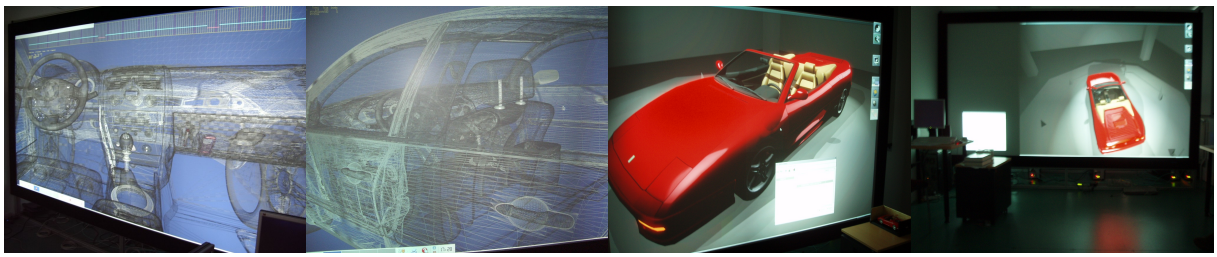
HiD^{2}RA, une bibliothèque extensible pour le rendu distribué sur mur d'écran

Submission id : 107



Abstract

Cet article présente HiD^{2}RA (High Definition Distributed Rendering Architecture), une bibliothèque pour le rendu sur grappe de PC et environnement multi-projecteurs. Cette architecture se focalise sur du rendu haute qualité en facilitant la programmabilité des cartes graphiques. Nous proposons une implantation efficace d'un graphe de scène mandataire distant, appelé métagraphe de scène. La bande passante réseau est controlée à l'aide d'une synchronisation à la demande. Isoler la couche réseau à la fois de l'application cliente et des algorithmes de rendu, facilite l'évolution et la création de nouvelles applications, une bonne approche dans un contexte de developpement multi-site.

1. Introduction

Un affichage haute résolution offre une bonne immersion dans une scène et améliore la perception de ses détails, comparé à un écran unique. Les cadres d'application existants sont focalisés sur la visualisation scientifique et la réalité virtuelle, où les performances et l'interactivité sont plus privilégiés que la qualité du rendu. En revanche, le prototypage virtuel et les phases de pré-marketing ont besoin d'une plus grande qualité visuelle, concrétisée à l'aide de techniques d'éclairage avancées. Ces dernières utilisent de manière intensive la programmabilité des cartes graphiques (GPU) et se limitent souvent à la résolution d'un écran de station de travail. Etendre cette résolution sur du matériel existant exige trop de capacité de calcul. De plus, parmi beaucoup d'architectures logicielles existantes pour le rendu sur mur d'affichage, peu offrent la possibilité d'implanter ces algorithmes.

Notre bibliothèque nommée High Definition Distributed Rendering Architecture (HiD^{2}RA) a pour but d'être une architecture haute performance pour un rendu sur mur d'écran. Dans notre contexte, les performances sont plus liées à celles des GPU qu'à celles de l'unité centrale. Par conséquent, nous nous ne sommes pas attaqués aux problèmes de rendu out-of-core et d'équilibrage de charge. Au contraire, nous nous sommes concentrés sur l'ergonomie vis à vis des créateurs de contenu (utilisateurs), mais aussi des créateurs d'applications (développeurs). Notre contribution, appelée le métagraphe de scène (metaGS), propose :

- un découplage élevé entre les codes de rendu et de distribution
- une interface de programmation simple et exhaustive pour des graphes de scène existants

Ce document est organisé de la manière suivante. Après une discussion sur des travaux précédents (section 2), nous présentons l'architecture générale de notre bibliothèque, basée sur une architecture 2-tier client/serveur (section 3). Les détails d'implantation sont donnés dans la section 4. En conclusion, nous comparons notre proposition avec des solutions *sort-first*, ClusterJuggler, OpenSG et hybrides (*sort-first/sort-middle*) comme Chromium et FlowVR-Render (section 5). †

† Figure 1 : De gauche à droite : Rendu HiD^{2}RA OpenSceneGraph ; TinySG avec un éclairage par pixel, un ombrage par carte d'ombre et un effet de brillance (modèles fournis par Renault CTS et DMI cars, dmi.chez-alice.fr) sur notre mur d'écran 3.6x1.8m

2. Travaux précédents

La parallélisation de tout algorithme est liée à ses tâches et à ses données d'entrée. Nous examinons ces aspects, appliqués au rendu temps réel par *rastérisation*.

2.1. Distribution des tâches

Le principe de la distribution des tâches est relativement simple. La surface d'affichage est divisée en sous surfaces sur chacunes desquelles la scène est distributée. Cela peut être réalisé selon trois niveaux de granularité des données : les objets de la scène, les primitives polygonales ou les fragments, respectivement connus comme des distributions sort-first, sort-middle et sort-last [MCEF94].

2.1.1. Distribution à grain fin

Une granularité fine favorise l'équilibrage de charge, mais nécessite un réseau à haut débit. La parallélisation par fragment est la forme la plus simple et la plus ancienne de rendu sur grappe [Fuc77, FJ79]. Des réalisations plus récentes, comme SEPIA [HM99], Scalable Graphics Engine [PJ01] et SAGE [JJR*05], se sont spécialisées sur la visualisation scientifique et exigent toujours des matériels coûteux avec des bus rapides dédiés.

Le rapport prix/performance attractif des grappes de PCs est à l'origine de l'apparition des réalisations logicielles sort-middle. WireGL [HEB*01] est une version distribuée de la bibliothèque OpenGL. Son principal avantage est sa capacité d'exécuter pratiquement toute application sans aucune modification de son code source.

Récemment, un tel mode de distribution à grain fin a été porté sur des GPU avec des bus PCI (Openvidia [FM05]) ou à l'aide de bus plus rapides : SLI de NVidia et Crossfire de ATI. Le SLI consiste en deux modes : entrelacement temporel et distribution des sommets sur des sous-surfaces redimensionnées dynamiquement. Le Crossfire d'ATI est plus prometteur puisqu'il fournit une composition en sort-last : l'image est séparée en damier et envoyée à chaque GPU. Un FPGA recompose ensuite les signaux vidéos numériques issus des cartes. Cependant, à l'heure actuelle, ces solutions n'offrent qu'une amélioration limitée des performances des algorithmes du rendu, de plus en plus orientés vers le traitement par fragments. Enfin, ces solutions sont seulement applicables à des résolutions limitées.

2.1.2. Distribution à gros grain

L'idée des distributions sort-first est d'agréger des primitives dans des représentations de plus haut niveau comme des maillages, eux-même regroupés en objets. Une élimination précoce des objets invisibles depuis la caméra devient alors plus efficace et permet de réduire sensiblement le débit nécessaire sur le réseau. Cette classe de distribution se décompose en deux sous-classes, une descendante, qui découpe la scène en différents ensembles d'objets et une ascendante, qui regroupe des primitives. L'approche descendante est souvent mise en application avec un graphe de scène, une structure hiérarchique de noeuds regroupant les objets de la scène.

Les performances des distributions sort-middle sont ainsi améliorées en créant des hybrides sort-first/sort-middle. Ainsi, Chromium [HHN*02] a été construit sur les bases de WireGL, comme une implantation ascendante. Au lieu de travailler en mode immédiat, des listes d'affichage (display list) sont créées afin de regrouper des triangles sur des heuristiques spatiales. FlowVR-Render [AR05] est une approche descendante : un métanoeud regroupe une géométrie, sa boîte englobante, son matériau et d'éventuels shaders. L'objectif de cet environnement logiciel est une visualisation scientifique efficace, compatible avec VTK [SML98]. Cependant, pour ces deux logiciels, seul un sous-ensemble des opérations graphiques de la librairie OpenGL est fonctionnel. Nous prolongerons la comparaison avec Chromium et FlowVR-Render dans la section 5.

2.2. Distribution des données

Une distribution des données est nécessaire afin de synchroniser sur une grappe, une scène modifiée de manière dynamique, par exemple, lors d'un changement de point de vue de la caméra ou de matériaux liés à des objets. Les graphes de scènes peuvent alors être considérés comme des bases de données réparties offrant des services de synchronisation distribuée de ses données. Le problème principal est alors de trouver un compromis entre une efficacité maximale et une certaine évolutivité. Dans les sous-sections suivantes, nous étudions différents moyens triés par granularité de données.

2.2.1. Propriété de noeud

Une propriété de noeud est le niveau le plus fin de la distribution, limitée aux membres de ce noeud, tels qu'une matrice d'un noeud de transformation.

Premièrement, cette distribution peut être implicite. Repo-3D [MF98] est ainsi écrit en Modula III, un langage avec des primitives nativement distribuées. Par conséquent, cette distribution à très bas niveau permet une transparence totale au niveau du code. Malheureusement, ce langage n'est pas très répandu et peut alors rendre la migration de d'applications existantes assez complexe.

Deuxièmement, un conteneur de donnéees peut fournir une distribution explicite. Par exemple, OpenSG [BRR02] est le seul graphe de scène où une exécution multiprocessus est garantie et mise en valeur à l'aide de conteneurs. ClusterJuggler [Ols02] fournit une classe sérialisable à VR-Juggler [JBBCN98], un framework de réalite virtuelle. Bien qu'offrant de nombreux services de distribution, ces conteneurs imposent de fortes contraintes lors de leurs utilisations, détaillées dans la section 5.

Enfin, des bibliothèques peuvent externaliser la distribution. Ainsi sont utilisés MPI, une bibliothèque de transport de messages issue du calcul scientifique parallèle, Diverse [AKKN01] ou Glass [dPGaGZ04], plus focalisées sur le rendu.

Travailler avec l'une de ces bibliothèques à une telle granularité offre certes de bonnes performances. Cependant cela impose d'écrire un algorithme de distribution à un niveau tout aussi bas. Le développement d'une application n'est alors pas facilité par ce processus relativement astreignant.

2.2.2. Noeud distribué

Inclure les propriétés distribuées dans des noeuds dédiés permet aussi d'automatiser la distribution. Par exemple, Syzygy offre son propre ensemble de noeuds [Sch00]. Néanmoins, il fonctionne avec son propre système d'exploitation,

ce qui limite relativement son évolutivité. Des graphes de scène existants ont été aussi étendus avec de tels noeuds. Distributed Open Inventor [HSFP99] fournit des noeuds distribués pour Open Inventor de SGI. Si Avango [Tra99] n'offre seulement qu'une surcharge partielle des noeuds de Performer, Blue-C est plus complet et efficace, avec une gestion préemptive d'accès concurrents en écriture [NSG05]. Cependant, ces graphes existants et par conséquent, leurs modifications ne facilitent pas la mise en oeuvre de rendu multipasses.

2.2.3. Langage de manipulation de graphe

Un langage de manipulation de graphe permet de séparer la couche de distribution du graphe de scène. Il décrit la création, la suppression ou l'édition de tout noeud. Ainsi, Syzygy fournit un protocole de modification de graphe, CDCL (Complex Data Communication Language) [Sch00] exhaustif mais plutôt complexe. Virpi-Aura propose un langage plus simple [dSRG*02]. Cependant, ces deux graphes de scène sont si étroitement intégrés à leurs environnements respectifs que les faire évoluer est difficile.

3. Architecture

L'état de l'art précédent nous a mené à quelques choix d'architecture pour HiD²RA. Concernant la distribution des tâches, un méthode sort-first pourrait être la meilleure option pour déployer des opérations de rendu par pixel à des fréquences d'affichage acceptables. La virtualisation‡, introduite dans le cadre du calcul distribué, est une bonne approche pour la programmation distribuée. Nous avons tenté de l'appliquer non seulement lors du déploiement et de l'exécution mais aussi lors du développement : la gestion du réseau et le code du rendu devant être clairement séparés. C'est pourquoi nous ne pouvons pas ni créer de conteneur distribué, ni proposer de noeud spécialisé, trop lié au graphe de scène sous-jacent. Nous avons donc choisi une solution hybride entre le noeud et le langage de modification, basée sur le motif de conception *remote proxy* [GHJV95].

HiD²RA est découpé en deux entitées, le *frontal* et les *serveurs de rendu* selon une architecture maître-esclave. Le frontal (FE) distribue les ordres issus de l'application cliente (AC), aux serveurs de rendu (BE) (section 3.1). Ces derniers sont responsables de la production de l'image sur le mur d'affichage (section 3.2). L'architecture complète est représentée sur la figure 3.

3.1. Frontal

Le frontal est l'interface entre HiD²RA et l'application cliente. Cette dernière est articulée autour d'une boucle de rendu à trois étapes bien connue : mise à jour de la scène, rendu et synchronisation. Le FE récupère ainsi les ordres de mise à jour et de rendu à travers la couche d'interface d'application et puis les expédie aux BE à l'aide de la couche de distribution de données. Enfin, l'AC attend à une barrière logicielle, levée par le FE à l'accomplissement du rendu.

‡ Une application ou un matériel réparti est alors perçu par ses utilisateurs comme une entité unique (ex : système multiprocesseur, partition virtuelle de disques).

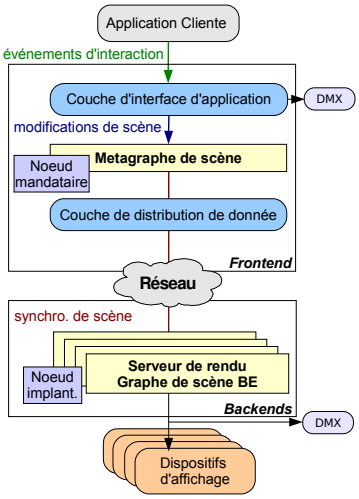


Figure 3: *Architecture générale*

3.1.1. Couche d'interface d'application

Cette interface fournit une virtualisation des entrées-sorties. Les entrées provenant des différents périphériques sont partagées vers les noeuds BE. De manière identique, les sorties sont appliquées à un environnement fenêtré (bureau) distribué. Dans un environnement collaboratif, un affichage fenêtré est sensiblement mieux adapté qu'un affichage plein écran, afin de partager des informations entre plusieurs fenêtres et utilisateurs. Avec Chromium, seul HiD²RA fournit une interaction avec DMX pour une intégration à bureau distribué. ClusterJuggler et OpenSG se contentant seulement d'un affichage plein écran.

Afin de faciliter l'administration de la grappe comme une machine virtuelle unique, nous fournissons également un outil pour la gestion du mur d'affichage avec des télécommandes logicielles des différents matériels. Le déploiement sur un matériel ciblé n'exige aucune recompilation ; il est effectué en éditant un fichier XML de configuration.

3.1.2. Couche de distribution de données

Le développement des BE devrait être concentré sur le code de rendu, et non pas sur la couche de distribution. C'est pourquoi cette couche ne devrait être contenue ni dans l'application ni dans les BE mais dans une couche spécifique du FE. Par conséquent, ce développement est centralisé autour d'un *métagraphe de scene*, une implantation efficace et générique d'une interface de procuration distante appliquée à un graphe de scène.

Cette couche fournit une interface de programmation simple pour manipuler les données de la scène et les synchroniser entre les BE. Un metaGS se compose de noeuds mandataires qui sont des interfaces intuitives et efficaces permettant d'accéder aux propriétés des noeuds du graphe par des métadonnées. Elles sont formées par des paires [clef, valeur] où la clef est le nom de la propriété de noeud et la valeur, un conteneur de données de type scalaire ou vectoriel.

Le metaGS fournit une couverture normalisée de l'interface des graphes existants et ouverte à de futurs développements. Différentes applications peuvent être alors branchées

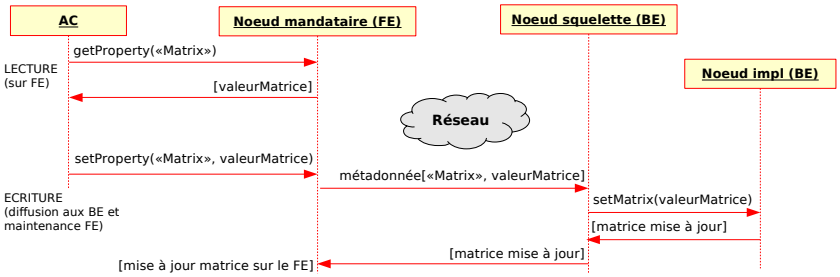


Figure 2: *Spécification de la distribution de données, basée sur le motif de conception mandataire distant avec une synchronisation à la demande*

sur notre bibliothèque pour bénéficier d'un même moteur de rendu fourni par un BE. De même, divers BE peuvent être liés avec une même AC pour lui fournir différents rendus.

3.2. Serveur de rendu

Un serveur de rendu calcule sa propre surface d'affichage. HiD^{2}RA facilite le développement de BE par l'intermédiaire d'une interface de *plugin* simple mais exhaustive. Ces derniers peuvent être permutés à la volée. Lorsqu'un graphe de scène n'est pas nécessaire ou qu'il n'est jamais modifié, l'interface *BeSceneView* est suffisante. Elle consiste en deux méthodes : placer la caméra (*setCamera*) et dessiner la scène (*draw*).

Pour un graphe de scène distribué, un BE doit implanter deux interfaces, *BeSgSceneView* qui hérite de *BeSceneView*, et *SgNodeImpl*. Cette interface se charge de fournir les listes de propriétés des noeuds du graphes ainsi que leurs accès en lecture/écriture (figure 4). En héritant des classes de couche de communication de données, ces classes n'ont seulement qu'à interpréter les messages reçus, sans jamais utiliser de code réseau. En utilisant ce principe, un metaGS peut être pratiquement lié à n'importe quel graphe existant. Les éléments écrits en rouge de la figure 5 récapitulent les objets qu'un développeur doit écrire.

4. Implantation

4.1. Frontal

4.1.1. Mandataire distant avec synchronisation à la demande

La couche de communication de données est responsable de la synchronisation des données sur le réseau. Cette couche contenant le métagraphe de scène, est une réalisation du motif de conception *remote proxy*. Suivant la terminologie Java Remote Method Invocation, les actions de l'application, sous forme de métadonnées, sont diffusées par les noeuds *mandataires* du FE. Ces métadonnées sont reçues par des noeuds *squelettes* des BE qui invoquent des méthodes sur les noeuds d'*implantation* (figure 2).

Ce modèle de mandataire distant est optimisé par des accesseurs locaux, en maintenant, sur le FE, un metagraphe de scène qui contient une copie locale des metadonnées. Les opérations de lecture restent alors confinées au FE. Seules les opérations d'écriture sont distribuées : sur demande du FE, les données sont changées sur le BE et une fois les modifications effectuées, le FE met à jour ses métadonnées correspondantes. Remplacer la duplication du graphe de scène sur le FE par une mise à jour d'un metagraphe de scène pourrait permettre de déployer ce dernier sur des clients légers de type PDA ou mobile.

4.1.2. Envoi des actions comme messages

La figure 5 propose une vue globale de la couche de distribution de données. Les étapes 1 et 2 décrivent la transformation des actions en ensemble de métadonnées, regroupées ensuite dans un message. Ces métadonnées obéissent à une synchronisation écrivain unique/lecteurs multiples. Ceci pourrait poser problème lorsque les BE doivent remonter des données vers le FE, par exemple lors d'un pointage 3D. Cependant la synchronisation du metagraphe permet d'effectuer une telle opération. En effet, une métadonnée indiquant si le noeud a été pointé pourra être ainsi mise à jour par les BE, répercutée ensuite sur le FE.

Les messages sont convoyés par une librairie iOS. Reprenant les concepts de la librairie Java Message Service, elle propose un protocole basé sur des boîtes aux lettres et des canaux de transmission. Ces communications utilisent un canal fiable UDP pour des données critiques pour lesquelles la réception est garantie. Un canal non-fiable UDP est réservé à des informations de faible sémantique ou non critiques. Un ordre strict de transmission basé sur un horodateur assure la consistance des données dans la queue des données reçues.

4.2. Serveur de rendu

4.2.1. Traduire les messages en actions

Le BE traduit en actions, les messages des noeuds mandataires du FE. La figure 5 illustre comment les noeuds squelettes appliquent ensuite ces actions aux noeuds d'implantation. Dans l'étape 3 , le reste du message est interprété par le noeud squelette du metaGS. Ce dernier ordonne ensuite au noeud d'implantation de mettre à jour à sa matrice (étape 4).

La correspondance entre les différents noeuds mandataire, squelette et implantation est établie à l'aide d'une signature commune à ces noeuds, calculée à partir des positions des noeuds dans le graphe de scène. Ces signatures sont initialisées au chargement de la scène puis mises à jour lors d'un ajout ou d'un déplacement de noeud.

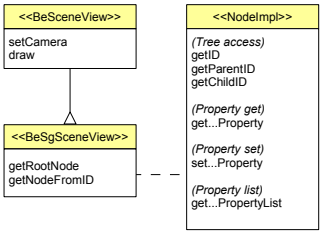


Figure 4: *Interfaces des serveurs de rendu BE. L'interface BeSceneView doit être implantée pour des rendus sans graphe de scène, autrement, BeSgSceneView et NodeImpl fournissent les interfaces requises pour un plugin nécessitant un graphe de scène distribué.*

4.2.2. Intégration dans un bureau distribué

Afin d'améliorer l'ergonomie vis à vis de l'utilisateur, HiD²RA permet d'utiliser un système de fenêtrage sur un mur d'affichage, suivant une implantation semblable à Chromium, à l'aide d'un serveur X window distribué, Distributed Multihead Server X (DMX) [MDF03]. Les entrées de l'application sont envoyées à une fenêtre de DMX, tandis que les sorties du BE sont envoyées vers une surface de rendu spécifique. Cette surface OpenGL est ensuite superposée à la fenêtre cible DMX.

5. Résultats

Notre grappe se compose d'une machine *frontale* et de 6 *serveurs de rendu*, composés par 2 Opteron 270 AMD64 bi-coeurs cadencés à 2Ghz avec des cartes graphiques NVidia GeForce 4500 associées à des cartes de synchronisation G-sync. Elles sont reliées à 6 projecteurs DLP F1+ avec une résolution 1400x1050. Un recouvrement de 320 pixels permet une transition douce entre les sous-surfaces d'affichage et rapporte la résolution à 3560x1780. La taille d'un pixel sur le mur d'affichage est alors d'un millimètre pour une surface totale de 3.6x1.8m.

Nous comparons HiD²RA avec deux solutions sort-first, ClusterJuggler et OpenSG, et deux hybrides sort-first/sort-middle, Chromium et Flow-VR. Cette comparaison est etayée par une étude de montée en charge en résolution et en trafic réseau. Un serveur de rendu a été dédié à ces tests, basé sur un graphe de scène disponible sous licence libre : OpenSceneGraph (OSG). Ce BE permet d'ombrer une scène à l'aide d'une carte d'ombre. Bien qu'ancienne [Woo92], cette méthode n'est pas souvent appliquées sur des murs d'affichage. Cette technique soulignera la différence de trafic entre des solutions sort-first et sort-middle. Nos expériences avec OpenSG et une comparaison précédente avec ClusterJuggler [SWNH03] nous ont permis de tirer certaines conclusions pour notre comparatif de solutions.

Nous travaillons aussi sur TinySG, fournissant une meilleure qualité de rendu à l'aide de fonctionnalités avancées : un rendu des couleurs haute dynamique, des matériaux réalistes et des effets d'éclairage comme des ombres, des réflexions/réfractions spéculaires et un effet de lueur diffuse (glow) (figure 1). Notre bibliothèque de distribution a été également employée avec un moteur de rendu fourni par un tiers, ONESIA, comportant des animations rigides de scène et des effets avancés d'éclairage basés image, utilisant le graphe de scène Ogre (figure 10).

nb triangles	local (i/s)	HiD²RA	ClusterJuggler	Chromium
dragon 200k	400 ;180	350 ;170	330 ;154	250 ;7
Fiat500 600k	340 ;120	320 ;100	220 ;90	220 ;7
Megane 1M	140 ;53	130 ;40	- ;-	- ;-
Scene 1M	160 ;64	150 ;60	150 ;60	- ;-

Figure 6: *Coût de la distribution et complexité géométrique sur une mosaïque d'affichage 3x2. Les fréquences d'affichage sont données en images par secondes pour des rendus de scènes non-ombragées puis ombragés.*

5.1. Tests de performance

Nous étudions la montée en charge en résolution d'affichage et en trafic réseau pour des scènes relativement différentes, de 200k à 1M de polygones sur 1,2,3,4 puis 6 écrans.

5.1.1. Montée en résolution

Seule la complexité géométrique est prise en compte pour ces tests. Elle permet d'illustrer l'impact d'une géométrie distribuée par rapport à une géométrie dupliquée. Les scènes non-ombragées et ombragées sont affichées sur un écran simple de station de travail et sur des configurations de {2,3,4 et 6} écrans.

Les scènes de la figure 6 sont composées d'un nombre croissant de polygones afin de tester la montée en charge en terme de taille de données. Ainsi, sur des scènes fortement maillées, nous avons dépassé le temps imparti à l'initialisation de ClusterJuggler et nous avons mis en défaut la migration de la géométrie sur Chromium, lors de mouvement trop brusques de caméra. OpenSG proposant un chargement progressif de la scène, ce dernier ne devrait pas rencontrer de tels problèmes.

L'*efficacité*, définie comme un rapport de performances distribution/local, estime l'impact de la distribution pour différentes surfaces de visualisations (figure 7-1). La fréquence d'affichage de Chromium diminue linéairement. Cette perte d'efficacité est d'autant plus mise en lumière pour une scène ombragée (figure 7-2). Au contraire, les approches sort-first, indépendantes vis à vis du nombre d'écrans, offrent une meilleure efficacité. On distingue alors deux facteurs limitant la fréquence d'affichage, liés à la carte graphique (60 images par secondes) et à la synchronisation logicielle (autour 300 i/s). Dans le premier cas, notre BE et ClusterJuggler ont des performances et nous pouvons nous attendre à des temps d'exécution similaires pour OpenSG [SWNH03]. Dans le second cas, la majeure partie du temps est allouée à envoyer des messages de synchronisation des données. L'étude suivante sur le débit du réseau révèle quelques différences.

5.1.2. Montée en charge et débit du réseau

5.1.2.1. Distribution des données Un analyseur de réseau, Etherape [TvA06], permet d'accumuler le *trafic* du réseau. Intuitivement, plus une scène est simple, plus sa fréquence d'affichage est importante, et plus le débit réseau est grand.

FlowVR-R et Chromium améliorent les solutions sort-middle en employant un cache de géométrie sur la carte

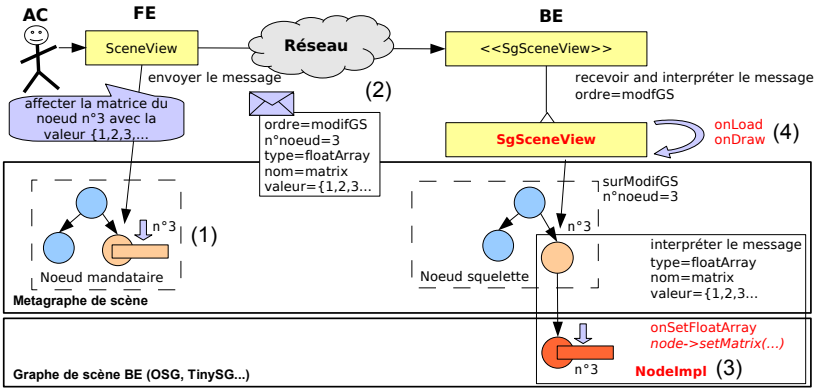


Figure 5: *Implantation de la distribution des données*

graphique, et évitent ainsi un flux constant de la géométrie depuis les unités centrales. Cependant, les performances peuvent chuter lors d'une migration de la géométrie, pour des scènes animées ou pour des mouvements rapides de caméra.

Au contraire, les méthodes sort-first n'envoient que des ordres de synchronisation. Sans aucune mise à jour du graphe, HiD²RA n'utilise que 266 octets par images pour les ordres de rendu et la position de la caméra (13 champs de 13 octets). Ces messages sont envoyés d'un frontal unique à n serveurs de rendu, contrairement à ClusterJuggler qui peut transmettre jusqu'à $n(n-1)$ messages (maillage des BE pair-à-pair).

5.1.2.2. Rendu multipasse Pour une solution sort-middle, le débit réseau est bien plus important pour un rendu multipasse (rendu vers une texture) : le débit théorique requis pour une carte d'ombre est de 1024x1024x24bit, soit 1.5Mo par image. Une solution comme Chromium impose un transport incessant des fragments (figure 7-3). Les tampons de fragments sont traités sur chaque BE. Ils sont ensuite assemblés pour former un tampon qui correspond à la totalité du mur d'affichage. Ce tampon est finalement renvoyé vers les BE[§]. Au contraire, HiD²RA et ClusterJuggler évitent ce flux de pixels en dupliquant les traitements sur chaque BE. A l'heure de la rédaction, la scène entière est rendue du point de vue de la lumière, ce qui explique pourquoi l'efficacité de la distribution reste autour de 1 (table 6). Elle pourrait être améliorée en distribuant le calcul de la carte d'ombre plutot que de le dupliquer .

5.1.2.3. Scène dynamique Nous avons mené toutes nos expériences sur des applications de navigation dans des scènes statiques. Pour une scène animée, nous pouvons distinguer deux cas. Une animation rigide ou une déformation précalculées ne nécessitent qu'un envoi de message de synchronisation entre les BE, ces derniers déroulant chaque animation de manière locale. Autrement, une animation non déterministe, impulsée par un utilisateur ou un moteur physique, nécessite plus de bande passante. Les données de mise à jour convoyées depuis l'application ou un moteur d'animation comme OpenMask [MAC*02] peuvent alors être transportées par des métadonnées.

Dans tous les cas, les solutions sort-first ont de meilleurs performances que celles en sort-middle. Ces solutions sort-first peuvent apporter des réponses différentes à ces problèmes d'animation. OpenSG propose un système de chargement progressif de la scène avec une migration efficace de sa géométrie et une élimination précoce des objets non visibles depuis la caméra. A l'opposé, HiD²RA et ClusterJuggler supposent que les BE ont assez de puissance en terme de calcul et de mémoire pour supporter une duplication de scène. De la même manière, l'élimination des objets cachés est déleguée aux graphes de scène des BE. Cependant, si OpenSG peut offrir de meilleurs temps d'exécutions dans des scènes statiques, ces derniers peuvent également empirer pour des scènes dynamiques. Les améliorations fournies par des mises à jour parallèles sont alors masquées par des goulots d'étranglement de réseau [SWNH03]. Notre solution permet de conserver les mêmes performances que l'on soit sur une station de travail ou sur un mur d'écran.

5.2. Evolutivité

5.2.1. Fonctionnalités OpenGL

Pour l'instant, seul Chromium est limité dans le support des fonctionnalités OpenGL 2.0, requises pour un rendu de qualité en environnement distribué. Contrairement à ce dernier, FlowVR peut alimenter une sortie vers un tampon flottant. Cependant, c'est avec difficulté que nous avons essayé de faire évoluer FlowVR-R. Mettre en application un algorithme multipasse exige ainsi de modifier la boucle de rendu à l'intérieur de ce cadre d'application. Ses propres shaders occupant déjà le processus de rendu sur la carte graphique, cela empêche une inclusion directe de shaders utilisateur.

5.2.2. Migration de graphes de scène existants

Une application écrite avec notre interface de programmation peut être réutilisée avec tous nos plugins de serveur de rendu. Ceci peut récompenser l'effort nécessaire pour réécrire une application existante. Dans ClusterJuggler, le code de distribution et le code de rendu sont étroitement liés. En pratique, chaque nouvelle modification induite par l'AC doit

§ Une unité de traitement spécifique (SPU) [HHN*02] pourrait empêcher ces allers-retours mais briserait la compatibilité OpenGL.

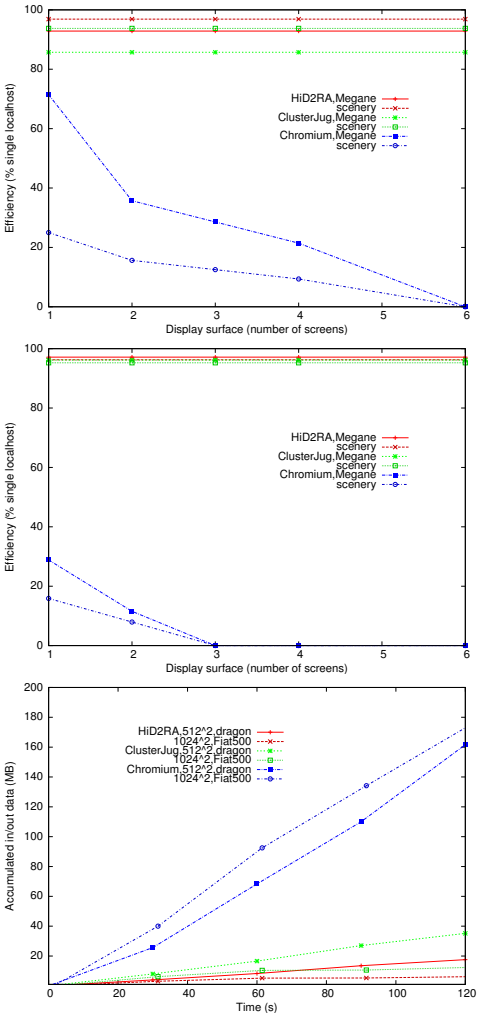


Figure 7: *De haut en bas : fréquence d'affichage en fonction de la surface d'affichage pour des scènes non-ombragées puis ombragées ; Débit réseau en fonction de la surface d'affichage sur une scène ombrée- en rouge HiD²RA, en vert, ClusterJuggler et en bleu, Chromium*

être propagée jusqu'aux BE. Avec HiD²RA, la couverture exhaustive offerte par le metaSG permet d'éviter de répercuter le code de distribution dans les BE. Cependant, notre interface de programmation possède également ses propres inconvénients. Les propriétés de méta-données invalides (désignant un mauvais membre ou un mauvais type de valeur) sont, pour l'instant, repérées seulement lors de l'exécution.

Pour le serveur de rendu sans graphe de scène ou pour un graphe statique, seules deux fonctions doivent être écrites. ONESIA a ainsi déployé en une demi-journée, son moteur de rendu basé sur Ogre. Pour finir, contrairement à ClusterJuggler, la boucle de rendu est exposée à l'application, fournissant ainsi une interface de programmation plus simple.

Pour un serveur de rendu exigeant un graphe de scène distribué, comparé à ClusterJuggler et à OpenSG, HiD²RA offre une distribution des données du graphe fortement abstraite. Les connexions entre le metagraphe et graphe BE

```
//créer un conteneur de données utilisateurs
class UserData : public vpr::SerializableObject...

// peer (BE+FE) ------------------
//init : associer la donnée à un identifiant unique
pour les droits d'accès
vpr::GUID monGuid("17109-15-17107-D34D");
mUserData.init(monGuid);                    ClusterJuggler

if(!mUserData.isLocal())
   // lire la donnée locale sur notre machine
   myMatrix=mUserData.getMatrix();
else
   // sinon l'envoyer vers un de ses pairs
   mUserData.setMatrix(myMatrix);
```

```
// la création du code du conteneur de champs
// UserData est assistée par une interface graphique
class UserData...

// FE ------------------------------------
// beginEditCP va déclencher la synchronisation
// des données entre les processus/serveurs OpenSG
beginEditCP(mUserData);
    mUserData->setMatrix(myMatrix);
endEditCP(mUserData);                       OpenSG

// BE ------------------------------------
myMatrix=mUserData->getMatrix();

//toute donnée éditée du côté serveur de rendu
//doit aussi être encadrée par des macros
begin/endEditCP
```

```
//UserData implante une interface BeSgNodeImpl
//ce code est centralisé dans la couche du
//metascenegraph
class UserData...
void onSetFloatArray(message,matrix)
{
//interprétation en action depuis le message reçu
if (message==« Matrix ») mMatrix=matrix;
}                                           HiD²RA

// FE ------------------------------------
mUserData->setFloatArray(« Matrix »,myMatrix);

// BE ------------------------------------
myMatrix=mUserData->getFloatArray(« Matrix »);
```

Figure 8: *Comparaison entre différents codes de distribution*

	ClusterJuggler	OpenSG	HiD²RA
Nom	données utilisateur	conteneur de champs	metadonnées
Fonctions	pair-à-pair	thread safe, optimal	haute niveau
Ergonomie -	ID unique, localisation	macros	compilation
Evolutivité	généricité + héritage	interface graphique	composition

Figure 9: *Comparaison des données distribuées*

peuvent être établies manuellement pour un graphe de scène simple, par exemple, pour TinySG. Autrement, une bibliothèque d'introspection peut automatiser ce processus. Nous avons ainsi utilisé le module osgIntrospection qui fournit un accès programmatique aux propriétés des noeuds de OpenSceneGraph. Dans la majeure partie des cas, ce travail est fait une fois pour toutes, puisque le metaGS fournit une couverture systématique du graphe de scène du BE. Dans certains cas, il pourrait être encore nécessaire d'étendre ce graphe avec des données d'utilisateur. Le paragraphe suivant compare quelques solutions.

5.2.3. Evolution avec des données distribués utilisateurs

FlowVR-R, OpenRM/Chromium et HiD²RA utilisent le passage de message, soit à travers MPI, soit à l'aide de classes iOS. Au contraire, ClusterJuggler et OpenSG offrent des fonctions plus sophistiquées comme la localisation ou le multiprocessus (voir le tableau 9). Ces fonctionnalités comportent de nombreux avantages mais aussi de nombreuses contraintes, en plus d'un inévitable mélange entre le code de rendu et de réseau. Les extraits de code de la figure 8 illustrent l'ergonomie du point de vue du développeur.

Dans ClusterJuggler, un noeud est à la fois un FE et un

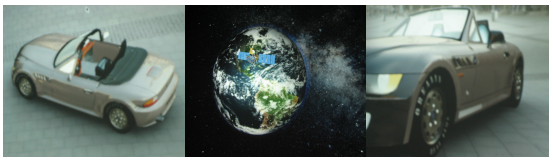


Figure 10: *Rendu haute qualité par un serveur de rendu tiers, supportant des animations rigides, de l'occlusion ambiante et de l'éclairage à partir de cartes d'environnement (scènes aimablement fournies par ONESIA)*

BE (structure de pair-à-pair). Le programmeur doit alors décrire les deux aspects suivants. La localisation (suivant qu'on soit sur le FE ou sur le BE), les droits d'accès aux données (par un identifiant unique associé aux pairs) doivent être clairement énoncés. A l'opposé, les BE et les FE sont clairement séparés dans HiD^{2}RA et OpenSG. Si ce dernier est une bonne solution pour le rendu distribué out-of-core, son ergonomie s'en trouve limitée puisque chaque donnée distribuée doit être encadrée par une macro commande. Dans notre solution, le metagraphe se charge de diffuser les données. Si des modifications sont toutefois apportées, elles restent confinées dans la classe *SgNodeImpl*.

6. Conclusion

Notre bibliothèque se compose de plusieurs couches afin de fournir un rendu distribué de qualité en temps réel, le plus transparent possible vis à vis du développeur et de l'utilisateur. Le metagraphe de scène sur le frontal est un mandataire distant permettant une séparation claire entre les algorithmes de rendu et la couche de distribution. Notre architecture modulaire permet de brancher des serveurs afin d'obtenir différents rendus. Elle facilite ainsi le développement d'applications clientes dans un cadre de développement délocalisé. Comparée à Chromium, OpenSG et VRJuggler, seule notre bibliothèque offre à la fois un rendu avancé à des fréquences d'affichage élévées, combiné avec une distribution des données de graphe de scène évolutive.

Jusqu'ici, notre priorité était de fournir l'épine dorsale de la distribution. Or, dans notre contexte, les performances globales dépendent principalement du serveur de rendu. Afin d'améliorer ses performances, des travaux futurs pourraient se focaliser sur une élimination précoces de parties cachées et une distribution de charge à travers la grappe. Cette dernière étape peut permettre la distribution d'algorithmes de rendu avancés et faire des murs d'affichage, un environnement intéressant pour des activités de prototypage et de design industriel.

References

[AKKN01] ARSENAULT L. E., KELSO J., KRIZ R. D., NEVES F. D. : *DIVERSE : A software toolkit to integrate distributed simulations with heterogeneous virtual environments.* Tech. Rep. 01-10, Department of Computer Science, Virginia Tech., 2001.

[AR05] ALLARD J., RAFFIN B. : A shader-based parallel rendering framework. In *16th IEEE Visualization Conference (VIS 2005), 23-28 October 2005, Minneapolis, MN, USA* (Minneapolis, USA, October 2005), IEEE Computer Society, p. 17.

[BRR02] BEHR G. V. J., REINERS D., ROTH M. : A multi-thread safe foundation for scene graphs and its extension to clusters. In *EGPGV '02 : Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization* (2002), Eurographics Association, pp. 33–37.

[dPGaGZ04] DE PAIVA GUIMARÃES M., GNECCO B. B., ZUFFO M. K. : Graphical interaction devices for distributed virtual reality systems. In *VRCAI '04 : Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry* (New York, NY, USA, 2004), ACM Press, pp. 363–367.

[dSRG*02] DER SCHAAF T. V., RENAMBOT L., GERMANS D., SPOELDER H., BAL H. : Retained mode parallel rendering for scalable tiled displays. *Proc. 6th annual Immersive Projection Technology (IPT) Symposium, Orlando Florida* (March 2002).

[FJ79] FUCHS H., JOHNSON B. W. : An expandable multiprocessor architecture for video graphics (preliminary report). In *ISCA '79 : Proceedings of the 6th annual symposium on Computer architecture* (New York, NY, USA, 1979), ACM Press, pp. 58–67.

[FM05] FUNG J., MANN S. : Openvidia : parallel gpu computer vision. In *MULTIMEDIA '05 : Proceedings of the 13th annual ACM international conference on Multimedia* (New York, NY, USA, 2005), ACM Press, pp. 849–852.

[Fuc77] FUCHS H. : Distributing a visible surface algorithm over multiple processors. In *1977 ACM Annual Conference, Seattle, WA* (October 1977), pp. 449–451.

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J. : *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley, 1995.

[HEB*01] HUMPHREYS G., ELDRIDGE M., BUCK I., STOLL G., EVERETT M., HANRAHAN P. : WireGL : a scalable graphics system for clusters. In *SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 129–140.

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., SEAN AHERN P. K., KLOSOWSKI J. T. : Chromium : A stream processing framework for interactive rendering on clusters. *In Proceedings of SIGGRAPH 2002* (2002), 693–702.

[HM99] HEIRICH A., MOLL L. : Scalable distributed visualization using off-the-shelf components. *IEEE Parallel Visualization and Graphics Symposium* (October 1999), 55–59.

[HSFP99] HESINA G., SCHMALSTIEG D., FURHMANN A., PURGATHOFER W. : Distributed open inventor : a practical approach to distributed 3d graphics. In *VRST '99 : Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 1999), ACM Press, pp. 74–81.

[JBBCN98] JUST C., BIERBAUM A., BAKER A., CRUZ-NEIRA C. : VR juggler : A framework for virtual reality development. In *2nd Immersive Projection Technology Workshop (IPT98), Ames, Iowa* (May 1998).

[JJR*05] JEONG B., JAGODIC R., RENAMBOT L., SINGH R., JOHNSON A., LEIGH J. : Scalable graphics architecture for high-resolution displays. In *IEEE Information Visualization Workshop 2005* (2005).

[MAC*02] MARGERY D., ARNALDI B., CHAUFFAUT A., DONIKIAN S., DUVAL T. : OpenMASK : Multi-threaded or modular animation and simulation kernel or kit : a general introduction. *VRIC 2002 Proceedings, S. Richir, P. Richard, B. Taravel (eds.)* (2002), 101–110.

[MCEF94] MOLNAR S. E., COX M., ELLSWORTH D. A., FUCHS H. : A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications 14*, 4 (July 1994), 23–32.

[MDF03] MARTIN K. E., DAWES D. H., FAITH R. E. : Dmx, distributed multihead X design, dmx.sourceforge.net, July 2003.

[MF98] MACINTYRE B., FEINER S. : A distributed 3D graphics library. In *Proceedings of SIGGRAPH 98* (1998), Cohen M., (Ed.), Addison Wesley, pp. 361–370.

[NSG05] NAEF M., STAADT O. G., GROSS M. : Multimedia integration into the Blue-C API. *Computers & Graphics 29*, 1 (2005), 3–15.

[Ols02] OLSON E. : Cluster juggler - PC cluster virtual reality, Master thesis, Iowa State University, 2002.

[PJ01] PERRINE K. A., JONES D. R. : Parallel graphics and interactivity with the scaleable graphics engine. In *Supercomputing '01 : Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)* (New York, NY, USA, 2001), ACM Press, pp. 5–13.

[Sch00] SCHAEFFER B. : A software system for inexpensive VR via graphics clusters, www.isl.uiuc.edu/clusteredvr/paper/dgdoverview.htm, 2000.

[SML98] SCHROEDER W., MARTIN K. M., LORENSEN W. E. : *The visualization toolkit (2nd ed.) : an object-oriented approach to 3D graphics.* Prentice-Hall, Inc., Upper Saddle River, NJ, 1998.

[SWNH03] STAADT O. G., WALKER J., NUBER C., HAMANN B. : A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *EGVE '03 : Proceedings of the workshop on Virtual environments 2003* (New York, NY, USA, 2003), ACM Press, pp. 261–270.

[Tra99] TRAMBEREND H. : Avocado : A distributed virtual reality framework. In *VR '99 : Proceedings of the IEEE Virtual Reality* (1999), IEEE Computer Society, pp. 14–21.

[TvA06] TOLEDO J., VAN ADRIGHEM V. : EtherApe, a graphical network monitor, etherape.sourceforge.net, 2006.

[Woo92] WOO A. : The shadow depth map revisited. In *Graphics Gems III* (1992), Academic Press Professional, Inc., pp. 338–342.