmes particulaires denses.

12.1 CHOIX D'ARCHITECTURE : ICE-T ET SORT-FIRST

12.1.1 *Pourquoi Ice-T ?*

Ice-T est la solution que nous avons adoptée pour la partie sort-last de notre système de visualisation particulaire.

Malgré ses limitations, en particulier son problème de consommation de débit réseau, nous pensons qu'il est plus adapté de se baser sur une architecture de sort-last, qui offre une répartition naturelle des charges efficace, lorsque l'on utilise un cluster de visualisation comportant plus de noeuds de rendu que de noeuds d'affichage.

De plus, la nécessité d'utiliser un mur d'images comme système d'affichage limite fortement les possibilités de solution sort-last. A notre connaissance, il n'existe pas de solution efficace autre qu'Ice-T adaptée à cette configuration. Un autre choix possible aurait été Chromium[HHN+]. Malheureusement, son grand point fort, à savoir sa généricité, est aussi son principal défaut : les performances en matière de rafraîchissement sont loin d'atteindre celles d'Ice-T[MT03] pour le rendu de type sort-last.

12.1.2 *Pourquoi une solution hybride ?*

L'idée d'une architecture hybride *sort-first/sort-last* n'est pas nouvelle.

Samanta et al.[SFLS00] propose une telle architecture pour des objets structurés représentés par des facettes (*vertex*), représentée en figure 74.

L'idée est d'effectuer une opération de *sort-first* afin de grouper les objets qui sont "visuellement proches" les uns des autres, c'est-à-dire proche après la projection en espace image. Ces groupes sont alors répartis équitablement (en nombre de facettes) sur les noeuds de rendu, puis un algorithme de type *sort-last* est réalisé pour rendre l'image finale (cf. figure 75).

Ce tri permet principalement de réduire le coût des communications du rendu grandes structures décomposées en éléments visuellement "cohérents" (principalement convexes). Samanta montre que le point

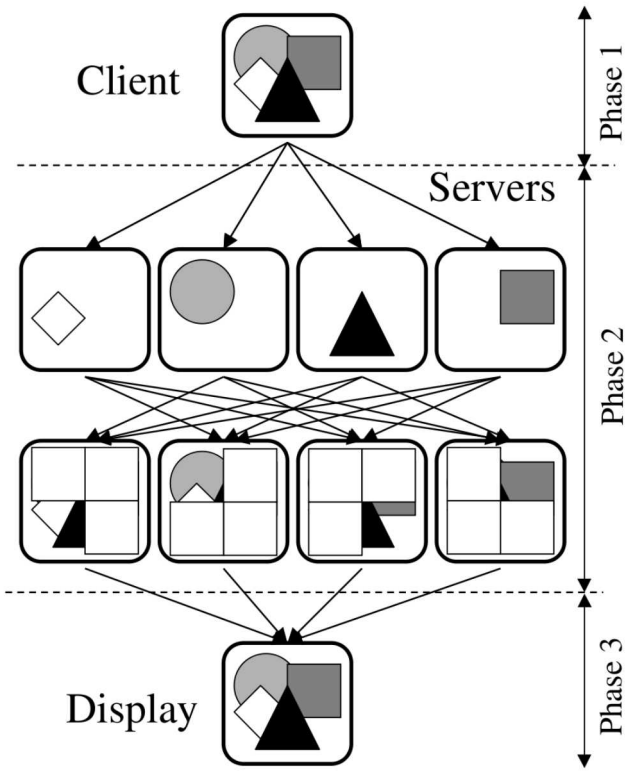


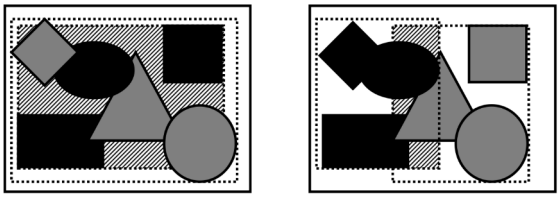
FIG. 74: Architecture hybride de Samanta et al.[SFLSoo] : architecture globale

fort de cette architecture est son comportement vis-à-vis de la montée en charge du système utilisé : que ce soit en augmentant le nombre de noeuds de rendu, en augmentant la granularité des données, ou en augmentant la définition de l'affichage, cette solution tire un bien meilleur parti du matériel que des solutions non hybrides (uniquement *sort-first* ou *sort-last*.

La difficulté de l'utilisation d'un tel algorithme pour la visualisation de systèmes particulaires denses est bien évidemment le manque de structure intrinsèque des données.

Heureusement, l'utilisation des HOMs décrite dans le chapitre 11 permet de faire apparaître une forme de structure qui permet alors de s'inspirer d'une tel système pour proposer un squelette d'architecture hybride optimisée pour la problématique de cette étude.

Ainsi la répartition des données, décrite en section 12.2.2, peut être considérée comme une forme d'adaptation du groupement spatial de Samanta dans le cas de l'utilisation du Kd-Tree décrit au chapitre 11. Le groupement ne s'effectue plus selon une projection 2D, mais selon la proximité spatiale 3D dûe à l'indexation en Z-order.



(a) Random grouping. (b) Spatial grouping.

FIG. 75: Architecture hybride de Samanta et al.[SFLSoo] : groupement spatial

12.2 COUPLAGE D'UN KD-TREE AVEC ICE-T POUR LA VISUALISATION DE MATIÈRE DENSE

Dans les chapitres précédents, nous avons décrit les optimisations de rendu et de *sort-first* que nous pouvions utiliser pour optimiser l'architecture Ice-T pour des systèmes particulaires denses.

Nous allons maintenant aborder la façon dont ces méthodes s'intègrent à l'architecture parallèle.

12.2.1 *Architecture*

Considérons N noeuds d'un cluster graphique, dont P sont réellement destinés à l'affichage. Sur chaque noeud tourne un processus MPI.

Chaque processus possède une copie complète de l'arbre en mémoire, mais une partie seulement lui est attribuée pour le rendu. Les données liées à cette partie sont donc aussi chargées en mémoire.

Lors de la phase de rendu de la scène partielle, le processus effectue le test d'occultation précédemment décrit pour la partie de l'arbre qui lui est attribuée. Puis l'ensemble des processus échangent leurs résultats et les fusionnent. Ils ont alors un arbre complet et à jour. Chaque noeud sait donc quelles sont les données qui doivent réellement être affichées.

Ils réattribuent en conséquence la distribution des données par noeud (cf. la section suivante), en veillant à ce qu'elle soit la plus équilibrée possible.

Les processus chargent alors les données manquantes en mémoire, et déchargent celles dont ils n'ont plus besoin si nécessaire. Vient ensuite le rendu de ces données.

Enfin, les images rendues par les N processus sont utilisées par Ice-T pour être affichées sur les P noeuds destinés à cette fonction.

Une représentation de cette architecture est proposée en figure 76.

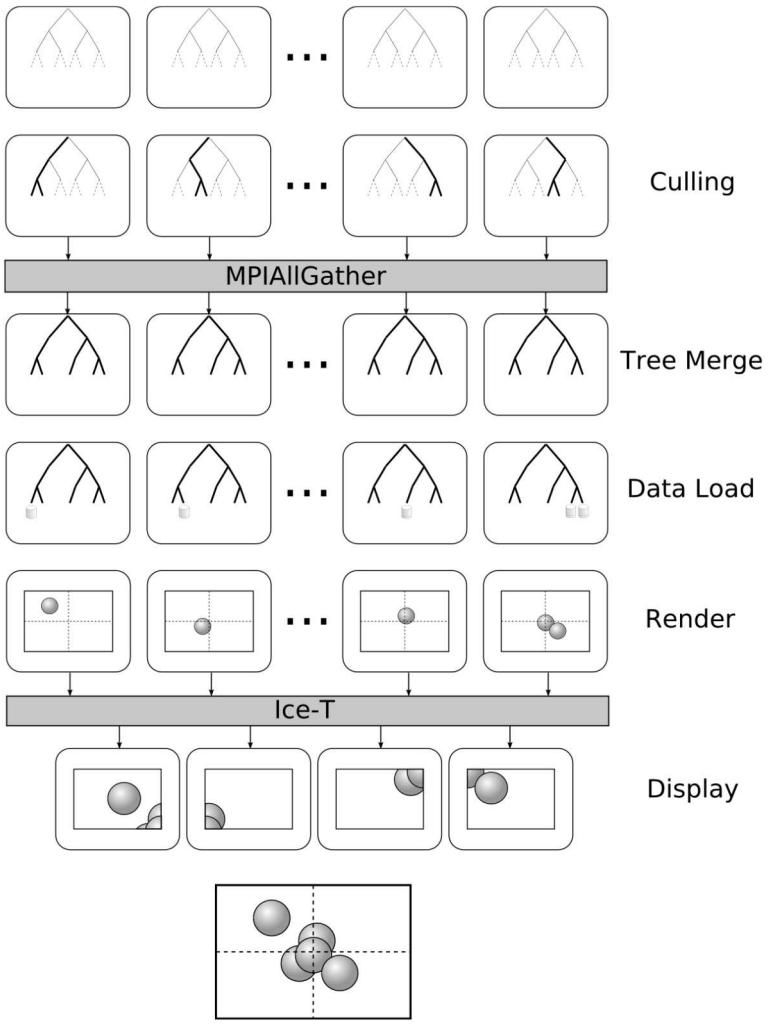


FIG. 76: Architecture globale de la solution

12.2.2 *Répartition géométrique des données et Z-order*

Afin de réduire la taille effective des images des scènes partielles rendues , il est important que le groupe de données attribuées à chaque processus soit le plus compact possible.

Le Z-order est une méthode de numérotation qui intègre des informations de hiérarchie dans la numérotation. Elle est extrêmement performante pour l'organisation des données de problèmes out-of-core[Pas00], et est utilisée dans Atomsviewer[Sha02] pour la numérotation des noeuds de leur octree.

Pour un arbre binaire de partitionnement d'espace, le principe est le suivant : la numérotation s'effectue en binaire. A chaque subdivision de l'espace, chaque fils prend le numéro du noeud père, puis le décale d'un bit vers la gauche, et ajoute 0 ou 1 suivant s'il est le fils gauche ou droit (cf. figure 77).

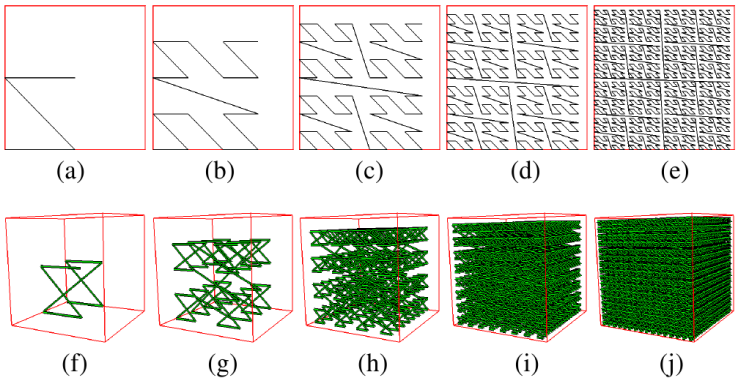


FIG. 77: Numérotation en Z-order en 2D et 3D.

Des numéros consécutifs, avec un même nombre de bits (donc la même profondeur dans l'arbre), correspondent donc à des sous-espaces proches.

La répartition que nous avons choisi d'utiliser se fonde grandement sur ce phénomène. Les noeuds de l'arbre sont donc numérotés selon cette méthode.

Chaque processus a déjà généré l'arbre complet et connait donc les parties occludées, et celles qui sont visibles.

Il calcule alors le nombre moyen de particules visibles par noeud (NMPV).

Chaque processus assigne ensuite les premières feuilles de l'arbre au premier processus, en s'arrêtant lorsque le nombre de particules visibles effectivement assignées est le plus proche possible du NMPV. Les suivantes sont assignées au second processus, toujours en faisant coïncider le plus possible les particules visibles avec le NMPV. Et ainsi de suite, jusqu'au dernier processus, à qui est assigné le reste des particules (cf. figure 78).

Afin de réduire l'erreur commise sur le nombre de particules par noeud (due à la granularité des feuilles), la condition d'arrêt pour chaque assignation à un processus alterne entre juste inférieure, et juste supérieure au NMPV.

Chaque processus possède donc une partie d'arbre à rendre, représentée par l'ensemble des feuilles qui lui sont assignées.

Cette stratégie, relativement simple, possède un triple avantage : elle est spatialement cohérente, et relativement bien équilibrée, et peut être calculée indépendamment des autres processus, du moment que les résultats de l'occultation sont partagés.

De plus, le fait que l'on assigne les feuilles de l'arbre aux processus dans le même ordre permet au système de cache par VBOs (cf. le chapitre 4) d'être beaucoup plus efficace. Supposons par exemple que l'ensemble de la mémoire de toutes les cartes vidéos utilisées lors du rendu peut contenir plus que les données visibles. Les données des feuilles qui précédemment étaient visibles, puis ont été cachées, et qui maintenant réapparaissent, sont encore en mémoire graphique. Il

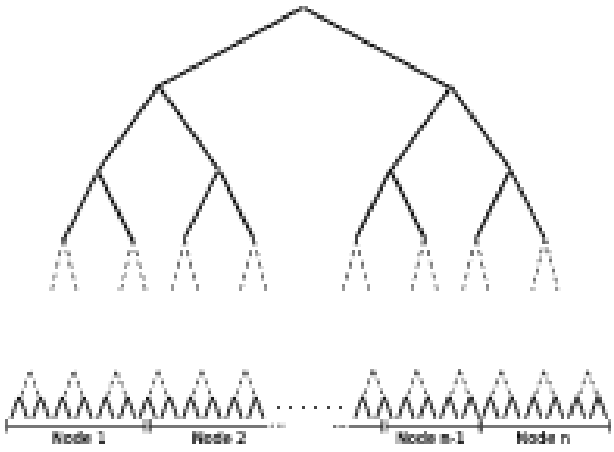


FIG. 78: Répartition de l'arbre pour les différents processus

n'est donc pas nécessaire de les renvoyer au GPU pour les rendre. Ce phénomène se produit fréquemment lors des phases d'exploration : la caméra tourne autour de la scène, afin d'obtenir une appréciation visuelle globale de la simulation.

12.2.3 *Répartition et recouvrement*

Il est rare qu'un algorithme puisse être complètement parallélisable, c'est-à-dire découpable en tâches indépendantes les unes des autres. L'un des principaux problèmes des architectures parallèles est donc de pouvoir répartir le plus possible les calculs tout en minimisant les communications entre les différents noeuds, et les barrières qui sont mises en place pour assurer la cohérence des informations manipulées. L'enjeu est d'utiliser le maximum de ressources en même temps, et donc de ne pas laisser de noeuds inactifs pendant que d'autres font de longs calculs.

Cas général

Assarsson propose des stratégies générales de gestion de liste de tâches dynamiques[AS01] :

LE GLOBAL TASK QUEUE : un processus possède la liste des tâches à faire, et chaque processus vient l'interroger pour savoir ce qu'il a à faire. Il s'agit de la stratégie la plus simple, mais qui comporte beaucoup de barrières et de communications.

LE GLOBAL COUNTER SCHEME : chaque processus possède la liste de tâches globale, et un compteur (un entier) global est incrémenté par tous les processuss pour informer les autres processuss de leurs actions. Cela réduit fortement le volume des communications, mais il reste néanmoins un nombre important de barrières, en particulier lors de la prise de possession du compteur par un processus : pour des raisons de cohérence et

de validité de la valeur du compteur, un seul processus doit y avoir accès en écriture.

LE LOCK-FREE TASK QUEUE : il s'agit d'enlever toutes les barrières, en évitant des accès concurrents aux listes de tâches. Chaque processus possède une liste de tâches locale, et une sous-liste par processus voisin. Les sous-listes sont créées de telle façon qu'il est possible de rajouter ou enlever des éléments en même temps (il suffit pour cela d'utiliser des index de début et fin de liste). Lors du parcours des tâches, le processus vérifie si les sous-listes des processus voisins sont peu remplies, et les remplit donc avec une partie de ses propres tâches. Sinon, il s'occupe lui-même de les traiter.

La dernière stratégie offre bien sûr un meilleur recouvrement, mais il faut pour cela des interconnexions très rapides entre les noeuds.

Cas avec information

Ces stratégies sont réellement performantes si la répartition de charge n'est pas connue à l'avance.

Or, dans le cas qui nous intéresse, chaque processus possède, une fois les résultats des tests d'occultation récupérés, une vue d'ensemble des données à traiter. On se retrouve alors dans un cas de *global counter scheme*, mais il n'est plus nécessaire aux processus d'avoir accès à un compteur.

Le choix d'une telle méthode se base néanmoins sur deux facteurs :

- le recouvrement lors des tests d'occultation doit être efficace
- l'envoi et la fusion des résultats doit être rapide, car sinon cela reviendrait à bloquer l'ensemble des processus pendant ce long laps de temps. On serait alors confronté à un goulet similaire à celui du *global counter scheme*.

Recouvrement et tests d'occultation

Comme décrit plus haut, chaque processus effectue les tests d'occultation pour la partie de l'arbre qu'il a rendue pour l'image précédente.

Les parties de l'arbre qui nécessitent le plus de temps de calcul pour l'occultation sont les parties qui ne sont pas cachées : en effet, il suffit d'un seul élément occultant la partie pour qu'elle soit considérée comme cachée. Si tel est le cas, plus aucun calcul la concernant ne sera effectué.

Nous ne pouvons pas savoir a priori quelles parties de l'arbre ne seront pas visibles. Néanmoins, nous pouvons supposer qu'il existe la plupart du temps une cohérence entre deux images consécutives lors de la visualisation de la scène : à moins d'un changement brutal du point de vue de l'observateur, il y a de fortes chances pour que ce qui était caché le soit encore.

La répartition des données à rendre étant équilibrée pour l'image précédente, elle a donc de très fortes chances de l'être pour les tests d'occultation de l'image suivante. Toutefois, si tel n'est pas le cas, la répartition sera de toute manière corrigée juste avant le rendu.

Envoi et fusion des tests d'occultation

Afin d'accélérer nettement l'envoi des tests d'occultation, les états des feuilles sont stockés dans une "signature" de l'arbre, à savoir un vecteur de $\frac{Nombredefeuilles}{4}$ octets, puisque chaque état peut être encodé sur deux bits. Chaque processus envoie donc 2^{p-2} octets, ce qui par exemple, pour une profondeur de 15 (celle que nous avons utilisé pour les tests), donne 8192 octets, ce qui est assez faible, puisqu'en-dessous du MTU pour une interface en Ethernet gigabit.

Le MTU (Maximum Transmission Unit) est la quantité maximale de données fixée lors d'une connexion TCP pouvant être transmise sans être fragmentée. Elle est en général de 1500 octets pour une interface ethernet 100Mbits, mais la plupart des appareils réseaux en Gigabit ethernet supportent 9000 octets.

Comme nous l'avons vu plus haut, le calcul des états intermédiaires de l'arbre est extrêmement rapide si l'on utilise l'encodage ci-dessous :

Occulté	Partiellement	Non Occulté
01	11	10

Etat du noeud = Etat du Fils gauche|Etat du Fils droit

Il reste donc à élaborer une méthode de fusion rapide entre les différents buffers.

Les deux opérations possibles lors de la fusion des résultats, pour un seul état donné, sont :

01(occulté) et 10(non occulté) donne 01(occulté)
10(non occulté) et 10(non occulté) donne 10(non occulté)

Si a et b sont respectivement l'état 1 et 2 à fusionner, on peut effectuer l'opération binaire suivante :

$$a \oplus b \oplus 10$$

$\oplus$ étant l'opération XOR (ou exclusif).

Les opérations binaires s'effectuant en général par octet, on a alors :

$$A \oplus B \oplus 10101010$$

soit en décimal :

$$A \oplus B \oplus 170$$

Ces opérations binaires consomment très peu de cycles CPU, la fusion est donc extrêmement rapide.

12.3 CONCLUSION

L'architecture hybride *sort-first/sort-last* proposée dans ce chapitre, basée sur la hiérarchisation des données présentée au chapitre précédent, sur une répartition équilibrée en Z-order des feuilles du Kd-Tree, ainsi que sur les optimisations de la solution *sort-last* Ice-T, permet de proposer une solution complète, cohérente et optimisée pour la visualisation de systèmes particulaires denses.

Voyons-en maintenant l'implémentation et les résultats.

13 IMPLÉMENTATION ET RÉSULTATS

Ce chapitre présente une implémentation de la solution hybride *sort-first/sort-last* présentée au chapitre précédent, et les tests effectués sur un mur d'images piloté par un petit cluster de visualisation (8 noeuds).

Sont également décrites les stratégies d'implémentations qui ont été essayées afin d'optimiser le transfert de données de la mémoire centrale de chaque machine de rendu vers leur mémoire graphique.

13.1 LA PLATEFORME

13.1.1 *Logiciels*

Ice-T est déjà intégré dans le logiciel Paraview, basé sur VTK[VTK]. Ce dernier est une bibliothèque générique de visualisation scientifique, écrite en C++, et "open-source".

Nous avons donc décidé de nous baser sur VTK pour implémenter la solution de visualisation particulaire pour matériau dense.

Nous avons extrait de Paraview les classes permettant d'utiliser Ice-T, et nous les avons utilisées telles quelles.

En ce qui concerne la partie rendu, des échanges avec le Centre Suisse de Calcul Scientifique (CSCS), par l'intermédiaire de Jean Favre et John Biddiscombe, avait permis de mettre au point une classe *vtkPointSpriteMapper*[Spa] implémentant les différentes méthodes OpenGL précédemment citées.

Néanmoins, il a fallu considérablement remanier cette classe, en particulier pour y intégrer la manipulation des VBOs. C'est pourquoi la solution utilise une classe fille, *vtkVBOPointSpriteMapper*, créée pour les besoins spécifiques de cette étude.

13.1.2 *Matériel*

Les tests ont été effectués sur :

- un mur d'images de 4 blocs, permettant un affichage de résolution 2048x1536,
- un petit cluster de 8 noeuds Bi-Xeon 3.4Ghz avec une carte NVidia Quadro FX 4500, le tout connecté à un réseau Gigabit Ethernet.

13.2 ETUDE DE PERFORMANCE DU KD-TREE POUR MATIÈRE DENSE AVEC ICE-T

13.2.1 *Le cas test et les scénarios*

Le jeu de données avec lequel nous avons effectué les tests est composé de 32 millions de particules. Il s'agit d'un cas typique de système moléculaire utilisé lors de simulations en détonique : un cylindre de particules en expansion (cf. figure 79).

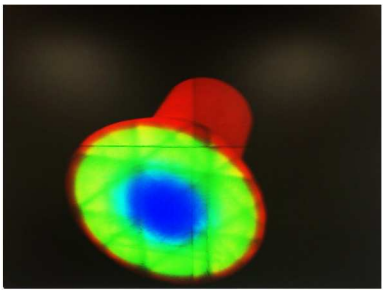


FIG. 79: Jeu de données de 32 millions de particules

80% de la scène est constituée de matière dense (la partie cylindrique), mais cette densité diminue fortement dans la partie conique de la scène (la partie en expansion).

Cette solution étant conçue pour l'exploration de la simulation, nous avons choisi deux scénarios typiquement utilisés lors de cette phase :

- soit la caméra effectue une rotation, à longue ou courte distance, autour du centre de la scène,
- soit la caméra se rapproche d'une zone d'intérêt de la scène (zoom).

13.2.2 *Les différentes implémentations*

Nous avons testé 3 méthodes différentes de *sort-first* :

- une méthode témoin, sans ordre particulier des données. Chaque processus reçoit un même nombre de particules à afficher, et le tout est stocké dans un VBO unique dans leur carte graphique respective. Il n'y a donc plus de transfert de données de la scène au GPU, excepté à l'initialisation.
- une méthode avec répartition en Z-order des données, mais sans occultation. Là encore, les données sont envoyées une seule fois à la carte graphique.
- la méthode complète, avec le système d'occultation.

13.2.3 *Le système de cache*

Comme nous l'avons vu dans le chapitre 11, les VBOs permettent de créer un système de cache afin de gérer la mémoire de la carte graphique.

Nous allons décrire les essais que nous avons effectués lors de cette étude.

Premier essai : un seul VBO

Nous avons d'abord utilisé un seul et unique VBO pour gérer la mémoire graphique. Le principe était le même que pour un système de fichier simple : on fixe la taille d'un bloc, pouvant contenir un nombre donné de particules. Ce bloc devient l'unité de base du système : le nombre de particules des feuilles se compte en blocs, et le VBO possède une taille de blocs entiers donnée. On crée alors la table des blocs chargés dans le VBO, ainsi que leur emplacement (cf. figure 80).

A chaque rendu, s'il y a modification des blocs à afficher, on efface les blocs qui ne sont plus visibles (s'il n'y a plus de mémoire GPU), puis on charge les nouveaux blocs dans les emplacements libres.

Si nous nous sommes d'abord intéressés à cette méthode, c'est parce que cela permettait de réduire considérablement le nombre de changements d'état de la machine OpenGL par rapport à une méthode avec plusieurs VBOs.

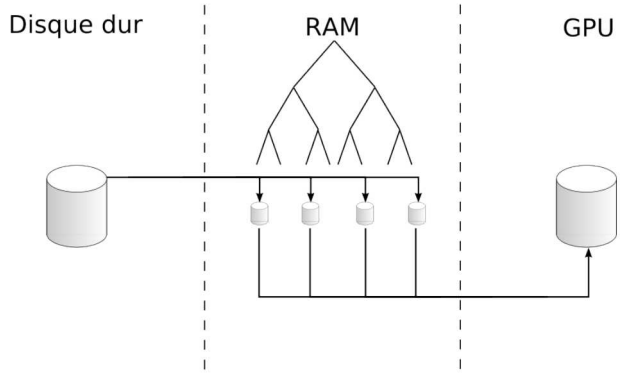


FIG. 80: Cache avec un seul VBO

Malheureusement, les résultats sont catastrophiques : le rafraîchissement est plus de 10 fois plus lent que la méthode sans occultation.

Le problème du transfert partiel

Après quelques tests, nous avons donc suspecté que ce ralentissement venait essentiellement du transfert de données partielles dans le VBO (par glBufferSubData).

Nous avons donc testé une architecture à mi-chemin entre la méthode sans occultation et celle avec : un grand buffer est recréé en mémoire centrale pour être renvoyé en une seule passe à la carte graphique (cf. figure 81).

Ce fut avec surprise que nous avons constaté que les performances de cette architecture étaient très proches de la solution sans occultation (donc sans transfert à la carte graphique). La réduction du temps de

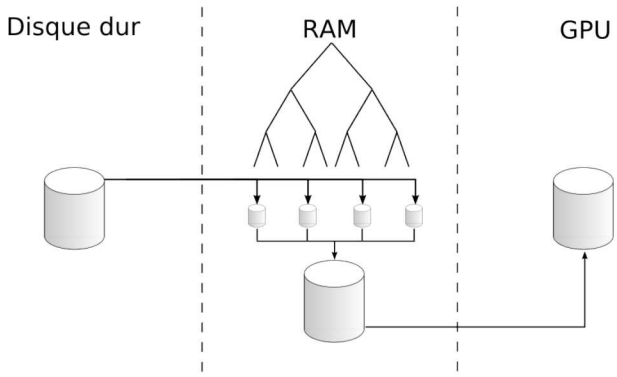


FIG. 81: Cache redondant

rendu due à l'occultation compensait à peu près le temps de transfert du buffer.

Cela a donc confirmé nos doutes : le transfert partiel était à bannir de toute utilisation d'un VBO.

La solution : scinder le VBO

Nous avons donc modifié l'architecture de cache, en créant un VBO à chaque fois qu'une feuille de l'arbre chargeait ses données en mémoire centrale.

A chaque image, seuls les VBOs associés aux feuilles visibles sont rendus. S'il y a modification des feuilles à afficher, les données sont stockées en mémoire centrale, puis transférées VBO par VBO en mémoire graphique. Dans le cas où la mémoire libre respectivement centrale et graphique est insuffisante, les VBOs ou les buffers des feuilles non visibles sont respectivement effacés (cf. figure 82).

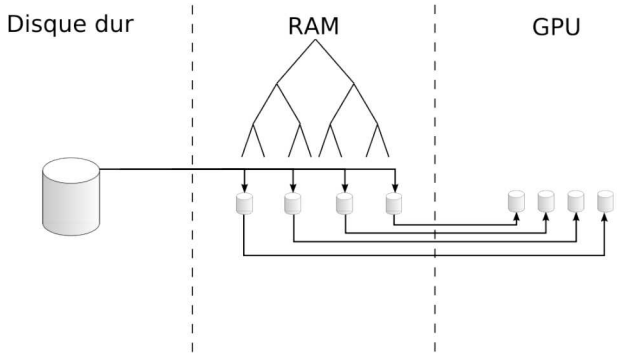


FIG. 82: Cache avec VBO scindé

Les résultats ci-dessous utilisent cette architecture pour la méthode occlusive.

13.2.4 *Résultats en terme de rafraîchissement*

Les figures ci-dessous présentent les resultats en terme de rafraîchissement pour les deux scénarios testés.

Afin de vérifier que la solution était bien évolutive en fonction du nombre de noeuds de rendus qu'elle utilisait, nous avons recueilli les résultats pour 4 noeuds, puis 8 noeuds de rendu.

Rotation autour de la scène

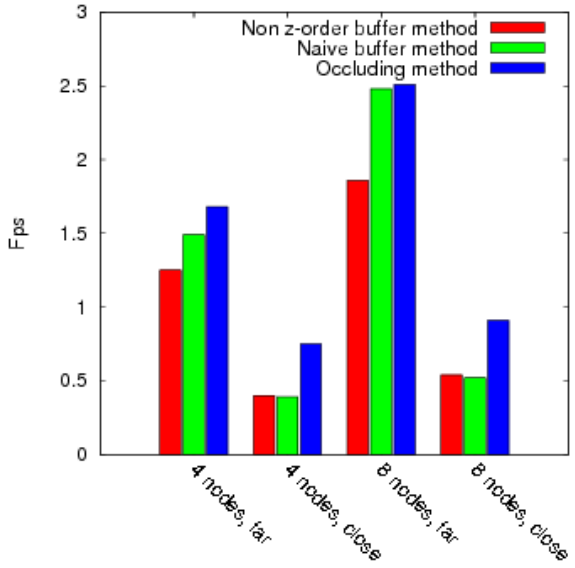


FIG. 83: Rafraîchissement moyen pour le scénario de rotation : les résultats en rouge correspondent à l'architecture parallèle proposée mais sans Z-order et sans occultation, ceux en vert avec Z-order mais sans occultation, et ceux en bleu à l'architecture complète.

Les résultats pour la rotation (cf. figure 83) montrent l'importance de la répartition en Z-order : dans le cas où l'observateur est loin de la scène, les différents rendus partiels de chaque processus ne prennent qu'une petite partie de l'image si les données sont proches géométriquement. Les floating viewports sont donc plus petits, donc les performances sont au rendez-vous. On peut aussi remarquer que cela n'est pas vrai pour une rotation très proche de la scène : les rendus partiels de la scène prennent de toute manière toute l'image, les floating viewports coïncident avec l'image totale, donc la répartition en Z-order n'apporte rien par rapport à une autre répartition.

On peut aussi remarquer que pour une rotation éloignée de la scène avec 8 noeuds de rendu, la méthode avec z-order mais sans occultation est presque aussi performante que celle avec. Cela vient de la limite de saturation en débit du réseau, du fait que la résolution

du mur est relativement faible pour 8 noeuds : chaque processus a peu de particules à afficher, et une image partielle de la scène avec peu d'information très concentrée. Les transferts sont donc relativement les mêmes (avec ou sans occultation). Pour atteindre de meillleures performances dans ce cas, il faudrait principalement s'attaquer aux problèmes de latence, en passant par exemple à un réseau infiniband.

Zoom sur une zone d'intérêt

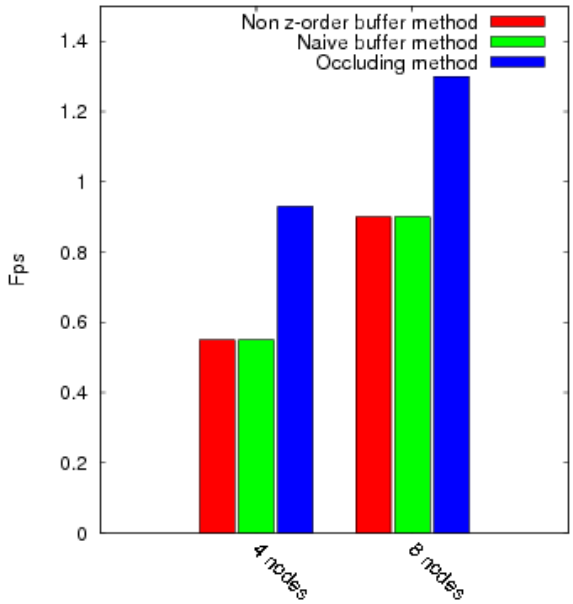


FIG. 84: Rafraîchissement moyen pour le scénario de zoom

Les résultats en matière de zoom (cf. figure 84) font apparaître une nette différence suivant l'utilisation ou non de l'occultation.

On retrouve le même phénomène de faible écart entre les performances du Z-order et de la répartition sans ordre que pour une rotation proche de la scène. Là encore, les floating viewport coïncident avec l'image complète.

Ce scénario met en évidence la différence de temps de rendu selon qu'il y a ou non occultation.

La forte augmentation du taux de rafraîchissement à la fin du zoom, représenté en figure 85, est due à la mise en route de l'algorithme de frustum culling, une partie de la scène sortant du champ de vision de l'observateur.

13.2.5 *Résultats en terme d'occultation*

L'algorithme d'occultation permet de réduire de 30% à 50% le nombre de particules réellement rendues dans le cas du scénario de rotation.

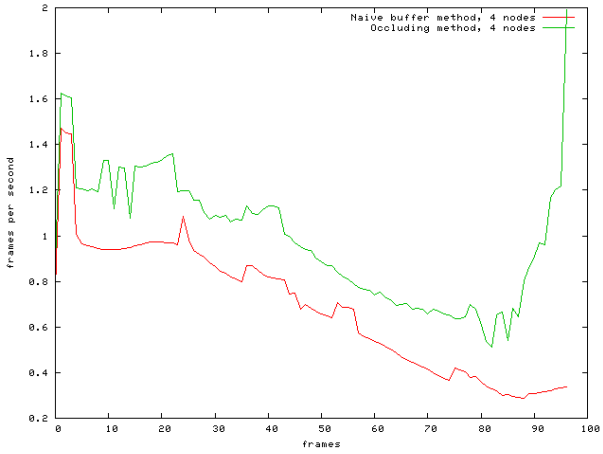


FIG. 85: Rafraîchissement moyen pour le scénario de zoom : variations au cours du temps

La figure 86 permet de s'apercevoir que cette stratégie remplit parfaitement son rôle pour le scénario de zoom.

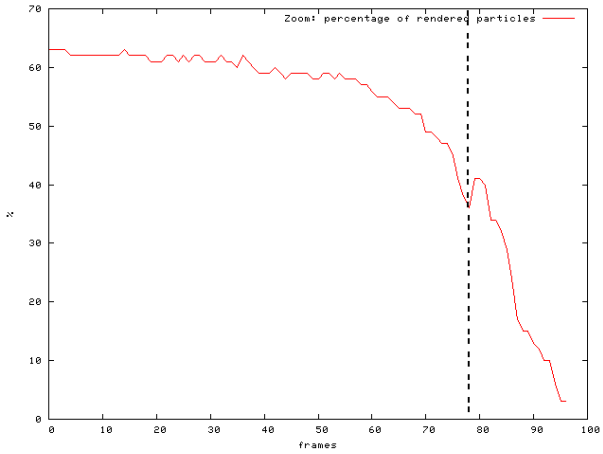


FIG. 86: Pourcentage de particules réellement rendues pendant le scénario de zoom

La séparation indique le moment où le frustum culling intervient. La baisse progressive du pourcentage de particules rendues est relativement simple à expliquer : plus l'observateur se rapproche de la scène, plus les parties occultantes prennent de place sur l'image, donc plus elles cachent de parties qui se trouvent derrière elles.

13.2.6 *Impact sur la qualité de l'image*

Seule la méthode d'occultation influe sur la qualité de l'image finale. Cette dernière dépend donc entièrement du seuil de densité fixé a priori.

Pour les tests précédemment cités, le seuil que nous avons fixé est de 0,95. Pour cette valeur, la différence entre les images avec et sans occultation est extrêmement faible (de l'ordre d'un pixel).

Il est intéressant de noter que la variation de qualité en fonction du seuil n'est en rien linéaire. Elle est en fait proche d'une fonction en escalier.

Il faut en effet rappeler que la densité est une probabilité moyenne pour un rayon de traverser une cellule. Si cette probabilité est faible, cela veut dire que la cellule est partiellement vide. Si on choisit un seuil faible, la solution considèrera que de telles cellules sont opaques. On verra alors apparaître des erreurs très importantes, comme par exemple des "trous" dans la scène (cf. figure 87).

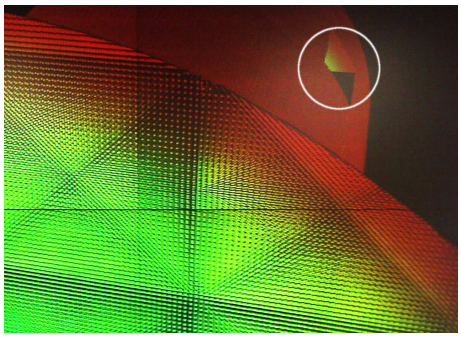


FIG. 87: Artefacts dus à un seuil de densité faible

Il est donc très important de garder un seuil relativement élevé. Il est conseillé de ne pas descendre en dessous de 0,95 , sauf dans des scènes très particulières (répartition très homogènes des particules, par exemple).

14 CONCLUSION ET PERSPECTIVES

Les résultats du chapitre 13 montrent que la solution hybride *sort-first/sort-last*, avec des optimisations de rendu de sphères par l'utilisation de shaders et de VBOs, offre des perspectives prometteuses pour l'exploration de systèmes de particules denses en Physique des Matériaux.

Bien que les taux de rafraîchissement obtenus ne puissent être considérés comme suffisants pour une exploration confortable du jeu de 32 millions de particules, il faut bien garder à l'esprit que le cluster utilisé est loin d'être dimensionné pour la visualisation de telles scènes :

- il est composé de peu de noeuds,
- les cartes graphiques sont basées sur un chipset G70, où les unités de calcul de fragment et de vertex shaders sont séparées. Les nouvelles cartes graphiques possèdent une architecture de shader unifiée, ce qui permet une répartition de charge de calcul automatique,
- le réseau d'interconnexion gigabit ethernet est très vite saturé.

Il serait donc intéressant de pouvoir tester cette solution sur un cluster plus adapté à de tels cas (plus de 10 noeuds, avec un réseau Infiniband, et des cartes graphiques récentes).

Parmi les études possibles qui découleraient des résultats présentés, nous aimerions aborder celles-ci :

- Comme nous l'avions précédemment écrit, Nvidia propose désormais des architectures matérielles unifiées pour le calcul de shaders. Afin d'exploiter au maximum le potentiel de telles unités, la société fournit une bibliothèque nommée CUDA[CUD] permettant d'effectuer des calculs directement sur la carte graphique. Un tel potentiel pourrait être très intéressant, puisqu'un certain nombre de calculs de pré-traitement, comme celui de la densité des feuilles du Kd-Tree, pourraient être effectués par le GPU. On pourrait même imaginer, si les résultats sont particulièrement bons, effectuer ces calculs de façon dynamique (pendant le processus de rendu).
- Par cette même technique, il serait parfaitement possible d'effectuer des pré-traitements supplémentaires, comme l'Ambient Occlusion, qui comme nous l'avons décrit dans la section 2 du chapitre 4, permet une perception des formes et des volumes beaucoup plus pertinente qu'en l'absence d'ombrage global.
- Toute cette étude vise à l'exploration de systèmes de particules denses statiques (qui n'évoluent pas dans le temps). Il semble difficile que l'architecture proposée puisse, sans modification, permettre une exploration de l'ensemble des pas de temps d'une scène aussi complexe que celles proposées en Physique des

Matériaux. On pourrait envisager un pré-traitement complet des données, par exemple en mode non graphique (mode *batch*), afin de réaliser un film, pour l'exploration "globale" de l'ensemble du phénomène dans le temps, puis utiliser les ces calculs de pré-rendu pour une phase interactive qui permettrait d'inspecter des zones d'intérêt. Quoi qu'il en soit, des optimisations pourraient être possibles. La structuration de l'espace en Kd-Tree pourrait par exemple permettre de détecter les parties de la scène inchangées par rapport aux pas de temps précédents, ce qui limiterait le transfert des informations entre les mémoires principales (RAM et disque dur) et la mémoire graphique aux seules particules qui ont réellement évolué dans le temps. On pourrait également étudier la possibilité d'un pré-chargement partiel ou total des pas de temps proches de celui rendu pour optimiser les temps de transfert.

CONCLUSION

Nous vous proposons un rapide bilan de cette étude, sous la forme d'un jeu de questions/réponses.

QUESTION : QUEL ÉTAIT LE BUT DE CETTE ÉTUDE ?

Cette étude avait pour but de concevoir une solution informatique complète de visualisation scientifique pour la Physique des Matériaux. Elle devait être adaptée en particulier à l'exploration de systèmes denses à grand nombre de particules, en utilisant un système matériel permettant un environnement immersif simple.

QUESTION : CE DOCUMENT APPORTE-T-IL UNE SOLUTION COMPLÈTE À L'EXPLORATION DE DONNÉES PARTICULAIRES DENSES ?

Il apporte des éléments de solution.

Ce document apporte bien des techniques d'optimisations et des choix d'architectures adaptés aux besoins de cette étude. Il propose ainsi une implémentation originale d'un menu permettant d'augmenter l'interaction avec le système de visualisation lors de l'exploration et le réglage de celui-ci. Il fournit également un programme couplant différentes architectures de rendu et des optimisations OpenGL qui permet une interactivité correcte sur du matériel plutôt modeste. Chaque décision sur les algorithmes adoptés a aussi été dictée par leurs excellents comportements lors de la montée en charge des données et en puissance du matériel utilisé.

Cette étude avait pour but de valider une solution relativement simple, couplant interactivité et interaction, compatible avec du matériel parallèle et immersif relativement limité. Ce point est réalisé.

Les deux sous-systèmes permettant interaction et interactivité n'ont pas encore été couplés, et ceux malheureusement par manque de temps. Ils sont en revanche largement basés sur la même bibliothèque principale, VTK. Ils intègent en particulier la même gestion des événements (clavier, souris, rafraîchissement, etc). En résumé, le *FlowMenu3D* "pilote" la partie s'occupant de l'affichage de VTK via le même jeu d'instructions qu'utilise le système hybride interactif pour placer la caméra, récupérer le signal de rafraîchissement, etc. Le couplage des deux sous-systèmes est donc une tâche a priori relativement naturelle et sans réelle difficulté.

QUESTION : QUELS SONT LES RÉSULTATS INNOVANTS DE CETTE THÈSE ?

Cette thèse propose deux principaux résultats innovants :

– Une nouvelle solution générique de contrôle d'application pour la visualisation scientifique, par la conception et l'implémentation du *FlowMenu3D*,
– Une proposition d'architecture spécifique hybride *sort-first/sort-last*, optimisée pour la visualisation de systèmes particulaires denses.

QUESTION : LA SOLUTION PROPOSÉE VA-T-ELLE ÊTRE UTILISÉE ?

Nous l'espérons.

Les optimisations qui concernent le rendu de sphères, présentées au chapitre 10, sont déjà utilisées régulièrement par le laboratoire en Physique des Matériaux du CEA/DIF sous forme d'un petit logiciel foncionnant sur un seul poste (pas de système parallèle).

Le reste des solutions est destiné à passer de l'exploration sur un seul ordinateur, à celle utilisant un mur d'images et un cluster de rendu.

QUESTION : QUAND CETTE SOLUTION SERA-T-ELLE OBSOLÈTE ?

Une petite partie des optimisations de cette étude sont très liées au matériel utilisé pour le rendu et l'affichage en visualisation scientifique.

Il suffirait d'utiliser de nouvelles cartes graphiques, possédant un chipset de dernière génération, pour que le rapport entre le coût de communication entre noeuds et le temps de rendu des différentes parties de la scène, ne soit plus aussi bon. La différence de résultat entre la solution "naïve" et celle hybride serait moins flagrante. Les optimisations OpenGL pourraient aussi être rattrapées par les progrès matériels, le calcul sur carte graphique (GPGPU) étant de plus en plus demandé par le monde du calcul scientifique, ce qui pousse les constructeurs à faire rapidement évoluer cette branche.

Au niveau algorithmique, l'essentiel des techniques restent valables. Il pourrait même être intéressant de reprendre les solutions proposées ici en matière d'échanges de données avec le GPU et de les tester avec les prochaines architectures (en particulier l'OpenGL 3.0, qui propose une architecture beaucoup plus unifiée).

Nous rappelons que OpenGL est une norme qui est conçue et rédigée par un groupe indépendant, ce qui donne à cette API une perennité et une interopérabilité indéniable.

De plus, la partie de l'étude sur l'interaction et le contrôle d'application est probablement moins sujette à l'obsolescence, ce domaine ne suivant pas la loi de Moore, et travaillant en lien étroit avec les sciences liées à la perception, ou l'apprentissage humain.

QUESTION : QUAND SONT LES PERSPECTIVES DE CETTE THÈSE ?

Pour la partie interaction, il reste beaucoup d'hypothèses à vérifier et de tests à effectuer. En particulier, il serait réellement intéressant de faire utiliser le *FlowMenu3D* sur des plateformes de réalité virtuelle plus variées, telle qu'un mur d'image avec un Fastrak, un Workbench,

etc. Une étude sur l'intégration de la stéréoscopie et de l'haptique dans ce système peut aussi donner des résultats pertinents.

Pour la partie interactivité, les récents changement dans l'architecture des GPUs ouvrent des perspectives extrèmement intéressantes. Il serait particulièrement intéressant de tester l'architecture proposée sur des systèmes à base de PCI Express 2.0, et/ou sur un réseau rapide (Infiniband par exemple), etc. Il serait également intéressant de tester cette solution sur un système plus important, ce qui permettrait de vérifier son comportement lors d'un passage à l'échelle.

Enfin, il est probable que l'ensemble de la solution proposée, interaction et interactivité confondues, puisse être utilisée dans d'autres domaines de la visualisation scientifique. On peut citer comme exemple la visualisation des vecteurs, à base de glyphes, qui pourrait facilement adapter notre système à leurs besoins.

Quatrième partie

APPENDICE

APPENDICE

A.1 PROTOCOLES DE TEST POUR LE FLOWMENU3D

A.1.1 *Informations délivrées à l'utilisateur*

- il s'agit d'un test de menu dans un environnement 3D. L'Omni est une machine qui simule un "stylo virtuel".
- on lui explique le principe du menu / sous-menu : on garde le bouton constamment appuyé pour afficher le menu, et c'est en passant d'une zone à une autre qu'on sélectionne celle-ci, avec un retour au centre pour la sélection finale.
- la couleur : on va guider l'utilisateur en coloriant en vert la zone à sélectionner, et en coloriant en vert tout le menu pour indiquer un retour au centre. En pratique : au clic, affichage du menu et coloriage du 1er quartier, puis sélection d'un quartier, puis coloriage du deuxième quartier, puis sélection d'un deuxième quartier, puis coloriage de tout le menu, puis retour au centre. Et ce, même si ce sont de mauvais quartiers qui sont sélectionnés.

On explique qu'il faut sélectionner le premier quartier, puis le deuxième (si ce n'est pas le même), et enfin seulement revenir au centre.

A.1.2 *Les paramètres mesurés :*

- le temps de réponse pour la sélection (le temps que met l'utilisateur depuis le clic jusqu'au retour au centre).
- le nombre d'erreurs de sélection.

Ne sont pris en compte que les essais où l'utilisateur est revenu au centre après avoir sélectionné au moins un quartier.

A.1.3 *Les modifications selon les tests :*

- la forme du menu : on passe d'un menu en octogone à un menu en carré (différenciation des quartiers en coin et ceux aux centres des côtés). Entre 40 et 50 essais pour chaque forme, avec alternance entre quartiers en coin et quartiers "au milieu".
- le rapport entre la taille et la "sensibilité" du curseur : 4 tests, avec 50 essais environ par test

A.1.4 *Résultats attendus a priori :*

- la forme : probablement un temps de réponse plus rapide, et peut-être un peu moins d'erreurs.

– le rapport taille/sensibilité : on devrait avoir les mêmes résultats en conservant le rapport taille du menu/sensibilité, ce qui permettrait de changer la taille du menu, pour qu'il soit moins intrusif dans la scène.

APPENDICE

B

B.1 ARTICLE SOUMIS AU WSCG'09

L'article suivant a été soumis à la conférence WSCG'2009, *The 17th International Conference on Computer Graphics, Visualization and Computer Vision'2009*, qui se tiendra du 2 au 5 février 2009 à Plzen, République Tchèque (*http ://wscg.zcu.cz/WSCG2009/wscg2009.htm*).

Il a été accepté en tant que *full paper*, et a donc été présenté lors de cet événement.

Hybrid sort-first/sort-last rendering for dense material particle systems

Simon Latapie

CEA
DAM, DIF
F-91297 Arpajon, France

Laboratoire MAS
Ecole Centrale Paris
Grande Voie des Vignes
92290, Châtenay-Malabry, France

simon.latapie@gmail.com

ABSTRACT

This paper describes a solution designed for efficient visualization of large and dense sets of particles, typically generated by molecular dynamics simulations in materials science. This solution is based on a hybrid distributed sort-first/sort-last architecture, and meant to work on a generic commodity cluster feeding a tiled display. The package relies on VTK framework with various extensions to achieve statistical occlusion culling, smart data partitioning and GPU-accelerated rendering.

Keywords

Hybrid sort-first/sort-last rendering, Statistical culling, VTK, Ice-T, Particle rendering, Dense particle system

1. INTRODUCTION

Materials science increasingly uses numerical simulations at different scales of space and time to better understand and predict the properties of matter. Molecular dynamics is one of the most widely used approaches in computational materials science. Thanks to the joint advances in parallel computing and in physics modeling, molecular dynamics can now be used to simulate systems with millions to hundreds of millions of particles[Stre 05]. We focus here on such simulations at nanoscopic to microscopic scales, which describe matter in dense states by large sets of particles.

Suitable and efficient visualization tools must be provided besides simulation codes in order to benefit from these very detailed computations, and especially interactive tools that help to explore complex 3D features such as blast waves, solidifications, dislocations, etc. We are going to describe how we built a distributed visualization tool to exploit a small graphics cluster and tiled display for almost interactive exploration of such datasets. The solution is quite standard since it relies on widely used software components, such as VTK[Schr 06], with various optimizations.

After reviewing related work which partly inspired us, we are going to describe the overall organization of the system, then give more details on some technical aspects of the algorithms we combined: culling by space partitioning and statistical occlusion, particle rendering and parallelization.

2. RELATED WORK

Only few existing solutions are specifically designed for visualization of global phenomena inside large particle sets on a tiled display.

Many systems are especially designed for biological molecular dynamics, like VMD[Hump 96] or Molekel[Fluk 00]. Some exhibit very advanced rendering techniques, via GPU programming, accelerating complex shape rendering like TexMol[Baja 04], or global illumination like QuteMol[Tari 06]. Such tools are generally optimized to represent domain specific features or sub-structures with non-spherical shapes, like ribbons, tubes, or molecular surfaces. These representations cannot be used in materials physics where there are no such apparent structures as proteins parts, or identified zones like in Terascale Particle Visualization[Ells 04].

Our application domain requires to focus on efficient raw rendering of particles, basically represented as colored opaque spheres. Opaque sphere representation is very important because on dense particles systems from materials science it can preserve graphical aspects of several structure properties, such as some surfaces granularities, which are lost with non-opaque, point-only

represented particles, or volume rendering solutions, such as in Liang et al.[Lian 05] solution.

Moreover, most of the aforementioned tools cannot be easily integrated in a distributed rendering architecture. Another example in another application domain is Kruger et al.[Krug 05] solution, a very efficient particle system rendering to visualize 3D flow fields. Such a solution takes advantage of new GPUs rendering capabilities, but as all data is stored in graphic card memory, scalability and possible extensions to a distributed architecture are compromised.

Very few solutions are scalable and designed to display raw real sphere representation of large sets of particles on large definition displays such as tiled displays. Atomsviewer[Shar 03] is one of them: it uses efficient optimizations for sort-first rendering of large sets of particles: Z-order data organization, octree space partitioning, probabilistic culling method. However, Atomsviewer has been adjusted to a specific hardware and display configuration (ImmersaDesk^{TM})[Shar 02b].

All these observations have lead us to work on the integration of culling methods and hardware-accelerated rendering in a generic distributed architecture.

3. GENERAL OVERVIEW

The main objective of our architecture is to provide new optimizations while re-using VTK/Ice-T[More 01]. Ice-T is a sort-last rendering solution for tiled displays, which has been proven[More 03] to be more efficient than generic solutions for tiled displays such as Chromium[Hump 02]. Sort-last rendering is scalable with respect to the size of the data, but the known bottleneck of such an approach is the network bandwidth. Ice-T brings improvements to usual sort-last rendering, such as an efficient distribution of images to be composed, or a floating viewport technique. Nethertheless, network bandwidth remains the bottleneck of such a method. Our strategy is to achieve distributed sort-first operations to increase spatial coherence of per process data to reduce network bandwidth usage, and lower the number of spheres to be displayed, to reduce the actual sphere rendering stage. This can be considered to be an attempt to transpose Samanta et al.[Sama 00] hybrid architecture to non-structured particle systems visualization.

Before data is to be displayed, we generate a space partitioning Kd-Tree, and we reorganize the dataset to gather all data attached to a same leaf of the tree. Each node of the tree contains its bounds, its data storage location, a link to its children if it has any, and a density factor, which is described in section 4.

The global architecture of our system, described in Figure 1, is meant to run on a cluster of N nodes, with P of them actually connected to a display with P physical or logical tiles. Each node runs a MPI process. Each process has a complete copy of the tree, but only part of the data, in memory. During a frame rendering, each process computes frustum and occlusion culling for its own part of the tree, then all processes merge their results to have a complete up-to-date tree. At that time all processes know which data is really to be displayed. Then they compute a balanced per process redistribution of the data, load missing data and unload useless data if necessary, and render them. For the sort-last part of the architecture, Ice-T library performs a distributed rendering, and displays the frame.

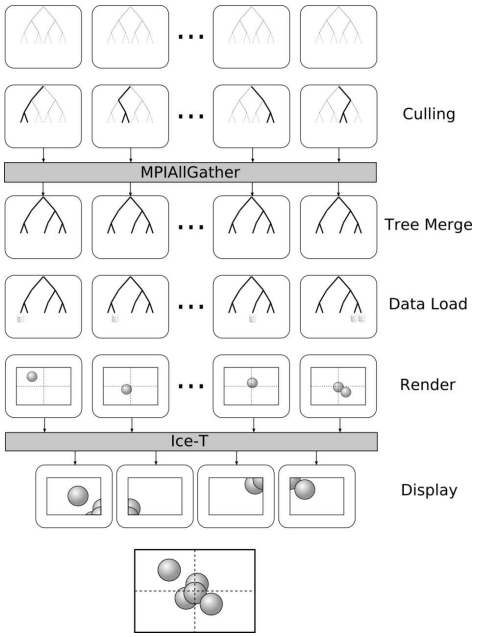


Figure 1: Global Architecture.

In the following sections, we are going to describe the tree for space decomposition and the culling algorithms we designed to organize datasets and optimize Ice-T rendering process. Then we will mention the GPU optimizations for the sphere rendering stage, the parallelization strategies, and finally we will highlight the implementation and a few results.

4. CULLING IN DENSE MATERIAL PARTICLE SYSTEMS

Although Atomsviewer is based on a specific architecture, some of the pre-rendering techniques it uses are very effective, such as the idea of probabilistic occlusion culling[Shar 02a]. This approach introduces a notion of density of particles in the cells of an octree, which is used to randomly drop part of the particles displayed.

Dense material systems are often composed of large groups of particles which are very close to each other. The probabilistic occlusion can be pre-computed, as our datasets are non-interactively generated. Atomsviewer's density computation, which is the volume

of particles divided by the volume of the cell, is not a very good factor to use for culling, because it assumes the distribution of particles is uniform in all cells.

In this section, we propose a complete pre-processing solution adapted to dense material systems visualization, which consists in a space partitioning algorithm, and a cell density computation. We also propose a culling algorithm which takes the pre-processing specificities into account.

Space partitioning

We use a Kd-Tree structure for space partitioning, which aims at separating dense space and empty space, in a very quick and simple way. We do not use the Kd-Tree VTK implementation, because of specific tree parsing methods and data order tests needs.

4.1.1 Empty space detection Kd-Tree

The algorithm described below is a simple way of generating a Kd-Tree for empty space detection.

As seen in Figure 2, for each tree node, we check particles positions for a given axis. If there is a left or right space between particles and node bounds, an empty space isolating split is done. Otherwise, a middle split is performed. Then the particles are sorted by comparing their axis position with the split position. Axis is alternatively x, y, then z.

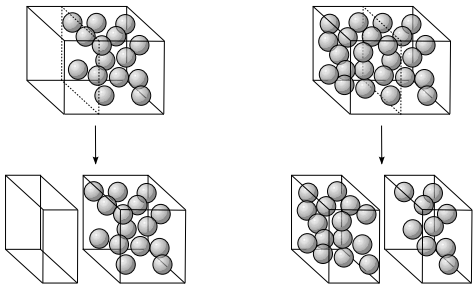


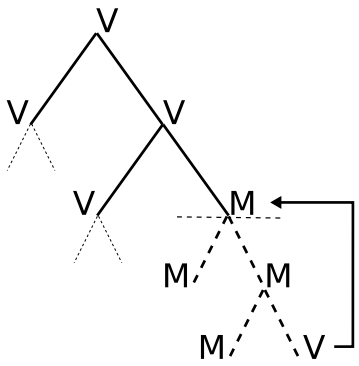
Figure 2: Kd-Tree generation: empty space partition

The complexity of the generation is $O(nln(n))$, where n is the number of particles, as the most consuming part is the particle sort, which is very much like a global quicksort, the split position being a pivot value. Optimum depth of the tree is very dependent on particles repartition in the scene, but empirical tests shows a 15 depth is a good overall value.

4.1.2 Kd-Tree optimization

The main goal of this space partition is to find dense cells which can occlude other cells. So we want the dense cells to be as large as possible. This is why we can optimize the Kd-Tree generation by moving up the effective splits, which are the empty space ones. Density computation is explained in section 4.2.

Figure 3 explains such an optimization: when a middle split is performed, a temporary tree branch is computed, until an empty space split is found, or maximum depth is reached. On the first case, the empty space split is applied on base node and the branch is computed again, otherwise the branch is kept as it is.



V node is an empty space splitting node (Void split)
M node is a middle splitting node

Figure 3: Kd-Tree optimization

Statistical occlusion culling

The following step in pre-processing is tree cells density computation. Density must describe the culling effect of a cell in relation to another one. We use a Monte-Carlo method to compute the probability for a ray to go through a space-partitioning cell containing spheres.

We launch N rays with random position and direction through the cell, and we check if the ray goes through any of the spheres or not (see Figure 4). Sphere radii are fixed attributes of the particles. It is a five degrees of freedom problem, and Monte-Carlo basic method provides a $1/\sqrt{N}$ error convergence. A typical number of casted rays for a 1% error on density for one cell is 100,000 casts.

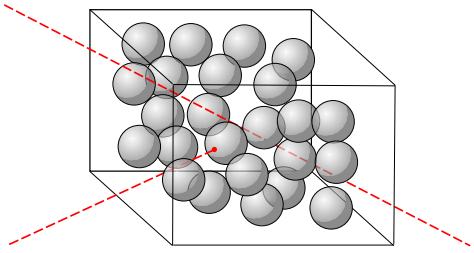


Figure 4: Monte-Carlo method for computing cell density.

This provides a 0 to 1 density factor, which represents the probability a ray has to go through the cell, and can be used as a trust percentage of this cell to occlude another. Each node of the tree has such a density factor.

Culling algorithms

We had to make a choice between the two most frequently used strategies for occlusion and frustum culling. The first one is silhouette comparison in viewport coordinates, like in [Coor 97]. The other one uses occlusion maps[Zhan 97]. We chose a strategy similar to the first one, because occlusion maps require a lot of GPU memory, which we would like to keep for the VBO cache system as described in section 5.2.

4.3.1 Silhouette computation

Occlusion culling is achieved by computing a silhouette of the occluding cell, and checking if a potentially occluded cell is inside the silhouette in viewport coordinates (cf Figure 5). Graham scan algorithm[Grah 72] gives us a y-sorted couple of point list which represents left side and right side of the silhouette.

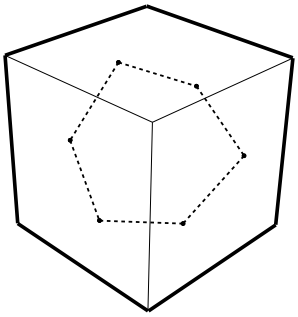


Figure 5: Block silhouette occluding another block

For each potentially occluded cell vertex, we find the left and right segments of the silhouette that share the same y-position as the top. Then we compare the x position of the top and the segments (see Figure6).

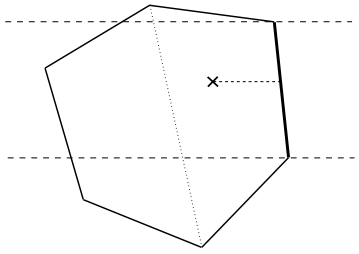


Figure 6: Silhouette occlusion check: X position compared to Y including segment

Depth check is done by comparing the z position of a vertex in relation to one or two planes. Planes are defined by triangles made by silhouette points close to the checked point, as seen in Figure 7. It is worth noting that this depth check is possible because we compare only disjoint cells.

4.3.2 Application to Kd-Tree

As seen in section 4.2, the Monte-Carlo pre-processing gives us a trust factor for cell opacity. If we set a threshold on this factor, we can consider some of the cells

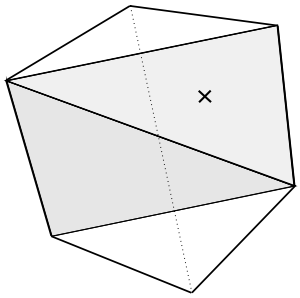


Figure 7: Depth check: triangles for planes definitions

are completely opaque, and then make a global culling comparison between tree nodes.

Each node has three possible states: *not occluded* (visible), *partially occluded*, and *occluded*. By default, all nodes are *not occluded* (visible). Leaves, as elementary undividable nodes, can only be tagged as *not occluded* (visible) or *occluded*.

As the tree is pre-computed, each node has a density attribute, and a maximum density of all the nodes beneath it. Then we recursively browse the tree from root to leaves, stopping as soon as a node with enough density is found.

For each occluding node, we perform the occlusion test, as seen in Figure 8: if the potentially occluded node (*PON*) contains no particle, or we already know it is occluded by another node, it is obviously occluded. If it is a child of the occluding node (*ON*), a test between the *ON*'s direct children and the *PON*. If the *ON* and the *PON* are the same node, and have children, they can potentially occlude a part of themselves, so we achieve a test between one of them and their children. If the *PON* is already partially occluded, we know that part of his children are already occluded, so we test his children. Otherwise, the silhouette of the *ON* is computed, and the occlusion is effective. Note that occlusion test between a node and one of is children can be meaningless because they can share bounds.

4.3.3 Frustum culling

Frustum culling is rather simple: the whole silhouette algorithm works in viewport coordinates. We only have to check if all cell coordinates are out of bounds per axis, i.e. if for any axis, cell coordinates are all either lower than -1 or higher than $+1$ in viewport coordinates.

5. PARTICLE RENDERING

In this section we describe some of the techniques we used to manage GPU memory and rendering.

Sphere rendering

We use GPU OpenGL shaders[Kess 06] programming to render particles as spheres. The big advantage of this technique is the per pixel precision of the rendering: thanks to the fragment shader program, all

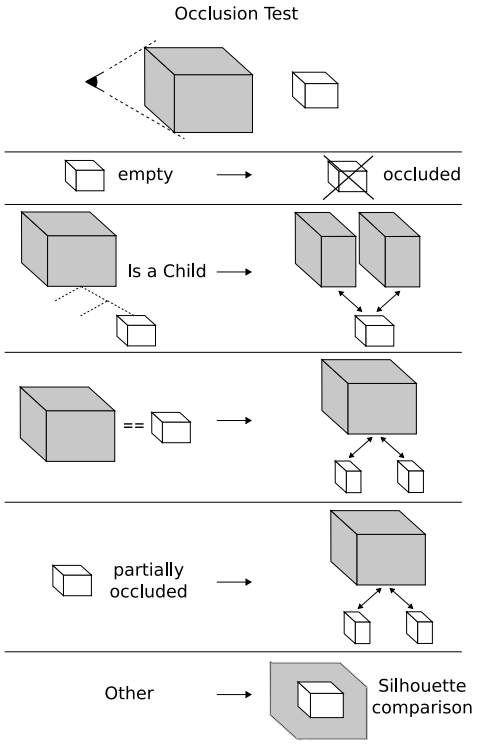


Figure 8: Occlusion Test

spheres are rendered pixel per pixel, and not with a group of vertices. This provides a direct level of detail feature, because rendering precision is only dependent on the number of pixels the sphere needs to be displayed. This also allows non standard lighting effects, like Phong[Phon 75] illumination, an example of which is shown in Figure 9.

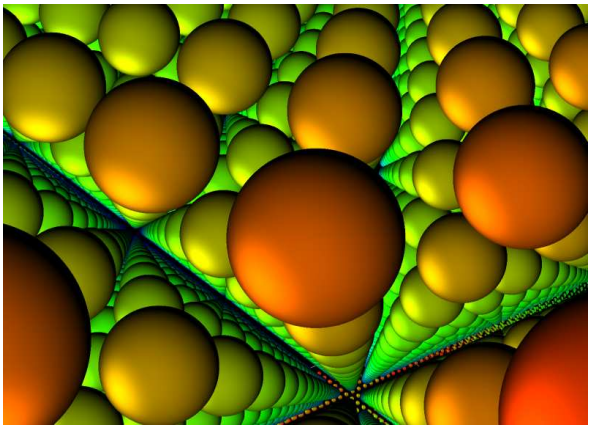


Figure 9: Phong illuminated spheres

GPU memory management

Vertex Buffer Objects (VBO)[Nvid 03] are a powerful way of managing GPU memory and RAM-to-GPU data transfers. We use them to create a cache system on GPU memory: for each tree leaf which is to be displayed, we create a VBO containing actual particles data used in rendering stage (positions, colors, radii). We unload it only if GPU memory is full and the leaf not displayed.

6. PARALLEL ALGORITHMS

This section presents the distributed strategies applied to previously described occluding and rendering algorithms.

Sort-last stage

As said before, we chose to use Ice-T, with *Reduce to Single Tile*[More 01] method: each process is assigned to one of the display tiles. First it renders the part of the scene which consists of the data loaded by this process. Then each process splits the rendered image, and for each part of the image, which is to be displayed by a tile, sends the part to one of the processes of the tile group. Each process receives a balanced number of partial images to be displayed by the tile. Then the processes of a same tile group compose their images by binary-swap before display. Moreland[More 03] tests on a generic architecture with a cluster and a tiled display were conclusive.

Since the Ice-T algorithm is very dependent on network bandwidth, it includes some optimizations, like floating viewport, which reduces the size of images transfered for compositing by detecting not rendered zones on the global viewport. This feature plays an important part in the sort-first algorithms efficiency and we used it as is.

Sort-first stage

6.2.1 *Partitioning of rendered tree parts*

We try to share parts of the tree to balance per cluster node rendering time and network transfers.

All processes have the complete tree structure loaded. Tree nodes numbering is in Z-order, like in [Shar 02a], which provides a good spatial coherence between nodes with close numbers.

For each frame, we assign the first not occluded leaves with an average number of particles to the first process, the following leaves to the second process, and so on, until all leaves are assigned, as seen in Figure 10. This average number or particles is the nearest integer to $(Visible particles)/(Number of cluster nodes)$, modulo the cardinality of the last leaf assigned to the current process.

The per node data is then spatially coherent, which is good for Ice-T floating viewport technique. Moreover, as two successive frames have relatively similar trees, cluster nodes have to render almost the same leaves, and the cache system is efficiently used.

6.2.2 *Overlapping culling algorithm*

The occlusion culling algorithm was easily modified to become distributed.

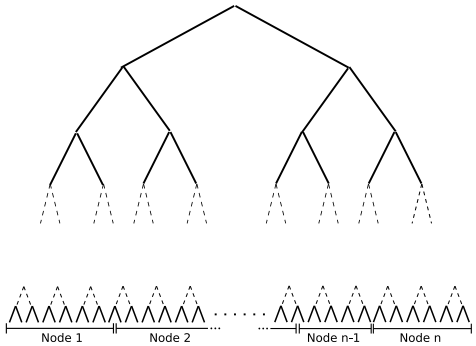


Figure 10: Displayable leaves assignment

Each frame, each cluster node performs an occlusion test, with only his previously assigned tree parts as potentially occluded cells. Then all nodes broadcast a signature of the tree, and each node merge them to have the entire tree configuration. The tree signature is a $(Numberofleaves)/4$ bytes buffer: each leaf is coded with 2 bits, because it has three states. All untested nodes are *not occluded*.

Although signature broadcast is not a very good scalable method, the really small size of the signature implies that this part does not slow down the overall process. As an example, with a typical depth of 15 for the tree for the 32 million particles dataset described in next section, the signature has a size of 8192 bytes, which is no more than Ethernet Gigabit maximum MTU (9000 bytes).

7. IMPLEMENTATION AND FIRST RESULTS

Base framework

Ice-T is integrated in Paraview[Ahre 05], based on the generic framework VTK. For our current implementation we use only core VTK and Ice-T, with a number of additional C++ classes compliant with VTK. We also created a *vtkMapper* family of classes to integrate VBO usage in VTK rendering process.

Protocol

We use a 32 millions particles dataset, describing a typical atomic system used in molecular dynamics detonation wave simulations. The graphics and display hardware is a four-tiled, 2048x1536 display, with an 8-node Gigabit Ethernet commodity cluster. Each node is a Bi-Xeon 3.4Ghz with a NVidia Quadro FX 4500 graphics board.

We compared three sort-first methods: a non hierarchical non ordered method, i.e. only a big VBO per cluster node; a Z-ordered repartition without occlusion; and the full occlusion method. In the first method, network is only used to send synchronization signals: data is distributed among nodes, and loaded once in GPU memory.

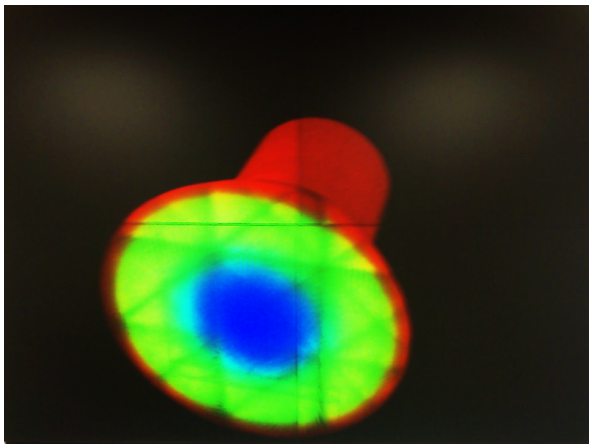


Figure 11: 32 million particle cylinder displayed on the 4-tile, 2m wide screen (around 3.3 M pixels)

As the full solution was designed to globally explore datasets, we tested two different camera movements: a rotation with constant long or short distance from dataset center; and a zoom towards a point of interest.

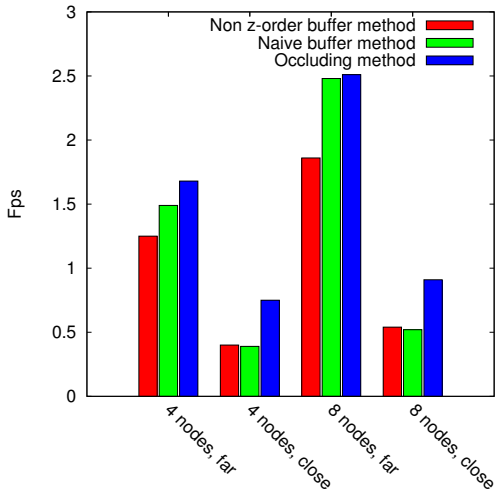


Figure 12: Camera rotating around dataset

On Figure 12 we can see the importance of Z-order in cluster nodes repartition.

We can notice that with eight nodes on the far view tests, culling (in blue) does not bring much better results than the naive (in green) method. The reason is that good Z-order data rendering repartition strongly reduces the floating viewport surfaces, which lower the network bandwidth usage. For example, in the 8 nodes far view test, bandwidth usage drops from 750Mb/s (effective maximum bandwitdth) with non Z-order method to an average 250Mb/s in both native and occluding method. On the close tests, Figure 12 shows the obvious efficiency of culling in such a situation. Z-order repartition and culling each reveal their efficiency in one of the different rotation scenarios.

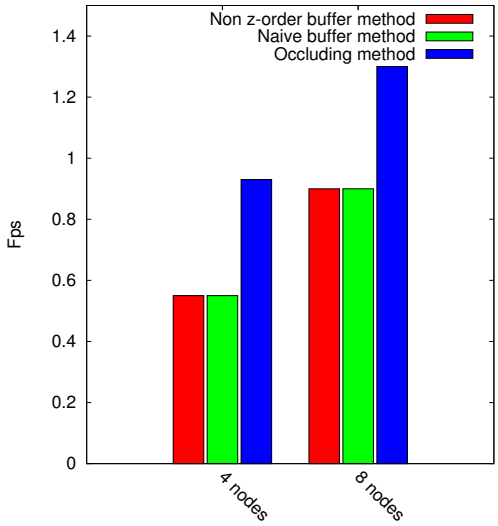


Figure 13: Camera zooming in dataset

Zooming results confirm this (Figure 13). As the camera gets closer to the dataset, the scene takes more and more space in the viewport. The network is saturated (like rotation tests, from an average 250Mb/s to 750Mb/s), and actual rendering becomes more and more time-consuming. Z-order is not very important, because the floating viewport is not activated here.

The rendering quality is a step function of the density threshold. If the density threshold is high enough, there is almost no difference between occluding and non occluding rendering. The threshold used for above results is 95%. For the camera rotation movement, 30% to 60% of the particles were culled. In the zoom test, culling got up to 94%. Lower threshold values obviously provide more interactive rendering, but artefacts do appear, like holes in the displayed dataset (cf Figure 14).

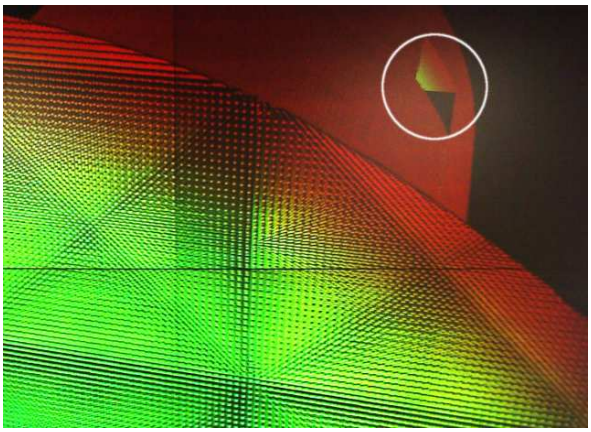


Figure 14: Artefacts with low density threshold

As priority for such a solution is global exploration and zone of interest detection, the most important factor in results is close rotation framerate. Our solution provides twice the framerate as the GPU only solution.

It is important to note that density is used the same way in far and in close cases. It should be interesting to lower threshold as camera's distance from datae increases. This could greatly improve results in far cases.

8. CONCLUSION AND FUTURE WORK

Our system provides a good framework for the exploration of large particle datasets representing dense material simulations. Sort-first rendering based on a specific Kd-Tree partitioning and statistical culling, with Monte-Carlo density computation, improves IceT sort-last strategies to reduce network bandwidth usage. Spheres rendering is also accelerated by GPU shaders. Future scale up tests will be achieved on a 12 tiled display fed by a 12 nodes cluster. We could further improve rendering by using better illumination methods, like ambient occlusion[Tari 06], to exploit the maximum potential of modern GPU power. Tests on a lower-latency and higher bandwidth network such as Infiniband should also be interesting.

Our next solution improvement will be to take camera's distance to the dataset into account to choose density threshold. We expect far camera tests framerate to be greatly improved.

Interactive tree computation is already fast, but with some optimizations, it could be done in real time. The main problem is density computation. One of the solutions could be to use General-Purpose computation on GPUs (GPGPU), like NVidia CUDA technology to accelerate the Monte-Carlo method.

9. ACKNOWLEDGEMENTS

Very special thanks to people from CEA/DIF, and especially Jean-Philippe Nominé for his support and his help, Thierry Carrard for his advices on GPU and parallel programming, Laurent Soulard for his teaching on materials physics. Also thanks to John Biddiscombe and Jean Favre from CSCS for respectively initial work on vtk sphere mapper classes and VTK advices, and Patrick Callet from Ecole Centrale Paris for his tips on sphere illumination, and reviewers for their thoughtful and accurate comments.

10. REFERENCES

[Ahre 05] J. Ahrens, B. Geveci, and C. Law. "ParaView: An End User Tool for Large Data Visualization". In: *Visualization Handbook*, Chap. 7, Academic Press, 2005.

[Baja 04] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. "TexMol: Interactive Visual Exploration of Large Flexible Multi-Component Molecular Complexes". In:

VIS '04: Proceedings of the conference on Visualization '04, pp. 243–250, IEEE Computer Society, Washington, DC, USA, 2004.

[Coor 97] S. Coorg and S. Teller. "Real-time occlusion culling for models with large occluders". In: *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pp. 83–ff., ACM, New York, NY, USA, 1997.

[Ells 04] D. Ellsworth, B. Green, and P. Moran. "Interactive Terascale Particle Visualization". In: *VIS '04: Proceedings of the conference on Visualization '04*, pp. 353–360, IEEE Computer Society, Washington, DC, USA, 2004.

[Fluk 00] P. Flukiger, H. Luthi, S. Portmann, and J. Weber. "MOLEKEL 4.0". 2000.

[Grah 72] R. L. Graham. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". *Inf. Process. Lett.*, Vol. 1, No. 4, pp. 132–133, 1972.

[Hump 02] G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. "Chromium: A Stream Processing Framework for Interactive Graphics on Clusters". 2002. presented at SIGGRAPH, San Antonio, Texas.

[Hump 96] W. Humphrey, A. Dalke, and K. Schulten. "VMD - Visual Molecular Dynamics". *Journal of Molecular Graphics*, Vol. 14, pp. 33–38, 1996.

[Kess 06] J. Kessenich. "The OpenGL Shading Language (1.20)". Tech. Rep., opengl.org, 2006.

[Krug 05] J. Kruger, P. Kipfer, P. Kondratieva, and R. Westermann. "A Particle System for Interactive Visualization of 3D Flows". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 11, No. 6, pp. 744–756, 2005.

[Lian 05] K. Liang, P. Monger, and H. Couchman. "Interactive parallel visualization of large particle datasets". *Parallel Comput.*, Vol. 31, No. 2, pp. 243–260, 2005.

[More 01] K. Moreland, B. Wylie, and C. Pavlakos. "Sort-Last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays". In: *PVG '01: Proceedings of the IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics*, pp. 85–92, IEEE Press, 2001.

[More 03] K. Moreland and D. Thompson. "From cluster to wall with VTK". *IEEE symposium on Parallel and Large-Data Visualization and Graphics*, pp. 25–31, 2003.

[Nvid 03] Nvidia. "Using vertex buffer objects". Tech. Rep., NVIDIA Corporation, 2003. `http://developer.nvidia.com/object/using_VBOs.html`.

[Phon 75] B. T. Phong. "Illumination for Computer Generated Pictures". *Commun. ACM*, Vol. 18, No. 6, pp. 311–317, 1975.

[Sama 00] R. Samanta, T. Funkhouser, K. Li, and J. Singh. "Hybrid sort-first and sort-last parallel rendering with a cluster of pcs". In: *Eurographics/SIGGRAPH workshop on Graphics hardware*, pp. 99–108, 2000.

[Schr 06] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th edition*. Kitware, Inc. publishers, 2006.

[Shar 02a] A. Sharma. *Techniques and algorithms for immersive and interactive visualization of large datasets*. PhD thesis, Louisiana State University, 2002.

[Shar 02b] A. Sharma, X. Liu, P. Miller, A. Nakano, R. K. Kalia, P. Vashishta, W. Zhao, T. J. Campbell, and A. Haas. "Immersive and Interactive Exploration of Billion-Atom Systems". In: *VR '02: Proceedings of the IEEE Virtual Reality Conference 2002*, p. 217, IEEE Computer Society, Washington, DC, USA, 2002.

[Shar 03] A. Sharma, R. K. Kalia, A. Nakano, and P. Vashishta. "Large Multidimensional Data Visualization for Materials Science". *Computing in Science and Engg.*, Vol. 5, No. 2, pp. 26–33, 2003.

[Stre 05] F. H. Streitz, J. N. Glosli, M. V. Patel, B. Chan, R. K. Yates, B. R. de Supinski, J. Sexton, and J. A. Gunnels. "100+ TFlop solidification simulations on BlueGene/L.". In: *Gordon Bell Prize at IEEE/ACM SC05*, 2005.

[Tari 06] M. Tarini, P. Cignoni, and C. Montani. "Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization". *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 5, pp. 1237–1244, 2006.

[Zhan 97] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff III. "Visibility Culling Using Hierarchical Occlusion Maps". *Computer Graphics*, Vol. 31, No. Annual Conference Series, pp. 77–88, 1997.

[Amia] Amira - advanced 3d visualization and volume modeling. http://www.amiravis.com.

[Amib] Amiramol. http://www.amiravis.com/mol/.

[AS01] Ulf Assarsson and Per Stenström. A case study of load distribution in parallel view frustum culling and collision detection. *Lecture Notes in Computer Science*, 2150 :663–673, 2001.

[Avi] Avizo. http://www.tgs.com/products/avizo.asp.

[AVS] Avs - advanced visual systems. http://www.avs.com.

[BB92] S. Bernard and P. Blottiau. Structure moléculaire et applications. *Revue scientifique et technique Chocs de la Direction des applications militaires du CEA*, 4 :69–82, May 1992.

[BDST04] Chandrajit Bajaj, Peter Djeu, Vinay Siddavanahalli, and Anthony Thane. Texmol : Interactive visual exploration of large flexible multi-component molecular complexes. In *VIS '04 : The conference on Visualization '04*, pages 243–250, Washington, DC, USA, 2004. IEEE Computer Society.

[BH99] Doug A. Bowman and Larry F. Hodges. Formalizing the design, evaluation, and application of interaction techniques for immersive virtual environments. *Journal of Visual Languages and Computing*, 10(1) :37–53, 1999.

[BJM+03] S. Bernard, F. Jollet, B. Magne, M. Torrent, and G. Zérah. Modélisation haute performance des matériaux à l'échelle microscopique. *Revue scientifique et technique Chocs de la Direction des applications militaires du CEA*, 428 :87–94, October 2003.

[Bra84] Myron L. Braunstein. Perception of rotation in depth : the psychophysical evidence. *SIGGRAPH Comput. Graph.*, 18(1) :25–26, 1984.

[Bry04] Steve Bryson. Direct manipulation in virtual reality. In *Visualization Handbook* [JH04], chapter 7.1.

[BSP+93] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. DeRose. Toolglass and magic lenses : The see-through interface. In *SIGGRAPH-93 : Computer Graphics*, pages 73–80, Anaheim, CA, 1993.

[BW01] D. Bowman and C. Wingrave. Design and evaluation of menu systems for immersive virtual environments. Technical Report TR-01-20, Computer Science Department of Virginia Tech, 2001.

[CFGP00] Sabine Coquillart, Philippe Fuchs, Jérôme Grosjean, and Alexis Palijic. Primitives comportementales virtuelles de communication avec autrui ou contrôle d'application. In *Le traité de la réalité virtuelle, 2e édition, Volume 1* [FM00], chapter 13.5, pages 454–463.

[CG05] G. Journé C. Guilbaud. Intégration de périphériques de réalité virtuelle dans des applications de visualisation scientifique au sein de la plate-forme vtkvrpn. *Rapport CEA R-6092*. Technical report, 2005.

[CHWS88] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *ACM CHI Conference on Human Factors in Computing Systems*, pages 95–100, 1988.

[Cos] Pierre Costesèque. Caractérisations structurales des matériaux - bases de cristallographie - introduction. http://mfca.ups-tlse.fr/radiocris/cristal/introduc/intrprin.htm.

[CPB04] Jian Chen, Pardha S. Pyla, and Doug A. Bowman. Testbed evaluation of navigation and text display techniques in an information-rich virtual environment. In *IEEE Virtual Reality Conference - Chicago, IL*, March 2004.

[Cry] Crystalmaker. http://www.crystalmaker.com/.

[CT97] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *SI3D '97 : Symposium on Interactive 3D graphics*, pages 83–ff., New York, NY, USA, 1997. ACM.

[CUD] Cuda. http://www.nvidia.com/object/cuda_learn.html.

[CV02] Carlos J. Camacho and Sandor Vajda. Protein-protein association kinetics and protein docking. *Current Opinion in Structural Biology*, 12-1 :36–40, 2002.

[Ens] Ensight. http://www.ensight.com.

[ePS00] Philippe Fuchs et Panagiotis Stergiopoulos. Les interfaces manuelles sensori-motrices, interfaces à retour d'effort. In *Le traité de la réalité virtuelle, 2e édition, Volume 1* [FM00], chapter 8, pages 267–310.

[Esn05] A. Esnard. *Analyse, conception et réalisation d'un environnement pour le pilotage et la visualisation en ligne de simulations numériques parallèles*. Informatique, Université de Bordeaux 1, Dec 2005.

[Exc90] Thierry Excoffier. Le lancer de rayons : fondements et méthodes d'accélération. Rapport technique, Université Claude Bernard Lyon I, 43, bd du 11 Novembre 1918, 69622 Villeurbanne, France, 1990.

[FM00] P. Fuchs and G. Moreau. Ecole de Mines de Paris, Les Presses, Dec 2000.

[GC01] J. Grosjean and S. Coquillart. Command & control cube : a shortcut paradigm for virtual environments. In *IPT-EGVE*, 2001.

[Gra72] Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4) :132–133, 1972.

[GTG84] Cindy M. Goral, Kenneth E. Torrance, and Donald P. Greenberg. Modeling the interaction of light between diffuse surfaces. In *Computer Graphics, 18(4), Annual Conference Series, ACM SIGGRAPH*, pages 213–222, 1984.

[Gum03] Stefan Gumhold. Splatting illuminated ellipsoids with depth correction. In *The Vision, Modeling, and Visualization Conference 2003 (VMV 2003), München, Germany*, pages 245–252, 2003.

[GW00] F. Guimbretière and T. Winograd. Flowmenu : Combining command, text, and data entry. Technical Report CS-TR-2000-01, Stanford Computer Science Department, May 2000.

[Han97] Chris Hand. A survey of 3d interaction techniques. *Comput. Graph. Forum*, 16(5) :269–281, 1997.

[HDS96] William Humphrey, Andrew Dalke, and Klaus Schulten. Vmd - visual molecular dynamics. *Journal of Molecular Graphics*, 14 :33–38, 1996.

[HHN+] G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski. Chromium : A stream processing framework for interactive graphics on clusters. presented at SIGGRAPH, San Antonio, Texas, 2002.

[HPGK94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. Passive real-world interface props for neurosurgical visualization. In *ACM CHI'94 Conference on Human Factors in Computing Systems*, volume 1, pages 452–458, 1994.

[HV97] M. Hadwiger and A. Varga. Visibility culling. In *Paper for seminar at the Institute of Computer Graphics and Algorithms, Vienna University of Technology*, 1997.

[Ins98] Alfred Inselberg. Visual data mining with parallel coordinates. *Computational Stat.*, 134(1) :47–63, 1998.

[JH04] Christopher R. Johnson and Charles D. Hansen. Academic Press, June 2004.

[KB93] G. Kurtenbach and W. Buxton. The limits of expert performance using hierarchic marking menus. In *ACM/IFIP Conference on Human Factors in Computing Systems (INTERCHI '93)*, pages 482–487. ACM Press, 1993.

[Kes06] John Kessenich. The opengl shading language (1.20). Technical report, opengl.org, 2006.

[Kok03] A. Kokalj. Computer graphics and graphical user interfaces as tools in simulations of matter at the atomic scale. *Comp. Mat. Sci.*, 28 :155–168, 2003.

[Kra91] Per J. Kraulis. Molscript : A program to produce both detailed and schematic plots of protein structures. *Journal of Applied Crystallography*, 24 :946–950, 1991.

[KvL07] Arjan J. F. Kok and Robert van Liere. A multimodal virtual reality interface for 3d interaction with vtk. *Knowl. Inf. Syst.*, 13(2) :197–219, 2007.

[KW04] R. Komerska and C. Ware. A study of haptic linear and pie menus in a 3d fish tank vr environment. In *IEEE Virtual Reality - Haptics Symposium, Chicago, IL*, March 2004.

[Li05] J. Li. Atomistic visualization. In Dordrecht S. Yip Springer, editor, *Handbook of Materials Modeling*, chapter 2.31, pages 1051–1068. 2005.

[LWC+02] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[McC98] Tulika Mitra and Tzi cker Chiueh. Implementation and evaluation of the parallel mesa library. In *IEEE International Conference on Parallel and Distributed Systems*, pages 84–91, 1998.

[MCEF94] Steve Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. Technical Report TR94-023, 8 1994.

[Mol] Molekel. http://bioinformatics.org/molekel.

[MT03] K. Moreland and D. Thompson. From cluster to wall with vtk. *IEEE symposium on Parallel and Large-Data Visualization and Graphics*, pages 25–31, 2003.

[MWP01] K. Moreland, B. Wylie, and C. Pavlakos. Sort-last parallel rendering for viewing extremely large data sets on tile displays. In *PVG '01 : Symposium on Parallel and Large-Data Visualization and Graphics*, pages 85–92. IEEE Press, 2001.

[NVI03] NVIDIA. Using vertex buffer objects. Technical report, NVIDIA Corporation, 2003. http://developer.nvidia.com/object/using_VBOs.html.

[OIV] Open inventor. http://oss.sgi.com/projects/inventor/.

[Opea] Openrt. http://www.openrt.de.

[Opeb] Openrtrt, real-time ray tracing. http://www.tgs.com/products/openRTRT.asp.

[Para] Paraview. http://www.paraview.org.

[Parb] Paraview - multiple views. http://www.paraview.org/Wiki/Multiple_views.

[Pas00] V. Pascucci. Multi-resolution indexing for hierarchical out-of-core traversal of rectilinear grids. In *NSF/DoE Lake Tahoe Workshop on Hierarchical Approximation and Geometrical Methods for Scientific Visualization, Tahoe City, California*, 2000.

[Per98] K. Perlin. Quikwriting : Continuous stylus-based text entry. In *UIST'98*, pages 215–216, 1998.

[PH04] Constantine Pavlakos and Philip D. Heermann. Issues and architectures for large data visualization. In *Visualization Handbook* [JH04], chapter 8.4.

[Pho75] B. Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6) :311–317, 1975.

[RTB96] Bernice E. Rogowitz, Lloyd A. Treinish, and Steve Bryson. How not to lie with visualization. *Comput. Phys.*, 10(3) :268–273, 1996.

[SA06] Mark Segal and Kurt Akeley. The opengl graphics system : A specification (version 2.1). Technical report, opengl.org, 2006.

[Sen] Sensable. http://www.sensable.com/.

[SFLS00] R. Samanta, T. Funkhouser, K. Li, and J. Singh. Hybrid sort-first and sort-last parallel rendering with a cluster of pcs. In *Eurographics/SIGGRAPH workshop on Graphics hardware*, pages 99–108, 2000.

[Sha02] Ashish Sharma. *Techniques and algorithms for immersive and interactive visualization of large datasets*. PhD thesis, Louisiana State University, 2002.

[SMW95] Roger Sayle and E. James Milner-White. Rasmol : Biomolecular graphics for all. *Biochemical Sciences (TIBS)*, 20(9) :374, 1995.

[Spa] Sparticles : A gpu/sprite based renderer for particle simulation data. http://www.cscs.ch/a-display.php?id=170.

[TCH+03] N. Tarrin, S. Coquillart, S. Hasegawa, L. Bouguila, and M. Sato. The stringed haptic workbench : a new haptic workbench solution. In *Eurographics*, Sept 2003.

[TCM06] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5) :1237–1244, 2006.

[VRP] Vrpn - virtual reality peripheral network. http://www.cs.unc.edu/Research/vrpn/.

[VTK] Vtk - the visualization toolkit. http://public.kitware.com/VTK/.

[WD00] G. Wesche and M. Droske. Conceptual free-form styling on the responsive workbench. In *ACM Symposium on Virtual Reality Software and Technology, Seoul, Korea*, pages 83–91, 2000.

[WDC99] K. Watsen, D. Darken, and M. Capps. A handheld computer as an interaction device to a virtual environment. In *The Immersive Projection Technology Workshop, Stuttgart, Germany*, 1999.

[WH00] G. Wagner and A. Hashibon. Aviz atomic vizualization. In *Computational Physics Group, Israel Inst. Technology Technion*, 2000.

[ZB04] Leonid Zhukov and Alan H. Barr. Oriented tensor reconstruction. In *Visualization Handbook* [JH04], chapter 5.1.

[ZLK04] Song Zhang, David H. Laidlaw, and Gordon Kindlmann. Diffusion tensor mri visulization. In *Visualization Handbook* [JH04], chapter 5.1.

[ZMHH97] Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. *Computer Graphics*, 31(Annual Conference Series) :77–88, 1997.